Section 3 Guide

Background

In Section 2, we created the Top Score system using an array of scores to show the Top 3 players. Since we had not learned about While-loops or For-loops yet, we had to press “Run Continuously” in order to play the game. This is not a good coding practice. Also, we want to have a more interactive experience for the user so we will be using pop-ups to guide the user through the game.

Task

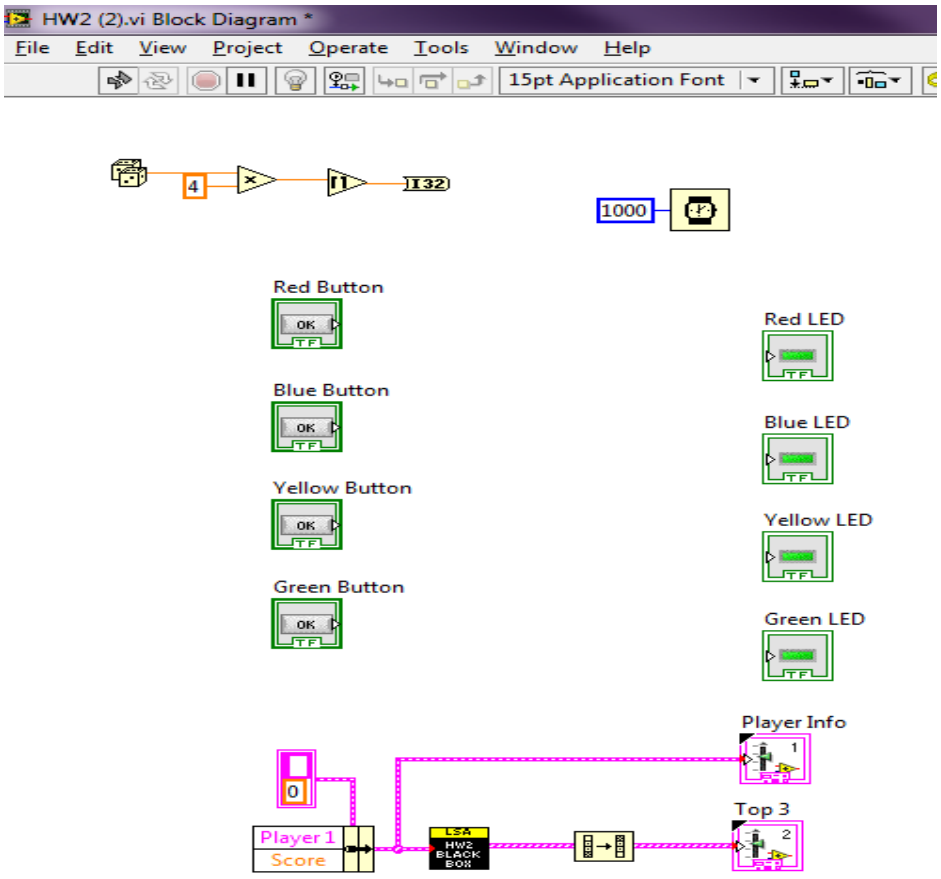
Taking the VI from the previous section, we want to insert a while-loop to keep the game running, a for-loop to display the LED sequence, and another for-loop to get the user’s input as they try to replicate the LED sequence. We will also be using case structures to determine which LEDs should turn on, and when to stop the game (when a timeout has occurred or user lost the game). Black Box subVI will still be used to keep the high scores list.

This Black Box subVI receives a cluster with the Name of the Player and the Score and will give back an array with all the previous plus the current Player Information and it sorts them from lowest score to highest score.

Steps

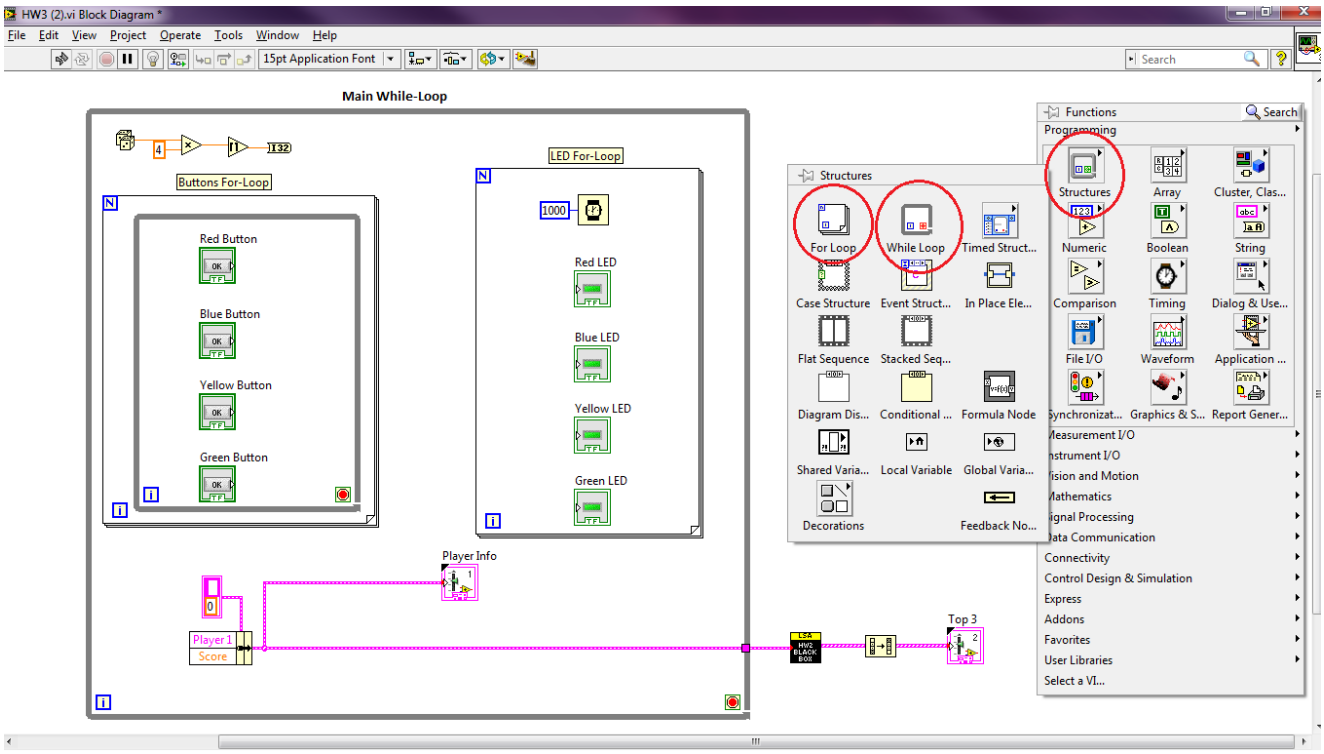
This part of the Guide provides you step-by-step instructions of how to obtain the expected results of this section. Feel free to follow all steps, skip some, or do even it by yourself and compare your solution at the end with the steps, filling in any pieces you may have missed. To maximize your learning experience, we suggest choosing the latter method.

1. On the block diagram, remove code so that all you have left is this:



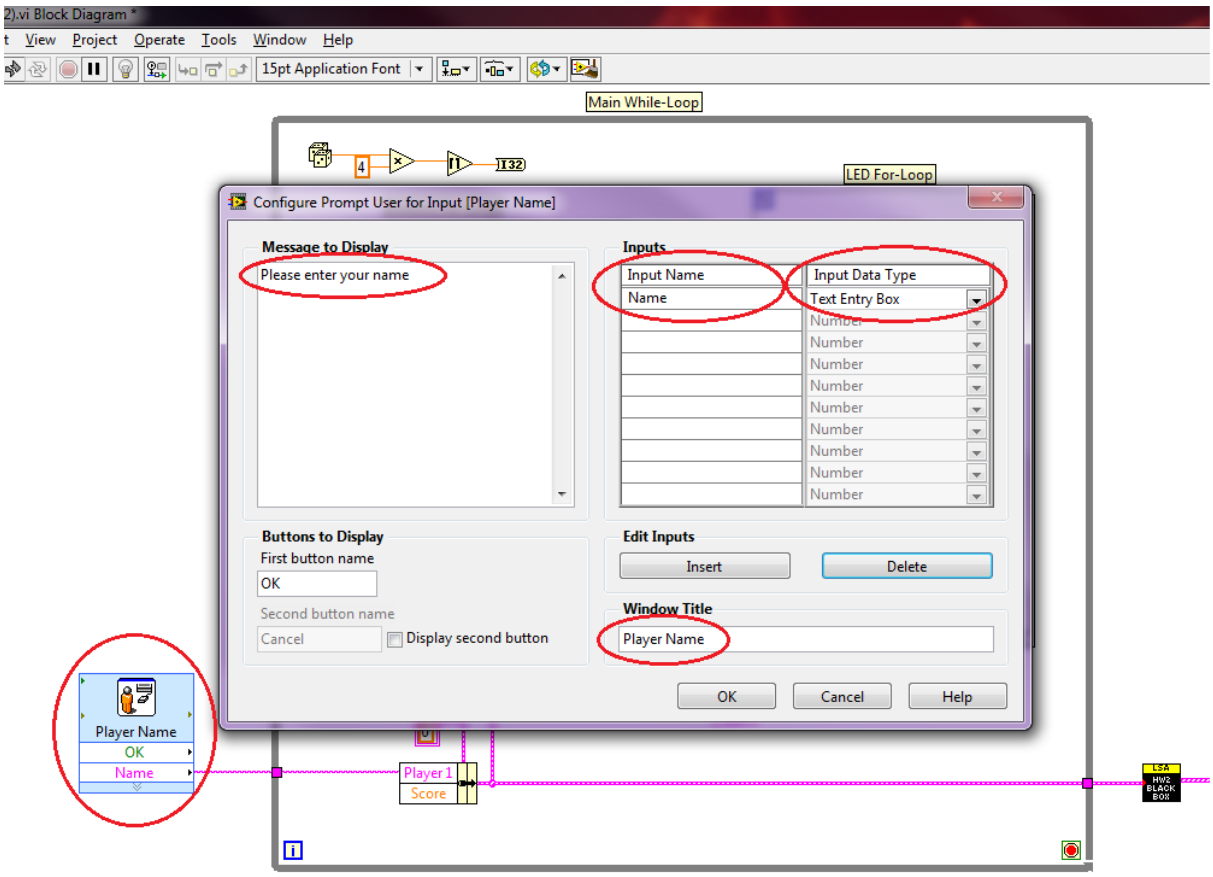
We only included the previous parts so you could familiarize yourself with LabVIEW and to make each section end with a VI that could run.

2. Now we will place portions of our code into loops. All loops can be found by: *Right Click > Functions Palette > Programming > Structures*. Place the LEDs, and timing code in a **for-loop** (we'll refer to this as the **LED for-loop**). Place the buttons inside of a **while-loop**, and place that while-loop inside of a **for-loop** (we'll refer to this as the **Buttons for-loop**). Now, place everything that is on the block diagram (except BlackBox subVI, reverse 1D array function, and Top 3 indicator) inside of a **while-loop** (we'll refer to this as the **Main while-loop**).. Your code should now look something like this:

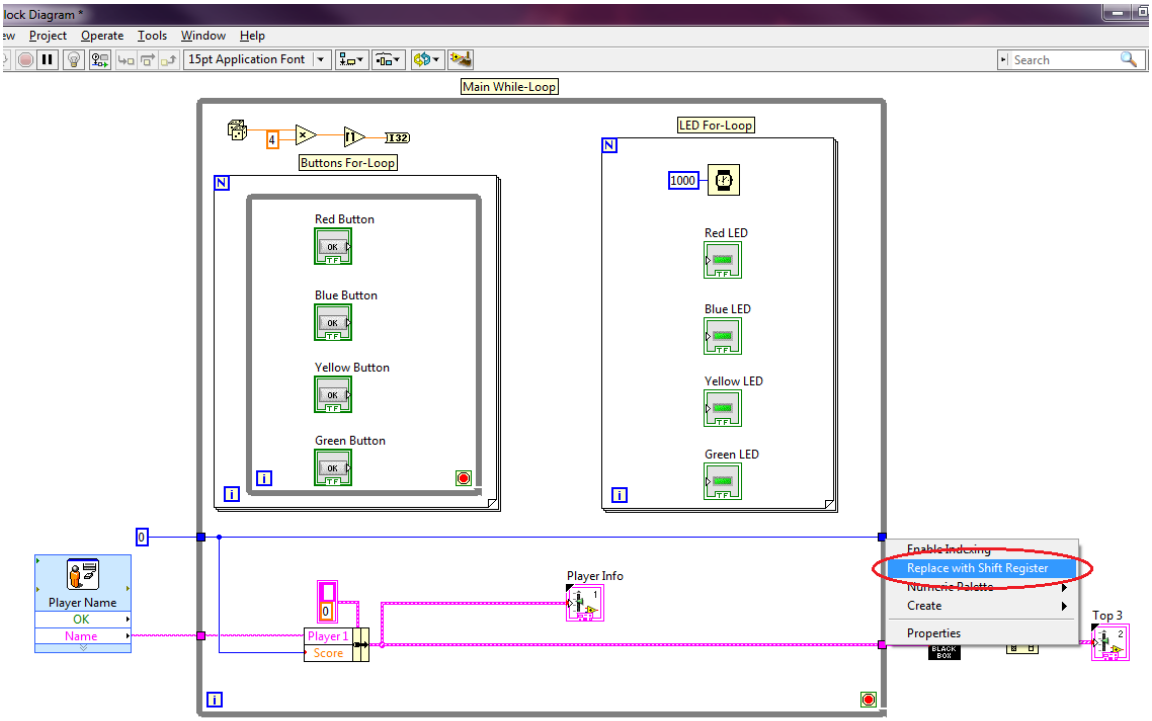


Main While-Loop

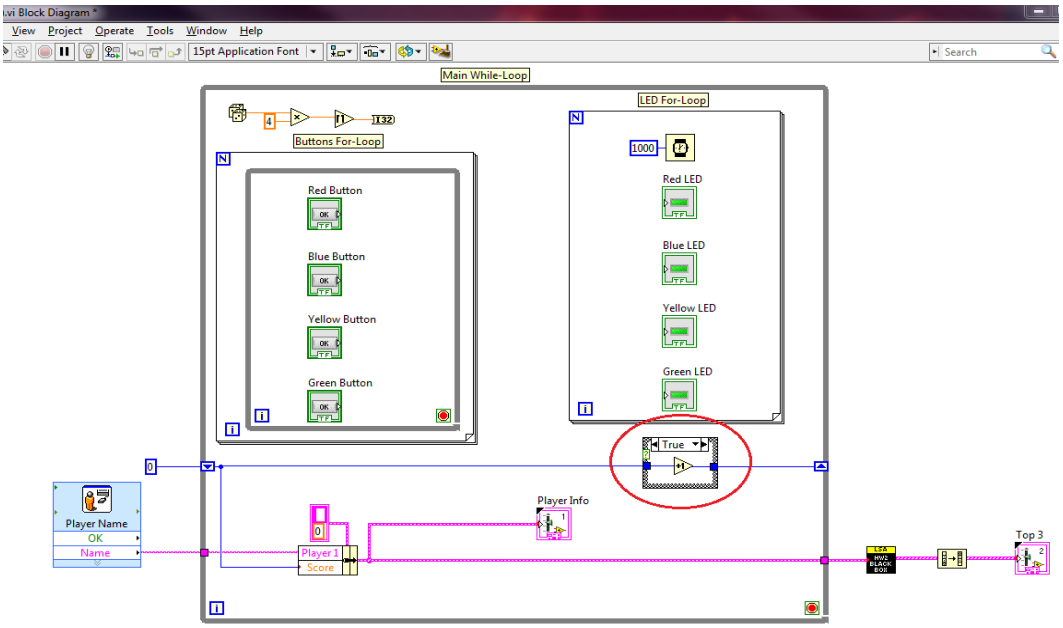
3. Place a **Prompt User for Input** (*Right Click > Functions Palette > Programming > Dialog and User Interface > Prompt User for Input*) function before (to the left and outside of) the **Main while-loop**. It should ask the user to input their Name (data type Text Entry Box). Do this by specifying the input data type to be a text entry box and label it 'Name' in the pop up that appears when you first put the express VI down. If you need to reopen this box you can do so by double clicking on the function. The Name output of this box should then be connected to the "Player 1" terminal of the Bundle by Name Player Info cluster.

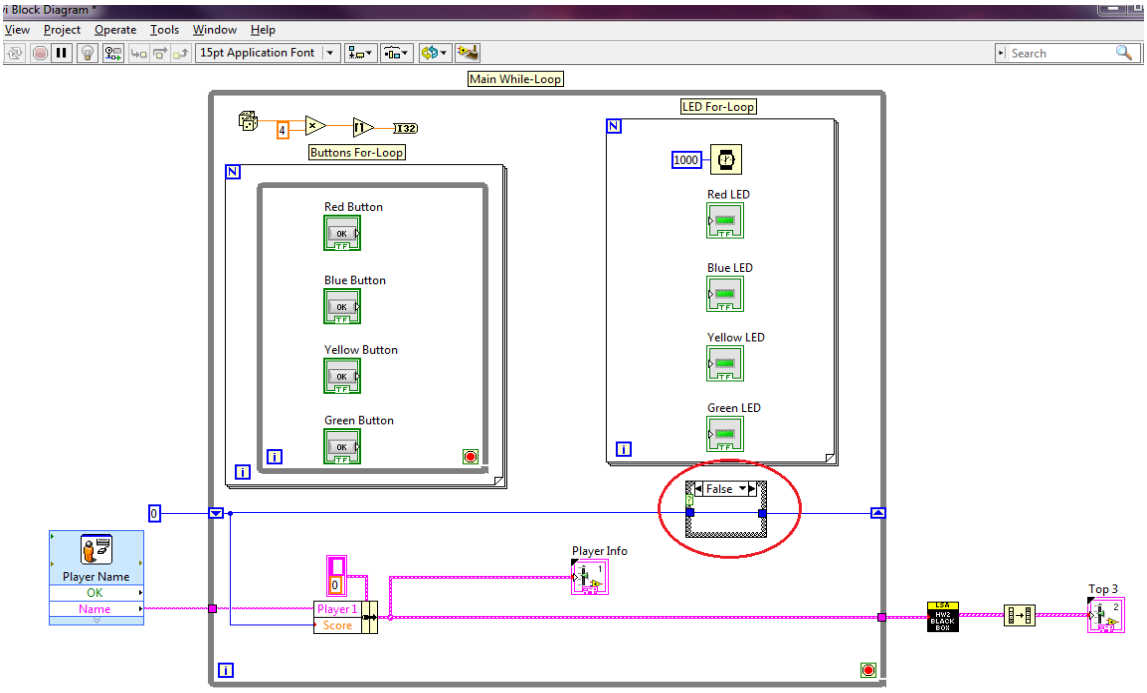


4. In order to keep track of the player's score through each iteration of the **Main while-loop,** a shift register is needed. Score will also go into the "Score" terminal of the Bundle by Name Player Info cluster. The Score shift register should be initialized to zero at the start of the game. To do this place a number constant (*Right Click > Functions Palette > Programming > Numeric > Numeric Constant*) on the block diagram and then wire it as shown below. To create the shift registers, right click a score terminal on the while loop and select 'Replace with Shift Register'. The terminal you clicked on will instantly become and shift register while your mouse will have a blacked out shift register attached to it. Simply take your mouse and click on the other terminal and turn that terminal into a shift register as well.



5. Add a True/False case structure inside of the **Main while-loop** (will be the last structure to execute within the while-loop) that will either increment the score (user got sequence right), or end the while-loop (user got sequence wrong or timeout occurred). We'll get back to what goes into the selector terminal in later instructions. To add the case structure *Right Click > Functions Palette > Programming > Structures > Case Structure.* To add the increment *Right Click > Functions Palette > Programming > Numeric > Increment.* An example of what this should look like can be seen below. Connect the score from the first shift register through true case to the increment function and connect its output to the right shift register, and through the false case.

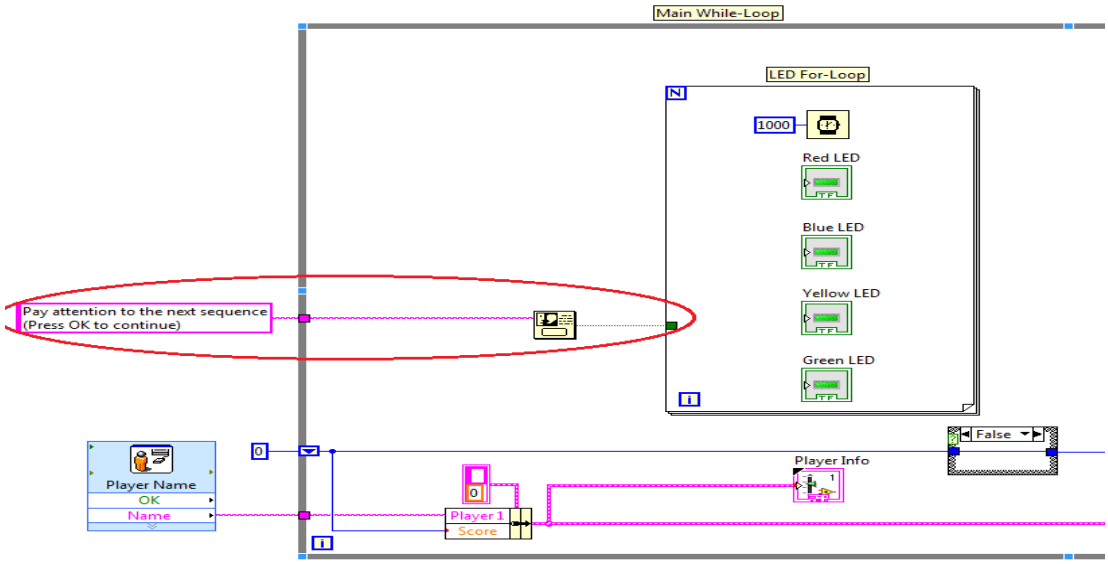




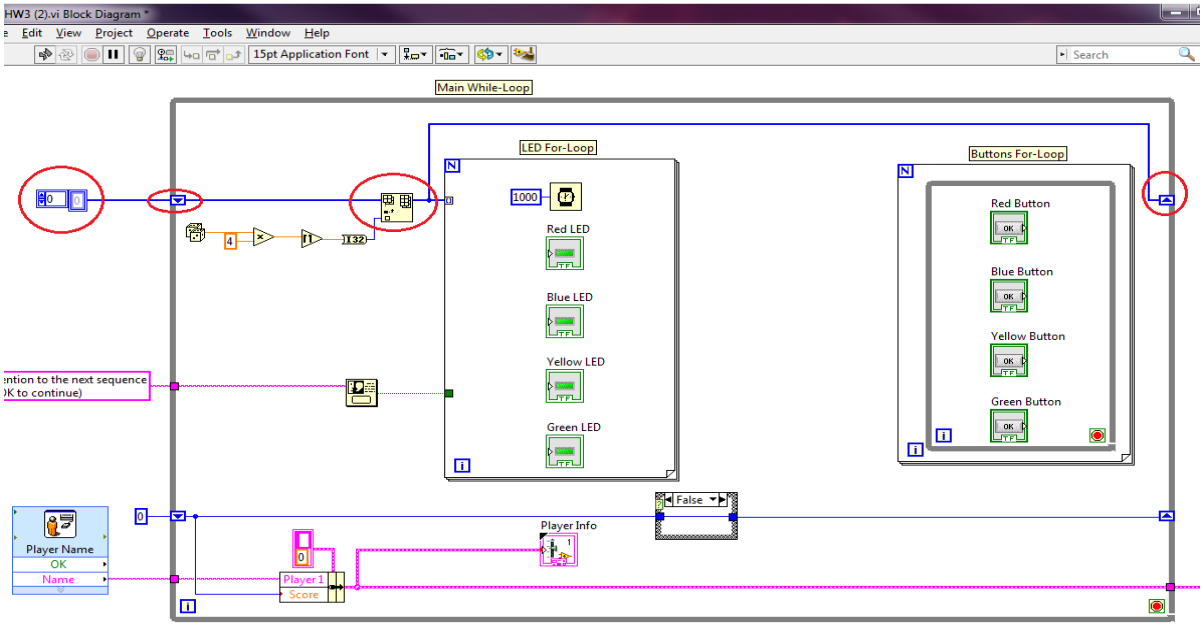
6. Move the **LED for-loop** over to the left (outside) of the **Buttons for-loop** so there is room to add code.

7. In order to alert the user that they should pay attention to the sequence, add a One-Button dialog box inside of the **Main while-loop** (will be the first function to execute within the **Main while-loop**). (i.e. text should say something like “Pay attention to the next sequence (Press OK to continue)”. Put the string that determines the message of the one button dialog box outside

of the while loop and connect the output of the one button dialog box to the LED for-loop to make sure that the dialog box will execute first. It is important to remember that portions of LabVIEW code will not execute until it has all required inputs. The string constant can be found here: *Right Click > Functions Palette > Programming > String > String Constant.* The one button dialog can be found here: *Right Click > Functions Palette > Programming > Dialog and User Interface > One Button Dialog.*

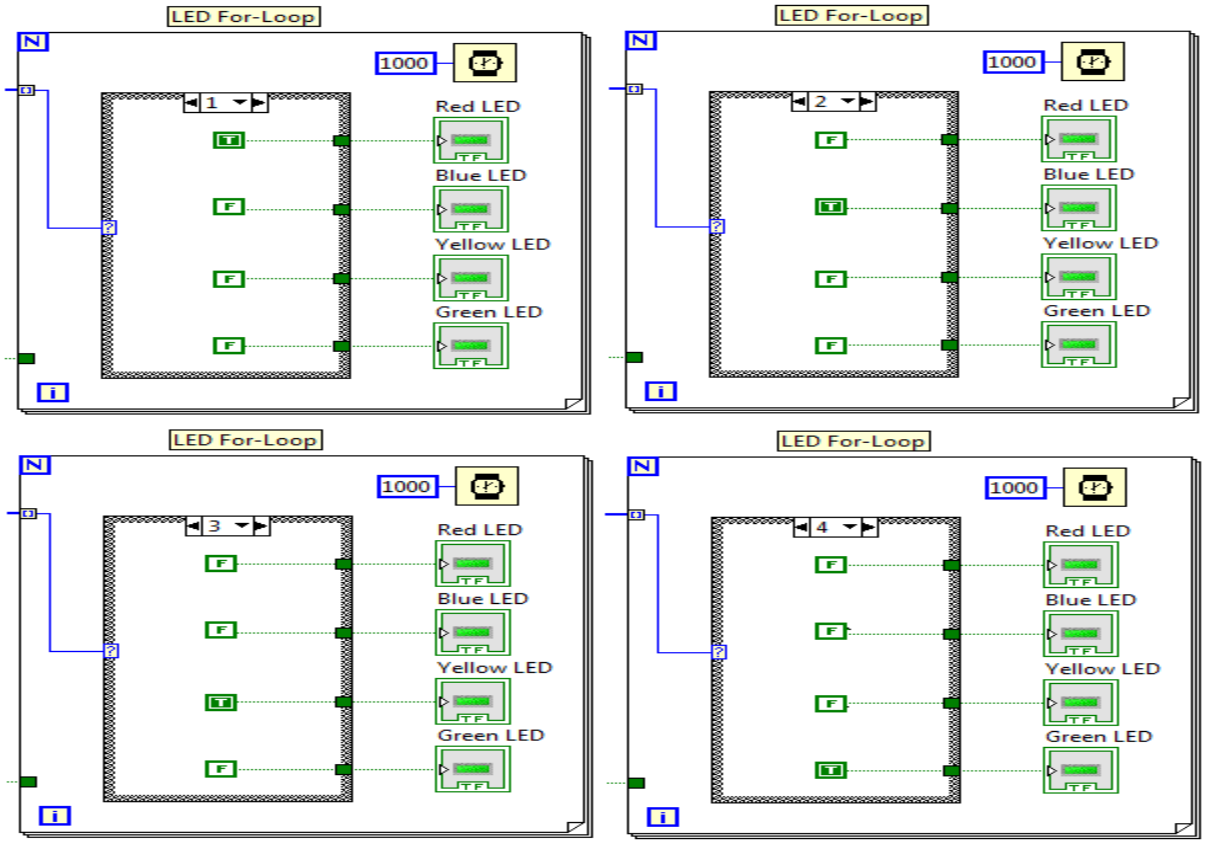


8. Every iteration of the main while loop, a new random number should be generated to add to the existing LED sequence (if any). We should keep the LED sequence in a numeric array of type U32. Use a shift register to carry the array through the **Main while-loop** iterations. Use an **Insert Into Array** function (*Right Click > Functions Palette > Programming > Array > Insert Into Array*) to add the new LED number into the existing sequence. The LED sequence array should be initialized to be empty before the start of the **Main while-loop**. This means that it should be an empty array constant (*Right Click > Functions Palette > Programming > Array > Array Constant)*. Specify the array of type numeric by adding a numeric constant into the empty array constant shell. You do not have to wire anything to the index terminal. Note that there will be some shift registers needed to feed the array back around between iterations.

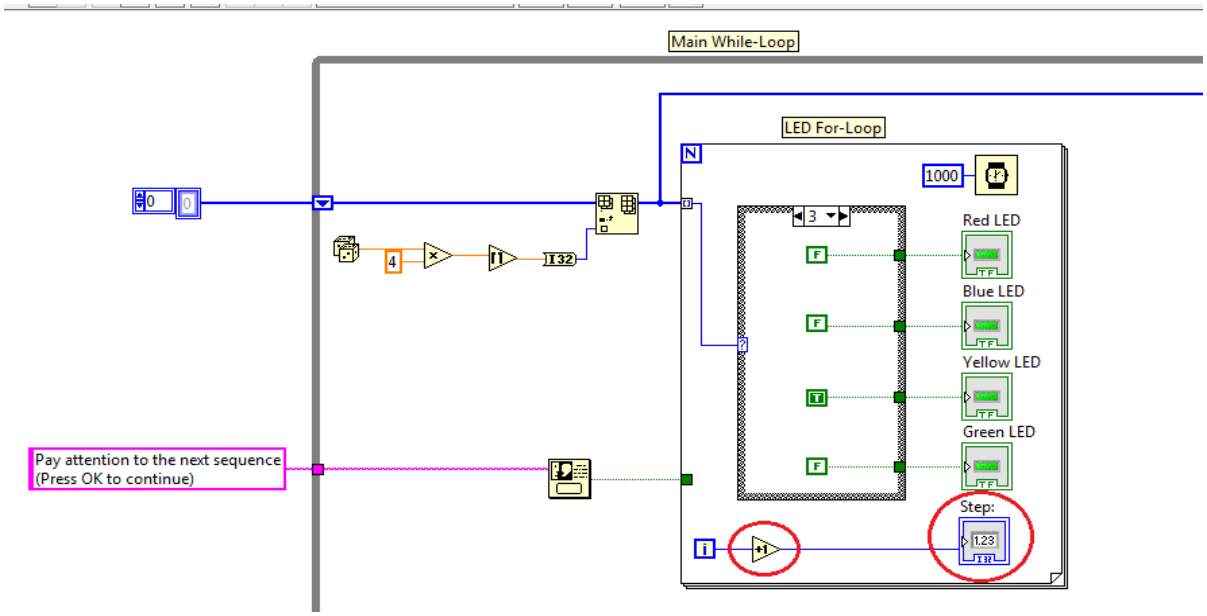


LED For-Loop (Displaying LED sequence)

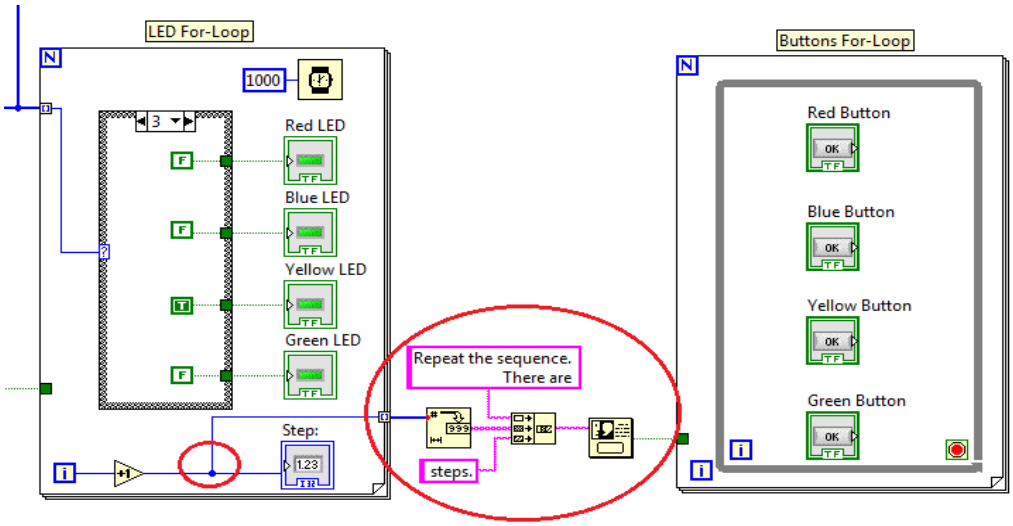
9. Make sure the **LED for-loop** is to the left of the **Buttons for-loop** so there is room to add code.
10. To specify how many times the **LED for loop** should execute (how many steps in the LED sequence), the LED sequence array should be connected to the left edge of the **LED for-loop**, with indexing enabled. This will allow the array size to control the number of iterations of the for-loop. You may have already done this step since it is pictured above.
11. In order to light up the correct LED to show the user the LED sequence, we will need to create a new case structure with cases for each possible value (1, 2, 3, 4). Connect the current iteration's U32 LED sequence integer to the selector terminal of the case structure. By this we mean connect the indexed array input to the for-loop to the input terminal of the case statement. Each case should have four Boolean constants (*Right Click > Functions Palette > Programming > Boolean > True/False Constant)* corresponding to the LEDs on the outside of the case structure. Each case will have only one True Constant, the rest False, depending on which color LED you want to light up during that case. You may need to add cases after case one and delete case zero to there are 1-4 cases. Do this by right clicking on the drop down portion of the case statement. Cases 1 through 4 can be seen below. *Make sure one of the cases is selected to be the default case since case zero was deleted, otherwise your VI won't run.* This happens because there are technically more possible integers that can be wired in other than 1-4.



12. The user should be able to see what step number they are on in the LED sequence. The iteration terminal of the **LED for-loop** will start at zero. For the step number to start at 1, connect an increment function to the iteration terminal of the **LED for-loop** and create an indicator named "Step:" from the output of the increment function. You can make the indicator size and its font size larger on the front panel if you wish.

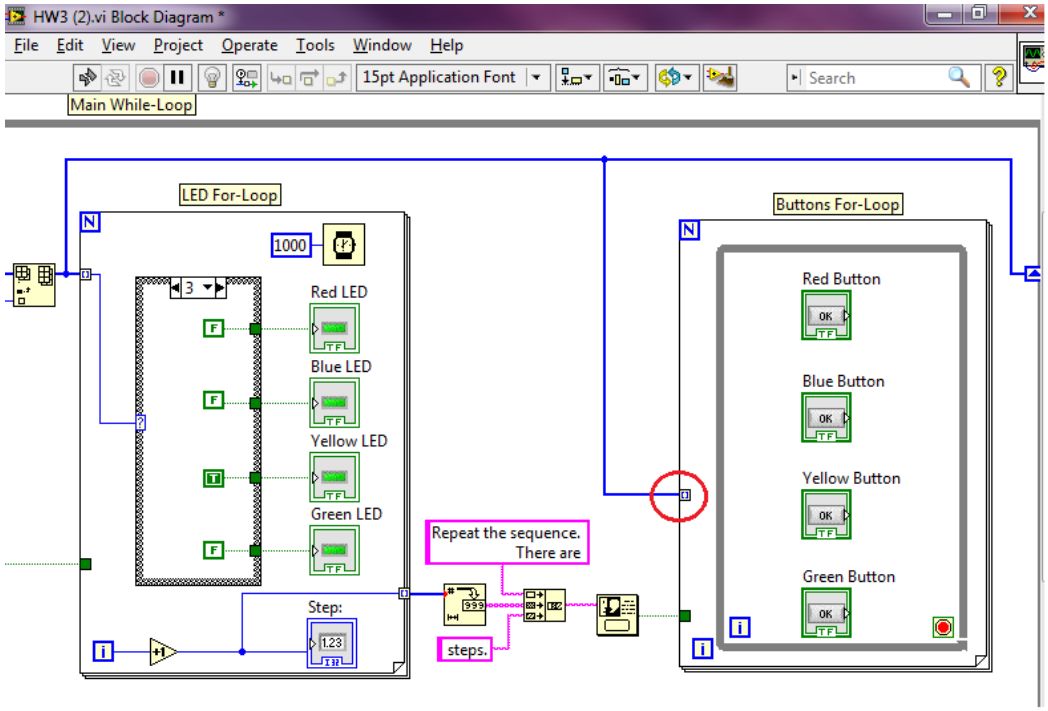


13. Add another one-button dialog box that will execute after the **LED for-loop** and before the **Buttons for-loop**. The message displayed should be "Repeat the sequence. There are [# of steps] steps". You can create this text by using the Concatenate Strings function (*Right Click > Functions Palette > Programming > String > Concatenate Strings)* and using the Number to Decimal String function (*Right Click > Functions Palette > Programming > String > String/Number Conversion > Number to Decimal String)*. Hint: there are three parts that get concatenated to create the desired string. Tie another wire to the output of the increment function in the **LED for-loop** and pass it out of the right side of the **LED for-loop** (disable indexing) and into the input terminal of the Number to Decimal String function. Remember, you moved the LED for-loop to the left of the buttons for-loop. Your block diagram should contain something like this:

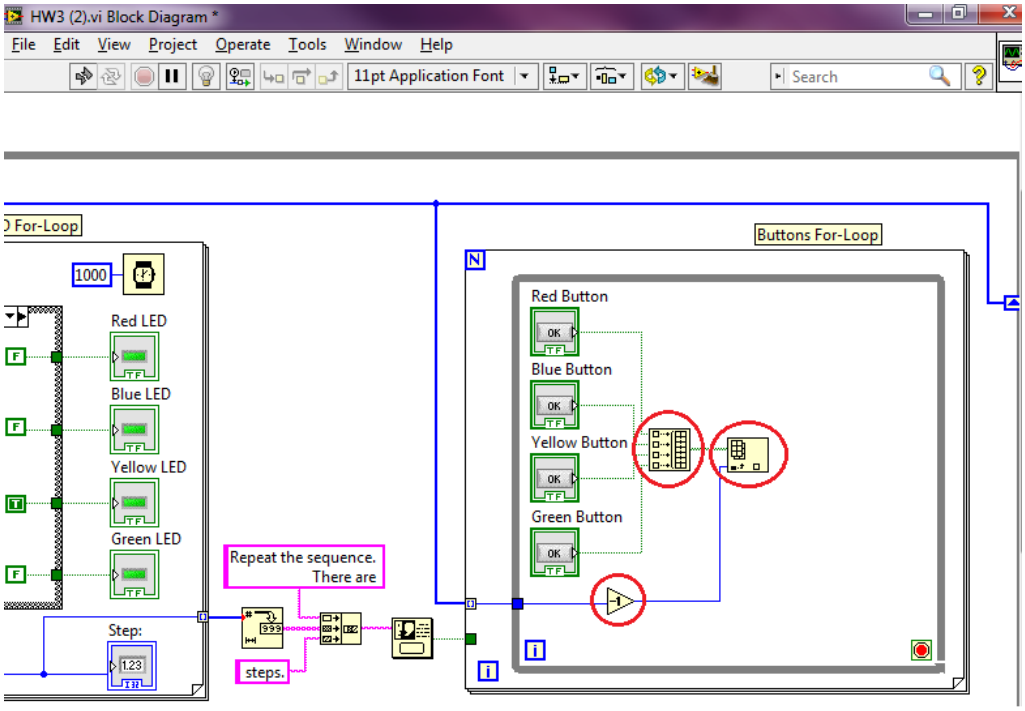

LED For-Loop
1000
3
Red LED
Blue LED
Yellow LED
Green LED
Step:
1.23
+1
Repeat the sequence.
There are
steps.
Buttons For-Loop
Red Button
Blue Button
Yellow Button
Green Button
OK
TF

Buttons For-Loop (Receiving user's input of what they think the LED sequence was)

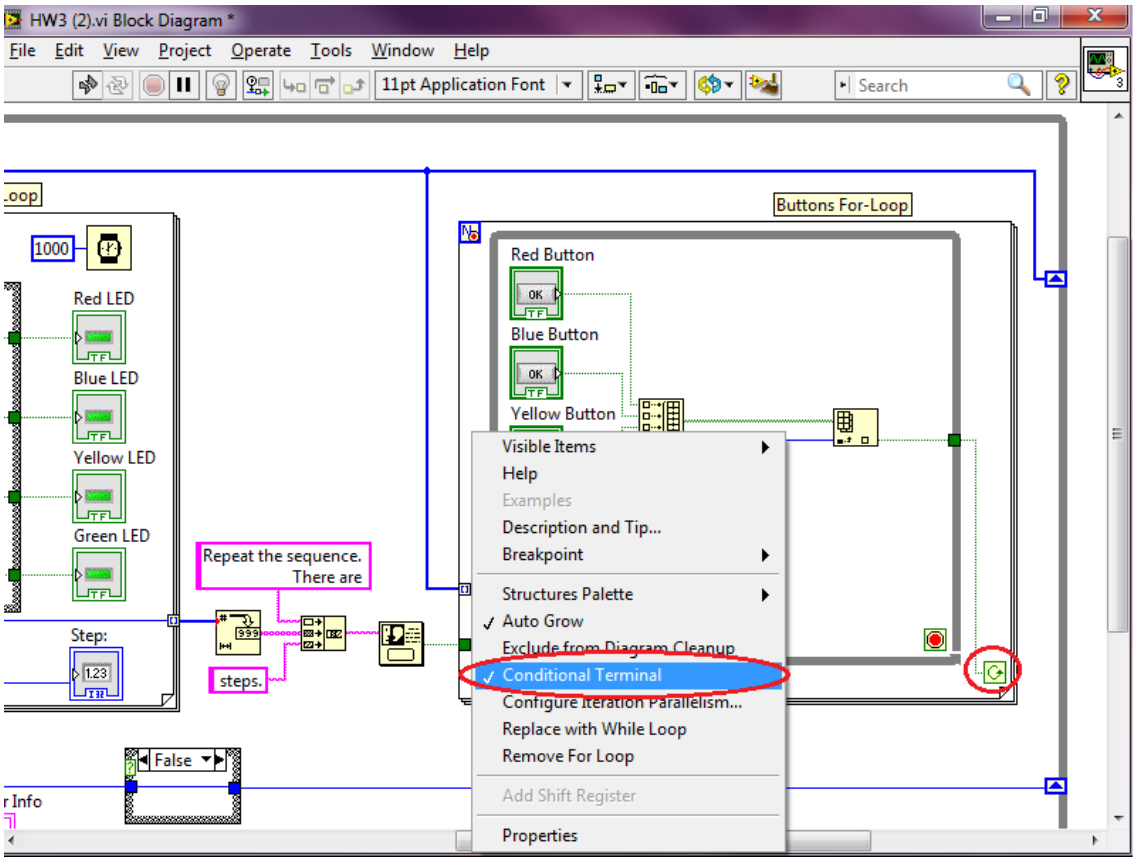
14. To ensure that the **Buttons for-loop** executes only after the user has pressed "OK", connect the output of the one-button dialog box we created in the last step to the left of the **Buttons for-loop**. You may have already done this since it is pictured above.
15. To tell the **Buttons for-loop** how many times to iterate, we should pass it the integer sequence array, which is also an input for the **LED for-loop.** Connect the integer sequence array to the left edge of the **Buttons for-loop**. Remember to keep indexing enabled.



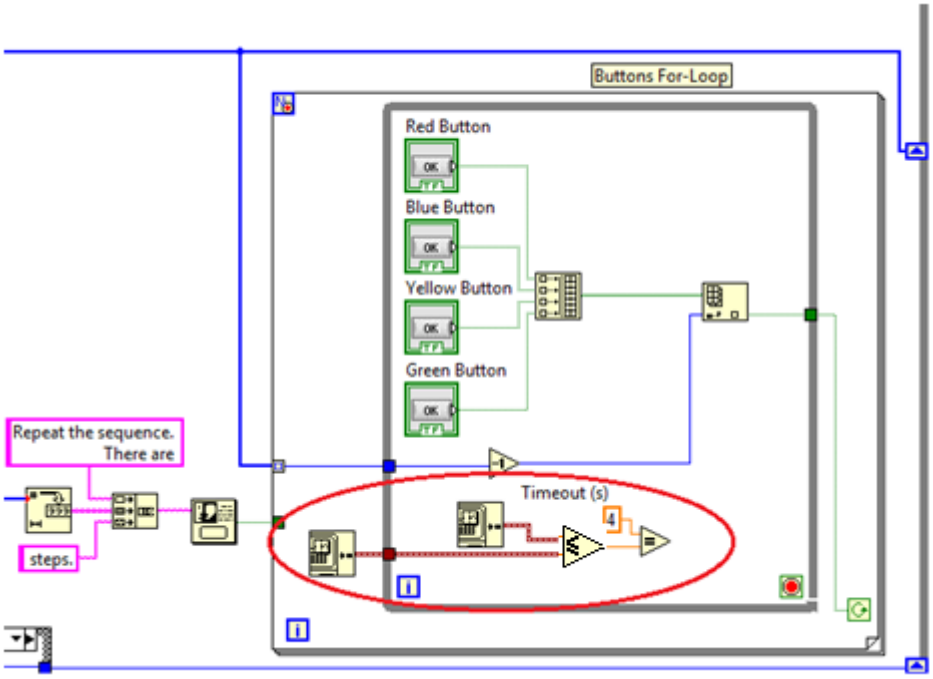
16. We need a way to figure out if the sequence is being repeated correctly by the player. One way is to build an array from the Buttons (*Right Click > Functions Palette > Programming > Array > Build Array)*, and then indexing (*Right Click > Functions Palette > Programming > Array > Index Array)* that array to get only the Button that should have been pressed (index comes from current iteration's sequence integer from integer sequence array that is passed in). However, since the range of that integer will be a number between 1-4, and we know that arrays start at index 0, we should decrement the sequence integer by 1 (*Right Click > Functions Palette > Programming > Numeric > Decrement)*.



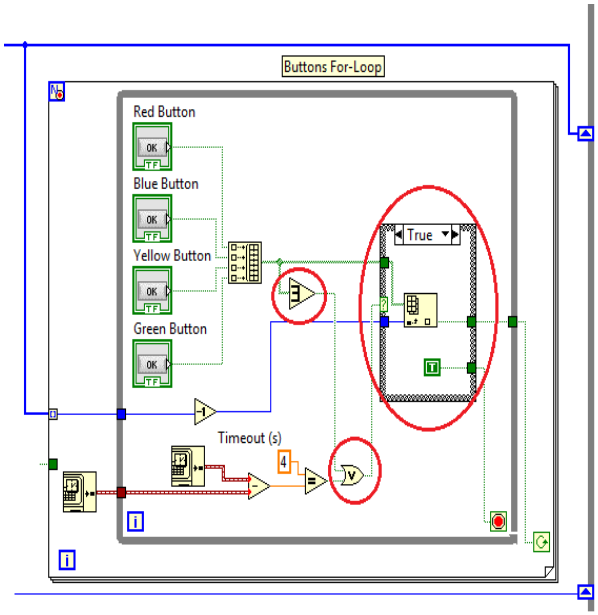
17. If the indexed Button Boolean value is not true, then we know that the user has not pressed the correct button, and we want to stop the game because they lost. To achieve this function, we can enable the conditional terminal of the **Buttons for-loop** by right clicking the outer edge of the for-loop and selecting Conditional Terminal. The change its loop condition to “Continue if true” by simply clicking on the stop sign. Then connect the indexed Button Boolean to it. The **Buttons for-loop** should look like this so far:

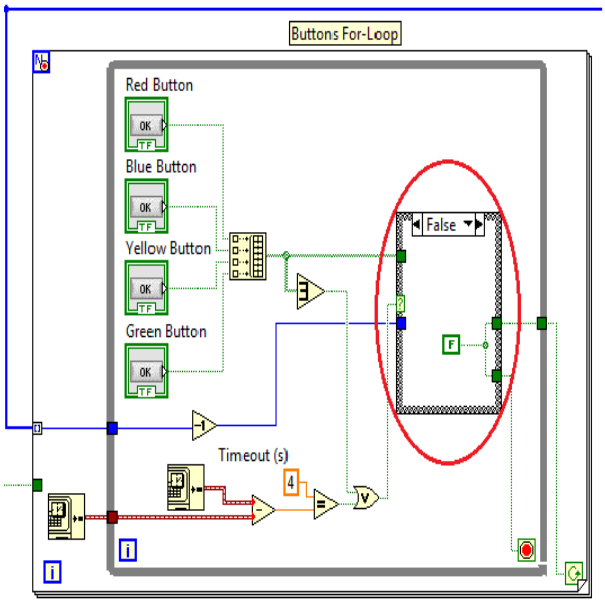


18. What is missing from this logic? How will the inner while-loop know when to stop? When do we want it to stop? We want the inner while loop to stop when a button has been pressed *or* when a timeout occurs (we'll say the timeout is 4 seconds for each sequence step). To achieve this, we can use the **Get Date/Time in Seconds** function (*Right Click > Functions Palette > Programming > Timing > Get Date/Time in Seconds)*. Place one to the left (outside) of the inner while-loop. This will give us the time at the start of the iteration of the **Buttons for-loop**. Place another one inside of the inner while-loop. This will get the time at every iteration of the inner while-loop. To see how much time has elapsed since the start of the iteration of the **Buttons for-loop**, subtract (*Right Click > Functions Palette > Programming > Numeric > Subtract)* the **Buttons for-loop** time from the inner while-loop time. If the resulting number equals (*Right Click > Functions Palette > Programming > Comparison > Less or Equal?)* "4", then it is safe to say that a timeout occurred.

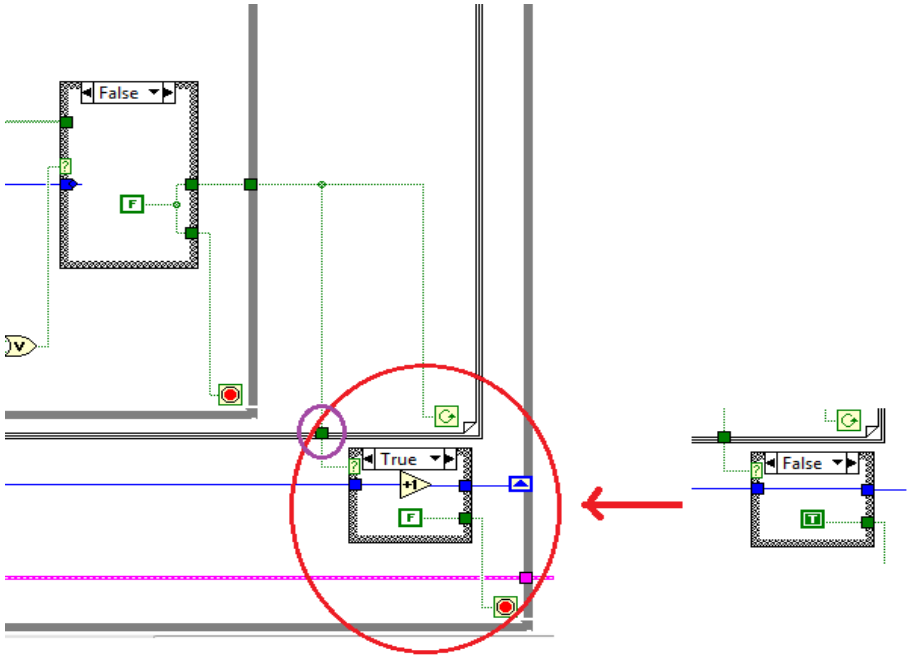


19. Furthermore, we only want to check what Button was pressed if there was actually a button pressed. One way to do that is to take the Buttons Boolean array and run it through an **Or Array Elements** function (*Right Click > Functions Palette > Programming > Boolean > Or Array Elements)* which will return true if any of the array elements are true. We can then OR (*Right Click > Functions Palette > Programming > Boolean > Or)* that result with the timeout Boolean in order to decide whether or not to check what Button was pressed. If the Button is checked, then the inner while loop should be stopped, otherwise false should be connected to the inner while-loop stop condition terminal. (Use a case Structure). Your final **Buttons for-loop** should look like this:

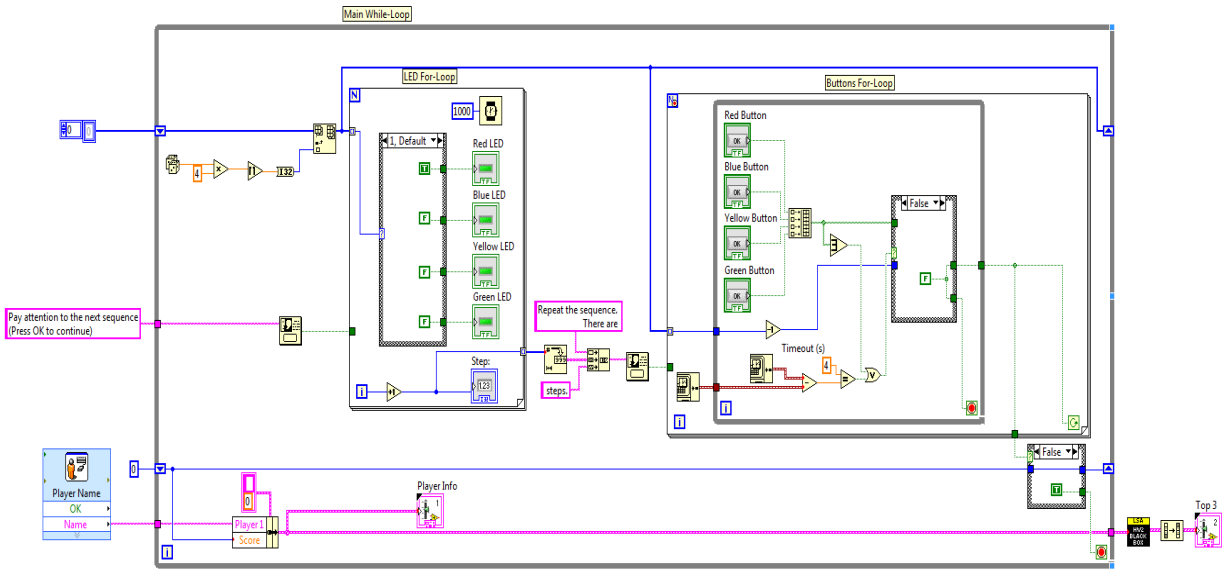




20. Finally, in order to decide whether to increment the score or to stop the game, the wire going into the “continue if true” of the **Buttons for-loop** should also be connected into the selector terminal of the score-counting case structure in the **Main while-loop**. Remember to disable indexing when that wire is coming out of the **Buttons for-loop**. To do this right click on the Boolean terminal coming out of the for-loop and select disable indexing. This terminal is circled in purple below. (If that Boolean is true in the last iteration of the **Buttons for-loop**, then it means that the player got all of the steps correct.) Also make sure you add Booleans to the case structures and wire up the Main while-loop’s condition terminal as seen below to stop the game when the user answered wrong.



21. Your block diagram should look similar to the one pictured below:



22. As a finishing touch, place Section3.vi in the Main VI virtual folder of the LabVIEW project you have been using for the Nigel Says project.