LABVIEW SLOT MACHINE PROJECT

Alexander Valentine

INDEX

INTRODUCTION

For this project, we were allowed complete freedom to create anything we please. For this reason, I decided to create a Slot Machine.

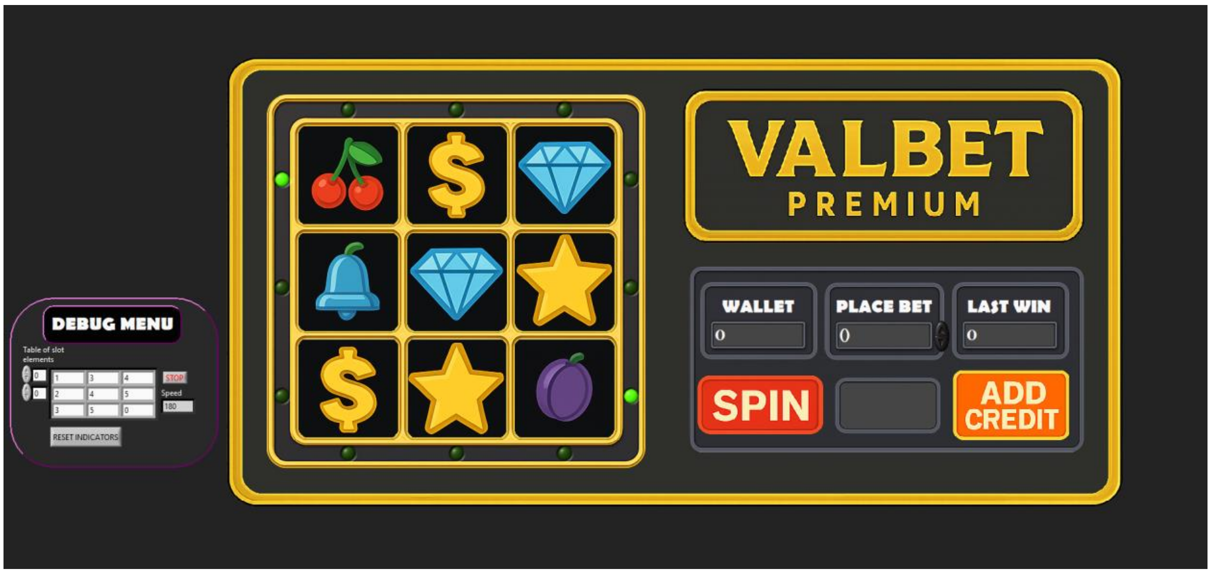


Table of slot elements

0
0

1	3	4
2	4	5
3	5	0

Speed 180

WALLET 0

PLACE BET 0

LAST WIN 0

(Fig. 1)

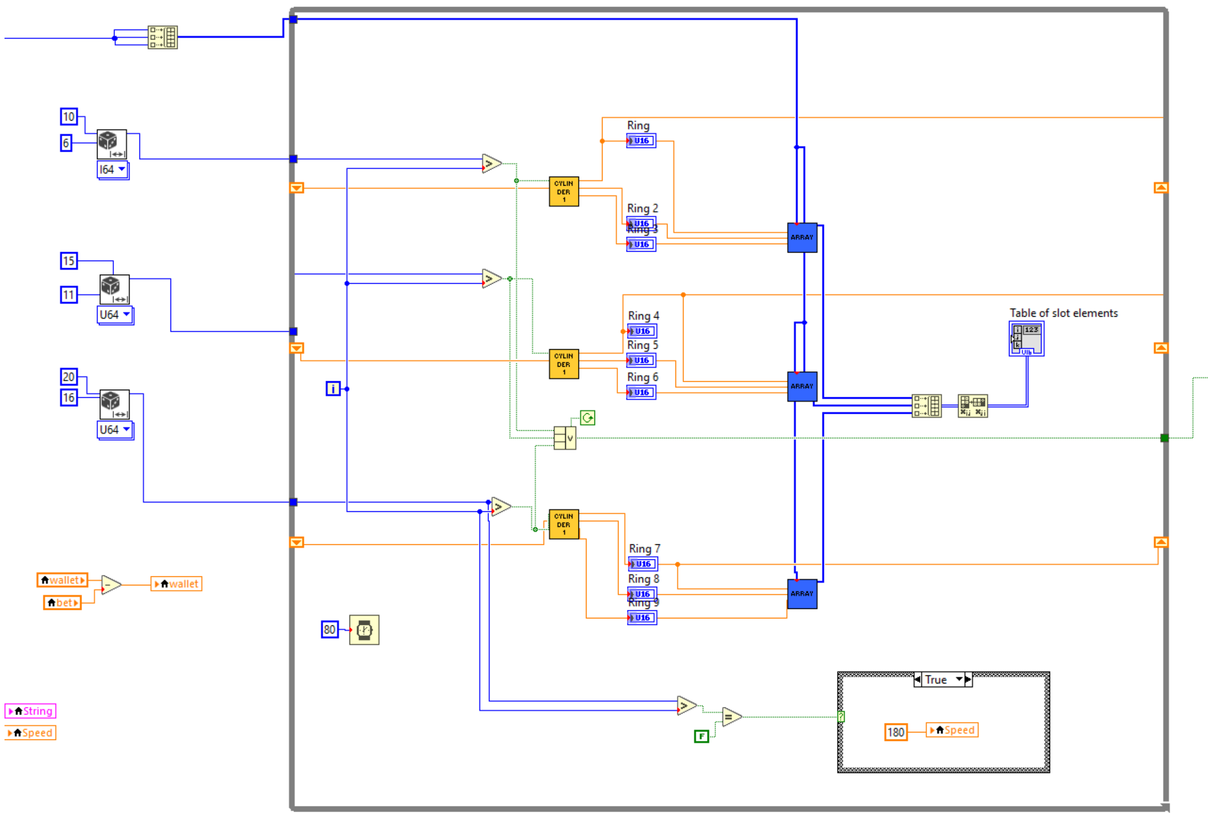
- The machine includes a 3x3 slot interface, with buttons to ‘spin’ it, and to add credit to the wallet in increments of 10.
- The “PLACE BET” control also increases in increments of 10 at the push of its buttons.
- The “LAST WIN” indicator shows the exact amount that was last added to the wallet after a win.
- Initially, lights flash in an anticlockwise pattern around the slot interface, and change later. They are explained further down in the report.
- The blank box between the two-coloured buttons displays text depending on certain cases.
- A “Debug Menu” is also included, in which there is a 2D array of the current elements that appear in the slot interface. In total there are 6 possible elements (0-5). It also includes a speed indicator which shows how much delay there is between the switching of the flashing lights. Finally, the “RESET INDICATORS” button, when pushed, resets the “WALLET”, “BET” and “LAST WIN” indicators.

EXPLANATION

SLOT GRID

The slot grid consists of 9 “Pict Ring” elements placed in a 3x3 format, decorated with custom made images. Each element is filled with the same 6 images: grape, cherry, bell, dollar, diamond and star (from 0-5 respectively).

When the “SPIN” button is pressed, each column of the grid is rotated separately as if it were a cylinder. This means that the images generated are not random, only the number of rotations is, as seen below:



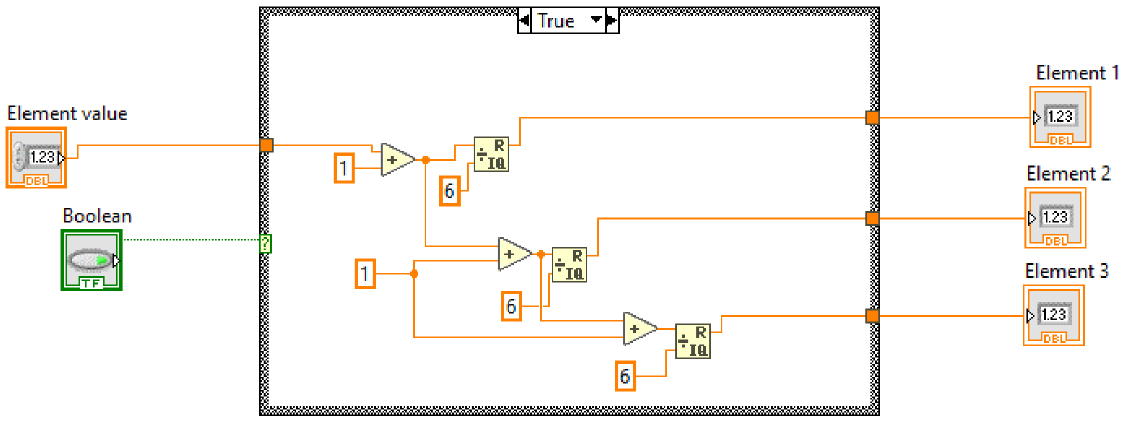
(Fig. 2)

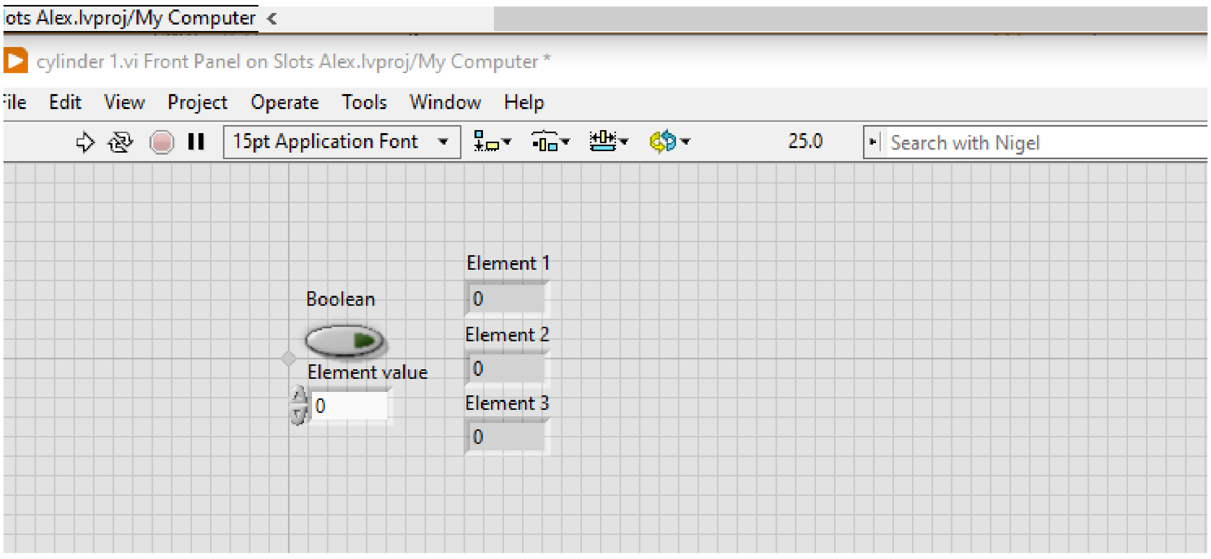
As you can see, there are 3 random number generators:

1. From 6-10, for the first column (left to right).
2. From 11-15, for the second column.
3. From 16-20, for the last column

By giving them gradually increasing ranges, we guarantee that the first column will have the shortest spin time, and the last column will always finish last.

Now, the way we keep them spinning is with a SubVI named Cylinder 1, which contains the following items:




ots Alex.lvproj/My Computer
cylinder 1.vi Front Panel on Slots Alex.lvproj/My Computer *
File Edit View Project Operate Tools Window Help
15pt Application Font
25.0
Search with Nigel
Boolean
Element value
0
Element 1
0
Element 2
0
Element 3
0

(Fig. 3)

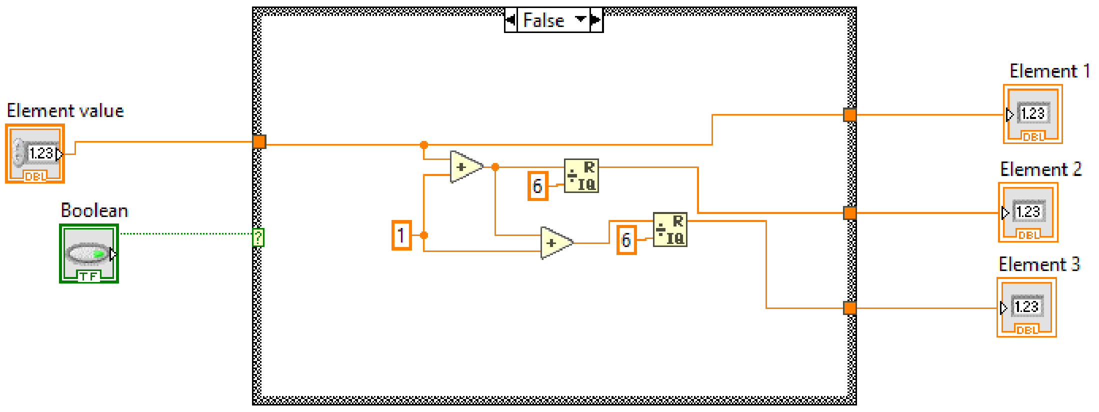
When true, using a “Remainder and Quotient” block, we add 1 to each element. Since we start from 0, after 1 loop we would have the following:

Element 1 = 1; Element 2 = 2; Element 3 = 3

And these elements would never be able to exceed 5, since at 6 the remainder is once again 0 and everything resets.

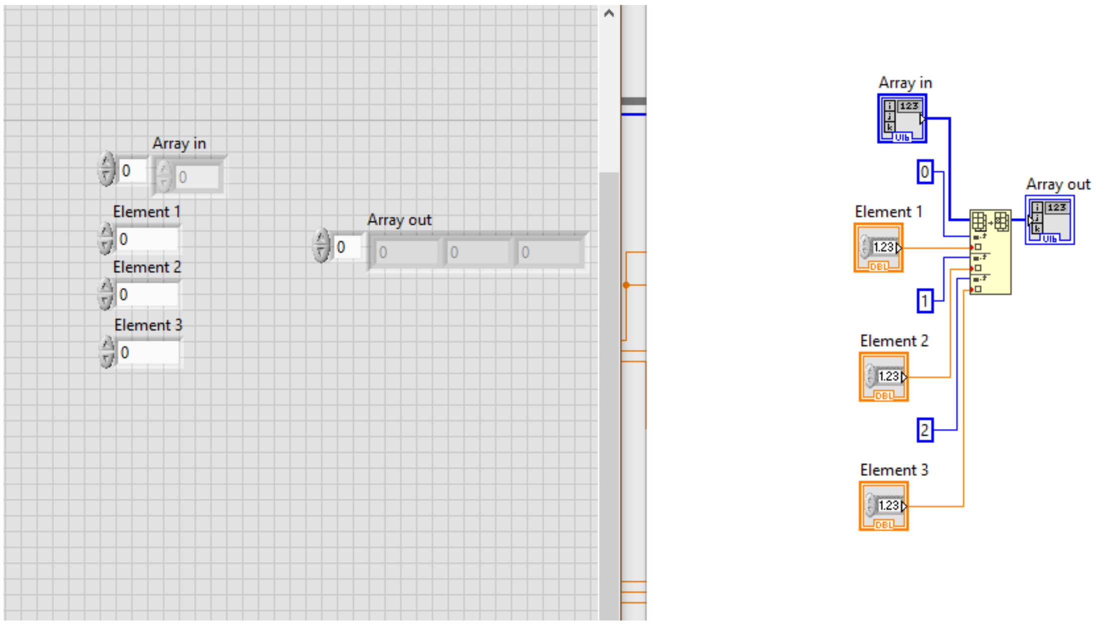
In this SubVI, the 3 elements that you see are output terminals which go to, for the first column, Ring 1; 2; 3. The Boolean and Element Value are input terminals, which are connected to the shift register of figure 2 and the "Greater?" comparison of the corresponding column.

The "Greater?" comparison compares if the number of iterations of the while loop has reached the randomly generated number, in which case if it does, then "Cylinder 1" switches to the following false case, in which the elements no-longer increase by 1, but stay the same, indicating the stoppage of the 'spinning':



(Fig. 4)

Now, after the rolling, each "Ring Pict" value gets sent to another SubVI called "Array". This SubVI takes each input and creates a 1D array with them:



(Fig. 5)

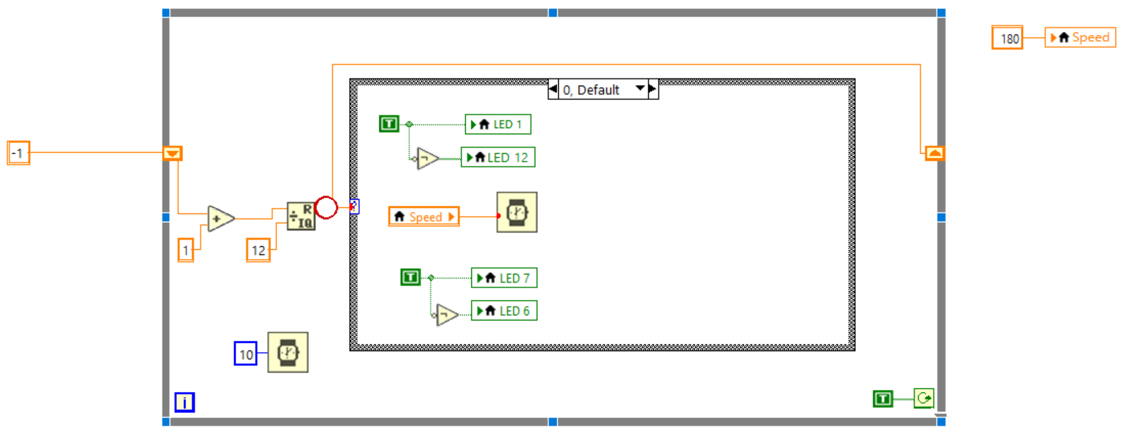
As you can see, we need a previously generated input array because it doesn't just create a new one each time. This is why we have a "Build Array" block on the top left of figure 2 which is connected to a constant 0 three times.

Once we create the three arrays for our 3 columns, we use another "Build Array" to bind all three and form our "Table of Slot Elements". (We need to transpose it because the SubVI creates the 1D arrays horizontally).

Finally, this all resides in a "While Loop" which continues running until the Compound Arithmetic "OR" block sends a FALSE signal and stops the loop. This happens when all three columns finish spinning, in other words, the number of iterations matches all of their randomly generated numbers.

LIGHTS

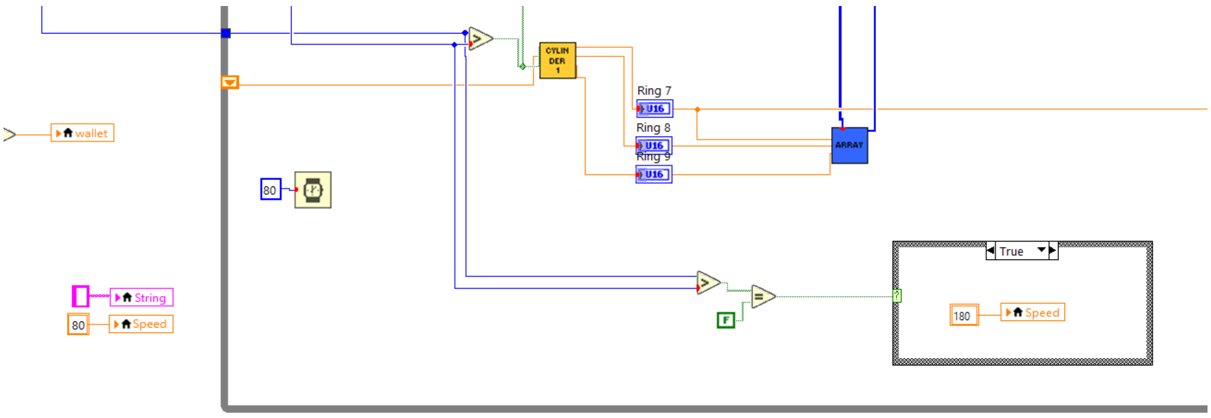
When the program starts, a variable called "SPEED" is set to 180. This variable controls the delay of the following while loop:



(Fig. 6)

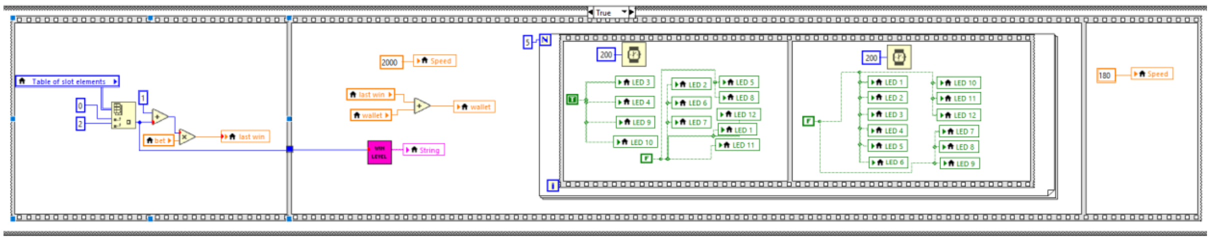
This while loop is outside of everything else, therefore it runs parallel to the actual slot machine. This is what is known as a "State Machine". It runs through 12 continuous cases, which are the ones to control the lights around the grid.

Initially, two lights chase each other in an anticlockwise fashion. When "SPIN" is pressed, they speed up until the last column stops spinning: (Fig. 7)



As you can tell, “SPEED” is set to 80 (decreased from 180) to match the spin speed of the columns. Once the iterations meet the random number, it is set back to its original value of 180.

Now, if there is a winning line (explained further below), with the following structure we have made them flash 5 times in accordance to the line:

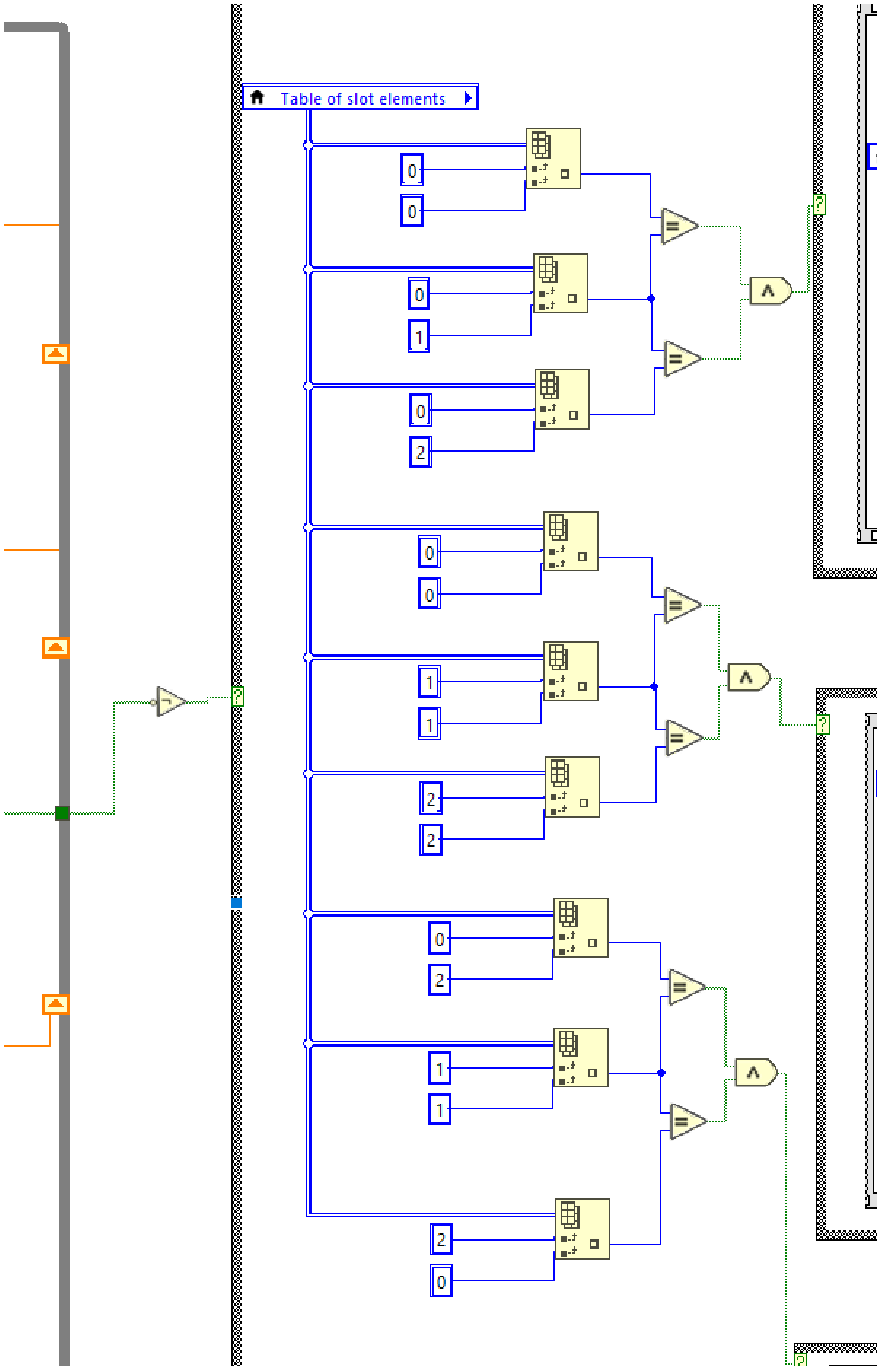


(Fig. 8)

In the second frame of the TRUE (winning) case, “SPEED” is set to 2000, to allow time for the “For Loop” to finish 5 iterations of flashing. After it finishes, the next frame sets the speed back to 180 and the light loop returns to normal.

WINNING LINES

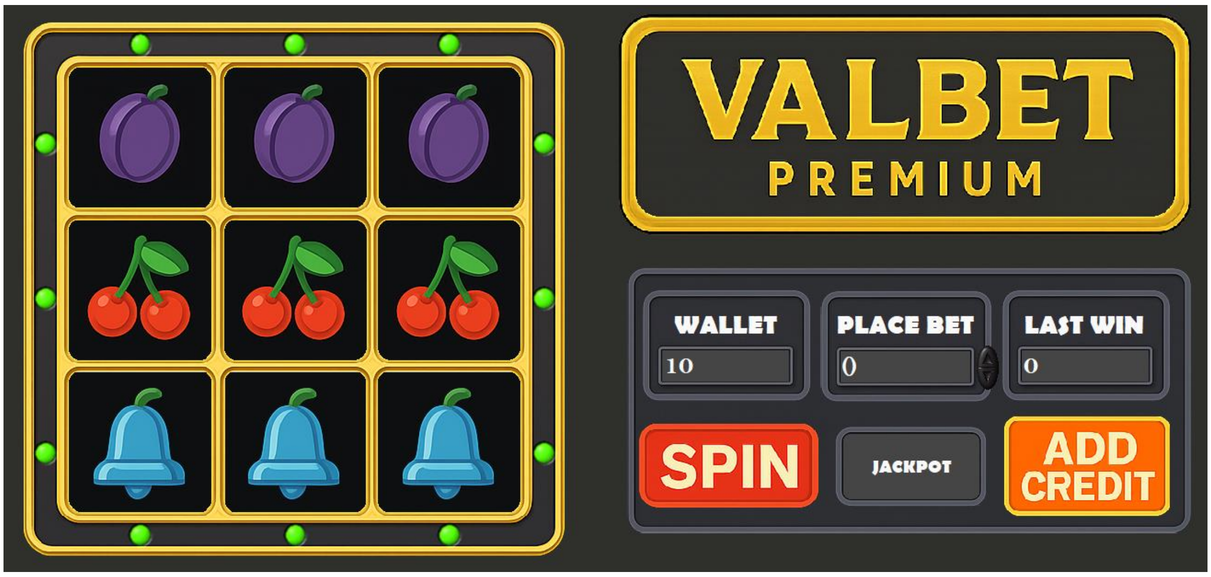
After the columns stop spinning, their false output is negated through a NOT gate, which activates the following TRUE case structure:



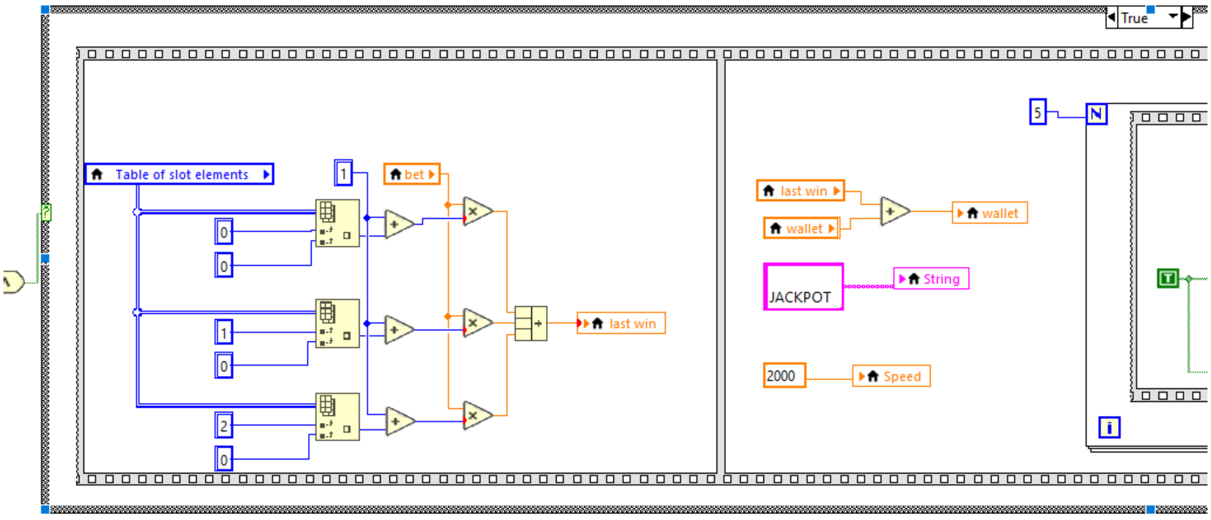
(Fig. 9)

This is why we needed the 2D slot element array. Here, with an "Index Array" block, we check for the following 3 possible winning lines:

1. JACKPOT: In this case, there is a straight horizontal line of the same image, which, since we are simulating rotating cylinders, automatically means that there will be 3 such lines at a time:



(Fig. 10)



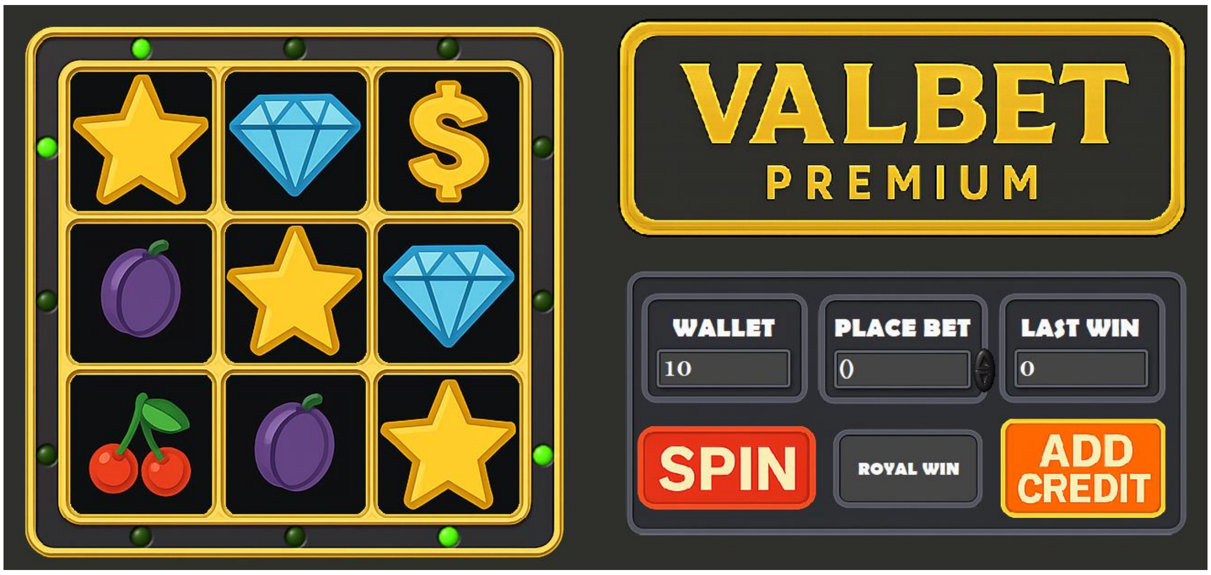
(Fig. 11)

For this case, the value of one picture of each row is taken, summed to 1, multiplied by the original bet and added to the wallet. For example, if our bet here was 10, we would have the following sum:

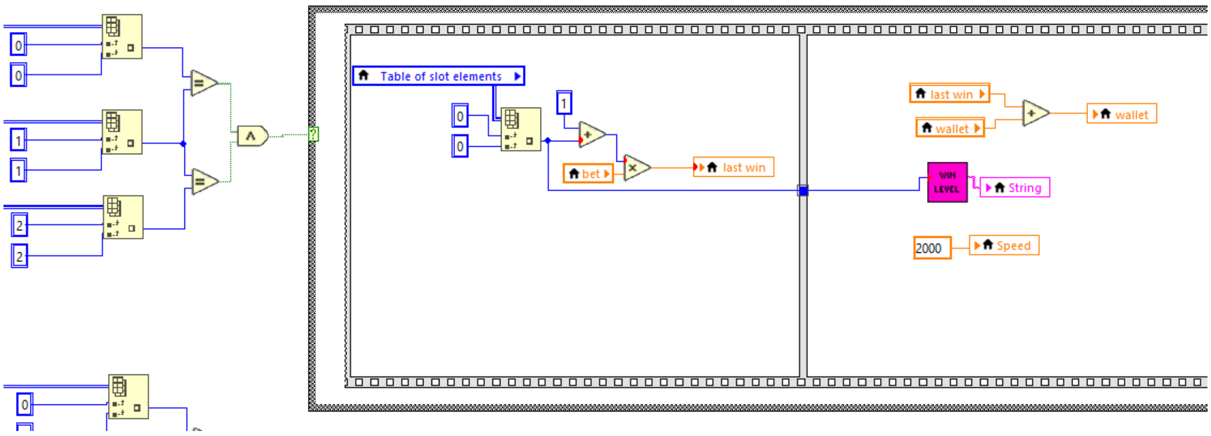
- Grape = 0; Cherrie = 1; Bell = 2; Bet=10
- (0+1) x 10 + (1+1) x 10 + (2+1) x 10 = 10 + 20 + 30 = 60

Therefore, we would have 60 in the "LAST WIN" indicator, and the text says "JACKPOT".

2. Diagonal 1: In this case, three of the same icons are in a diagonal line from top to bottom:



(Fig. 12)



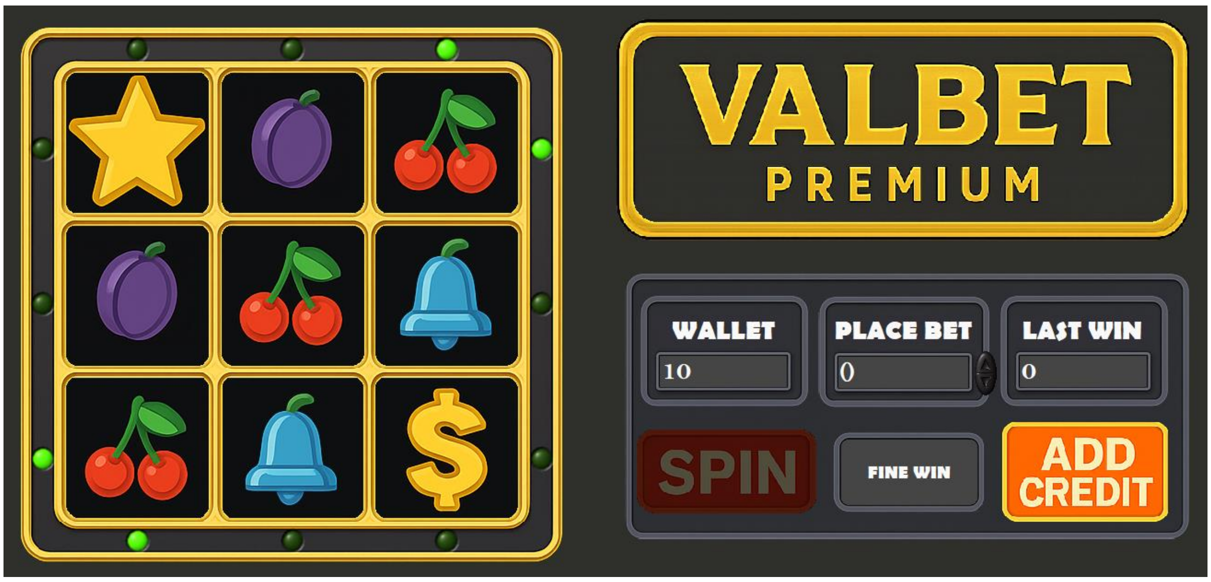
(Fig. 13)

In this case, we only check the value of the top right corner, since in reality we only have one winning line here, not three like before. For this reason, once again 1 is summed to the read value, and then it is all multiplied by the bet amount, for example:

- Star = 5; Bet = 10
- 10 x (5+1) = 60

Therefore, we would have 60 in the “LAST WIN” indicator. It is worth mentioning that the text displayed is different, and it is connected to yet another SubVI called “WIN LEVEL”. This will be explained after the last possible winning line.

3. Diagonal 2: In this case, the diagonal is just reversed, ergo from bottom to top:

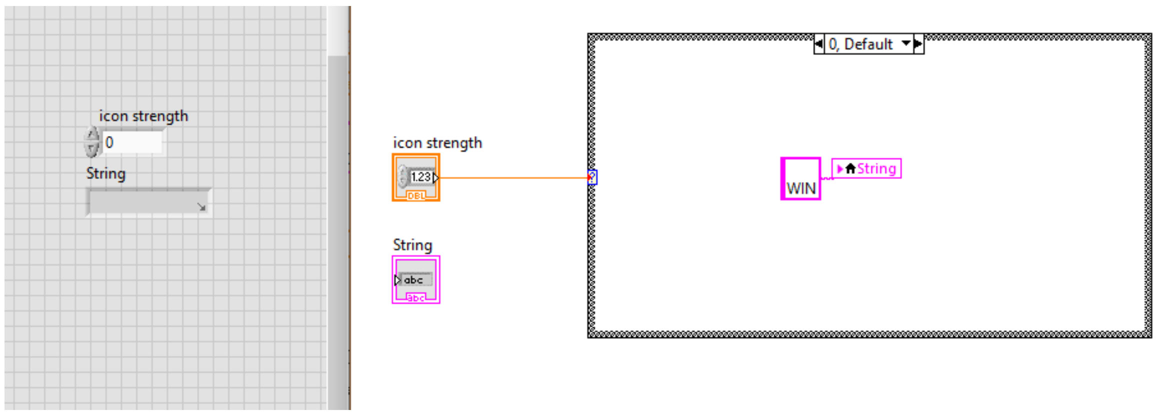


(Fig. 14)

This case is absolutely the same as the previous one, the only difference being that the diagonal is reversed, so there is no need for further explanation.

In summary, each case checks if there is a line, or 3 depending on the orientation. However, there are different 'types' of wins, characterised by the value of the icon (0 to 5), since this value plus 1 becomes the multiplier for the bet made. These values were mentioned in the introduction. Additionally, the light configurations, which you see in figures 10, 12 and 14, are the same ones which will flash 5 times if there is such a win, otherwise nothing will happen and the lights will continue their default pattern.

With the SubVI "WIN LEVEL", we can determine which text message will be displayed:


icon strength
0
String
icon strength
1.23
DBL
String
abc
0, Default
WIN
String

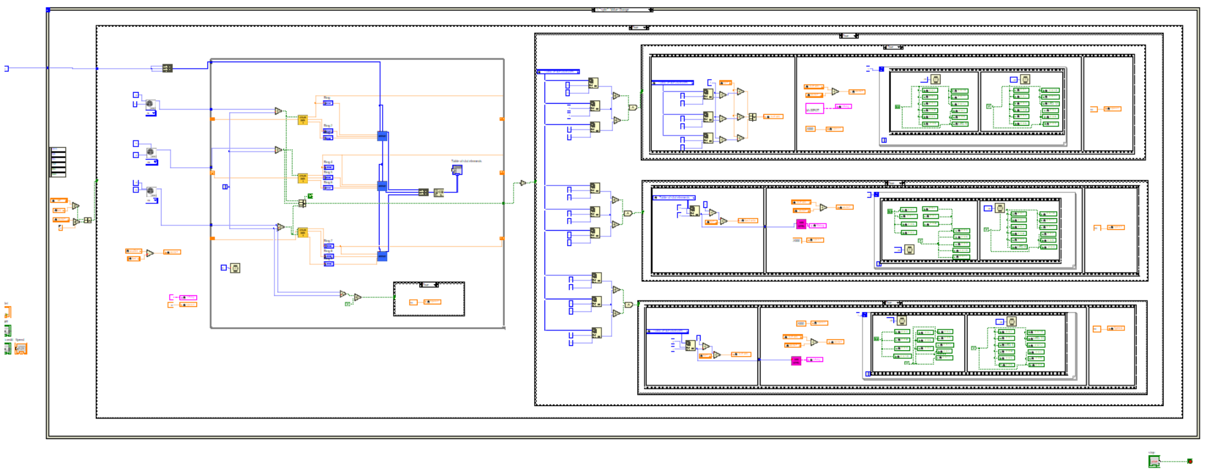
(Fig. 15)

All it is, is a “Case Structure” linked to an “Icon Strength” input terminal. In the structure, we have a total of 6 cases:

1. Grapes: (Value = 0) text displays “WIN”; bet multiplier is 1
2. Cherries: (Value = 1) text displays “FINE WIN”; bet multiplier is 2
3. Bell: (Value = 2) text displays “MAJESTIC WIN”; bet multiplier is 3
4. Dollar: (Value = 3) text displays “GRAND WIN”; bet multiplier is 4
5. Diamond: (Value = 4) text displays “SUPREME WIN”; bet multiplier is 5
6. Star: (Value = 5) text displays “ROYAL WIN”; bet multiplier is 6

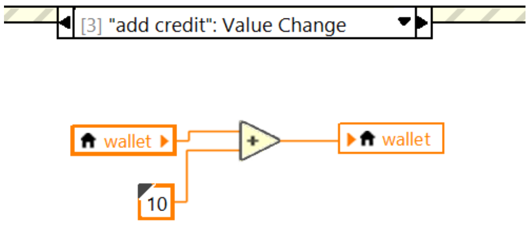
BUTTONS

We have a total of three important buttons: “SPIN”, “ADD CREDIT” and “RESET INDICATORS” (debug menu). These buttons each complete their actions with the help of an “Event Structure”, which contains everything except the parallel running lights, which sit in a “While Loop” outside of the big loop that contains the event structure (the outmost furthest box):


(Fig. 16)

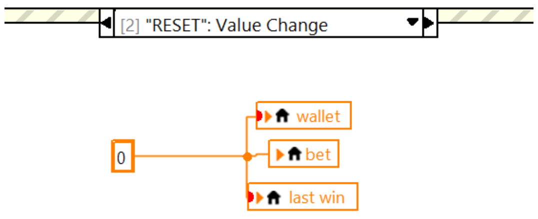
Each button has its own “Value Change” event:

- “SPIN”: This button activates the case in figure 16, which is explained above under “Slot Grid”.
- “ADD CREDIT”: This button adds 10 credits to your wallet each time it gets pressed:



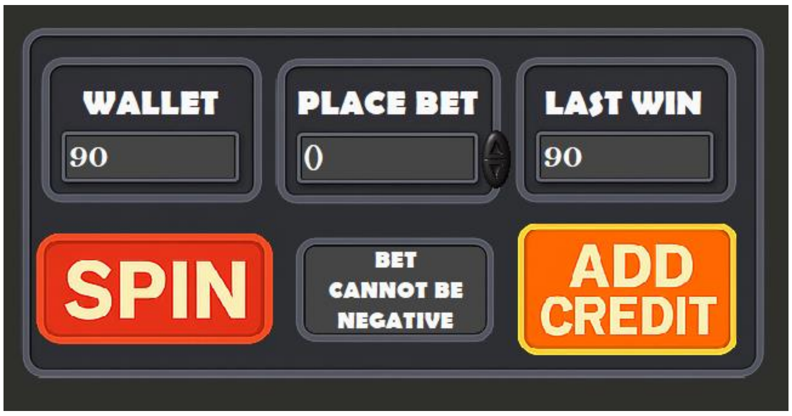
(Fig. 17)

- “RESET INDICATORS”: This button resets each indicator on the machine interface when pushed:

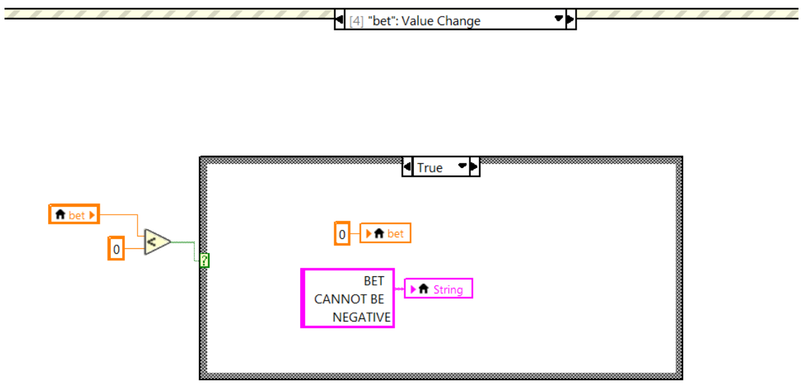


(Fig. 18)

- Bet Control: If someone was to try to make the bet negative, the message box will display “Bet cannot be negative”, and the bet would be reset to 0:



(Fig. 19)



(Fig. 20)

DECORATION

To decorate the front panel, I relied on ChatGPT and an app called Paint.Net. I used ChatGPT to generate various different images which I then adjusted with Paint.Net. If I could, I would've generated everything to save myself the effort, but ChatGPT often makes mistakes and doesn't seem to like to generate the same theme every time, often making me ask for a big image containing almost every element in it. Nevertheless, Paint.Net proved incredibly useful for editing and making adjustments to the images.

Furthermore, I created an abundance of custom "Type Definitions" (controls), which can be found under "Custom Controls" in the project viewer. These were quite tough to get right at first, because many borders and small details are not able to be deleted for some reason in LabView.

CONCLUSION

In conclusion, this project was quite fun to create. I was shocked at how much simpler it was than what I had imagined, which is why I started adding various different details such as the lights, which turned out to be the hardest part since there isn't an efficient way to turn a while loop on and off. However, in spite of the lights being such a challenge, the emotions I felt when I completed the whole project were on another level.