

Subscribe to DeepL Pro to edit this document.  
Visit www.DeepL.com/Pro for more information.

Project Report:

Minesweeper

1. **Project Description.**
   1. Project Objective:

Making a functional version of the game Minesweeper in the LabView environment

* 1. Project details:

1. Before starting the game in LabView, the user sets the board size (width and height of the board) and the number of mines. Based on the above data, the game board is created.
2. If the exposed field has a mine it is game over. In this case a failure message should be displayed,
3. If there is at least 1 mine around the exposed field, the number of adjacent mines is displayed at the exposed field location,
4. If there are no mines around the uncovered field (i.e., adjacent fields have no mines) we uncover the corresponding fields (i.e., fields that also have no mines around them),
5. If all the squares without mines are revealed or if all the mines are correctly pointed out, the game ends with a success. In this situation a success message should be displayed.

**2 Divide the project into tasks:**

1. Creating a board:
   1. Ability to set board size and handle errors associated with it.
   2. Adding mines and scattering them on the board.
   3. Calculate the number of mines around a given field for each field.
   4. Graphical representation of each possible field.
2. Gameplay:
   1. Recognition of left and right mouse button clicks on the game face.
   2. Execution by the program of the player's actions and display of the relevant data (information about the clicked field).
   3. Revealing the corresponding fields around the clicked field.
   4. Display information about success, loss. (In case of loss, displays the positions of all mines)

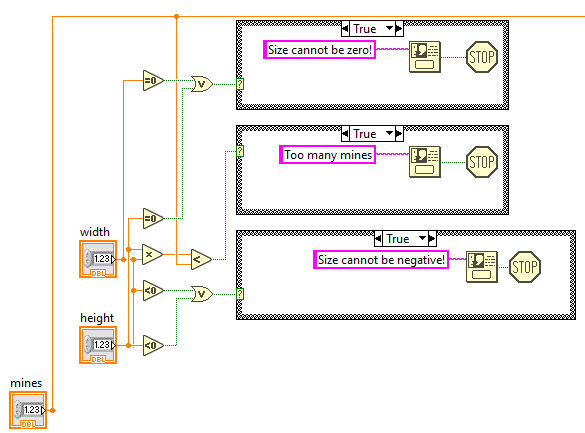
**3 Implementation:**

1. Creating a board:

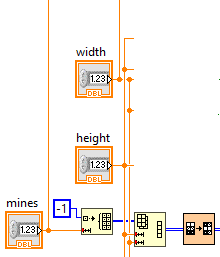
1.1 Ability to set the board size and handle related errors

This task was accomplished with "Numeric control" buttons allowing the user to select the corresponding board size.

The supported errors are:

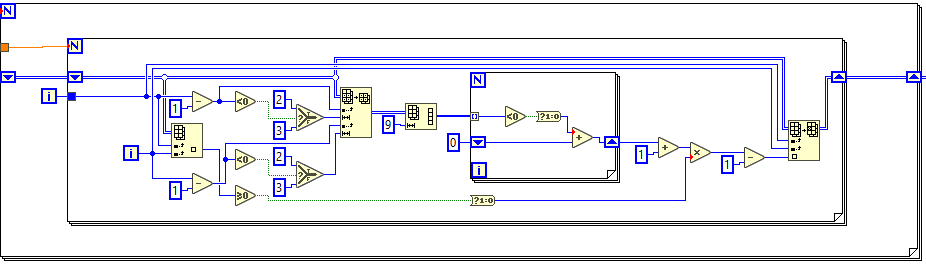
1. Negative board dimensions
2. No board dimensions
3. Number of mines greater than board size

1.2 Add mines and scatter them on the board.

A one-dimensional "array" is created that contains a user-specified number of mines (denoted by the value "-1"), the dimensions are then changed to those given by "Numeric controls", the values of the resulting "array" are scattered using the "Shuffle Array" function

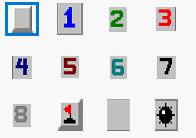
* 1. Calculate the number of mines around a given field for each field.

For each field on the board, its surroundings are checked (by default, the eight surrounding fields; for edge fields, only existing fields are considered). If none of the surrounding fields is a mine, the value of the field is 0 by default. If some of the fields are mines, 1 is added to the value of the field for each mine. When the checked field is a mine its value does not change



1.4 Graphical representation of each possible field.

For graphical representation, "picture rings" are used

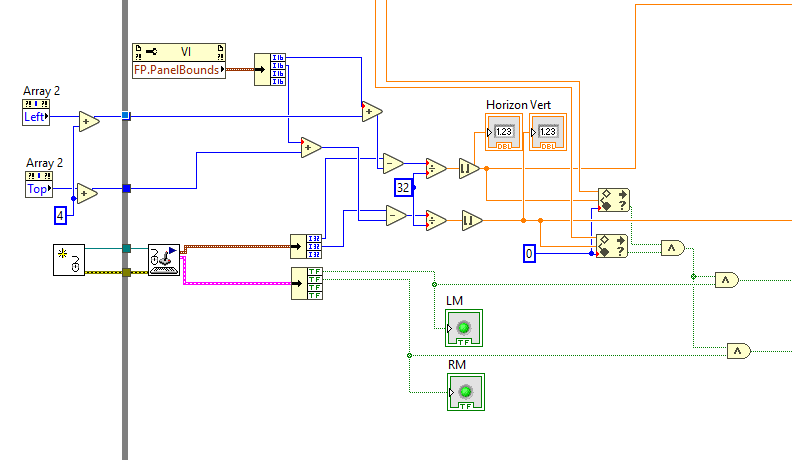


Each box takes on values from zero to eleven, with zero being the box marked, one being the box with the value one, and so on, a flag being marked with the value nine, a box containing zero min around it with the value ten, min with the value 11.

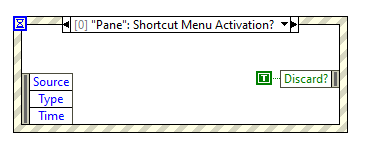
2 Gameplay:

* 1. Recognition of left and right mouse button clicks on the game face.

The mouse is initialized and the clicked field is recognized taking into account the position of the LabView window on the monitor and the game shield in the Labview window (obtained using "Property Node") and assuming a fixed size of the shield field. The exact coordinates of the clicked field in the 2d array are obtained, taking into account the right and left mouse buttons.



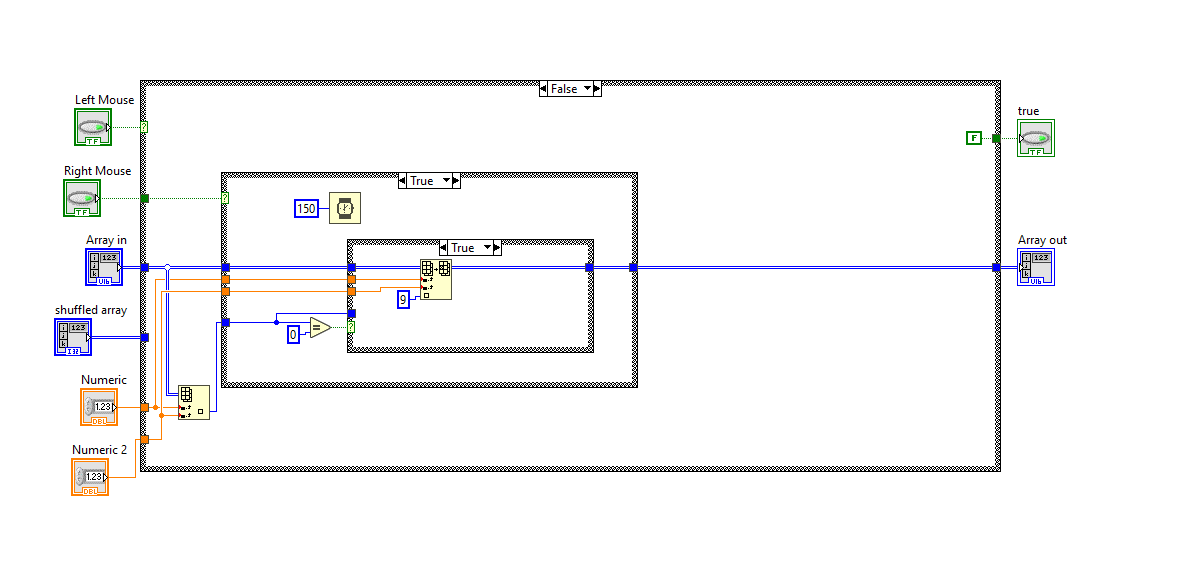
Getting rid of the menu when right-clicking while the program is running was accomplished with an "Event Structure" that captures all menu activations and removes them.



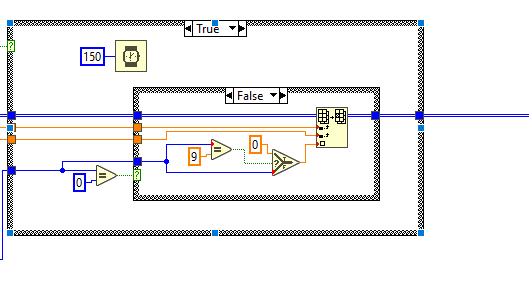
2.2 The program executes the player's actions and displays the relevant data (information about the clicked field).

This was accomplished by using user click data, for an "array" of images assigns corresponding values from a numeric "array". The "Case Structure" is used for this, determining the type of click. Implemented using a separate nested VI (Untitled-1SubVI.vi)

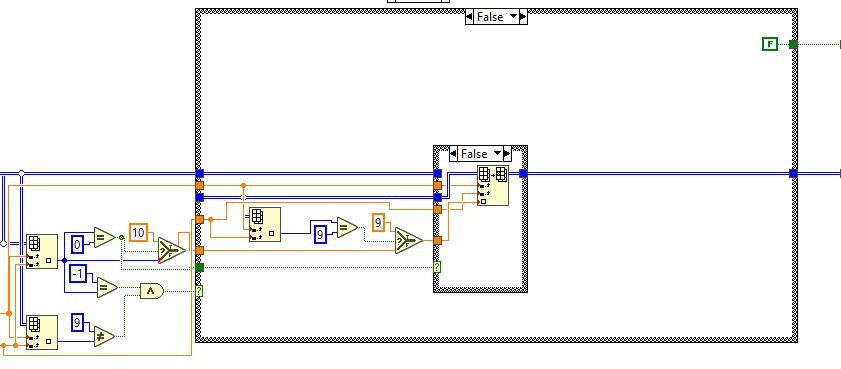
If you click the right mouse button, a flag appears on the inactive fields to prevent accidental left-clicking.

****

Right-clicking the flag again changes the field to an inactive field



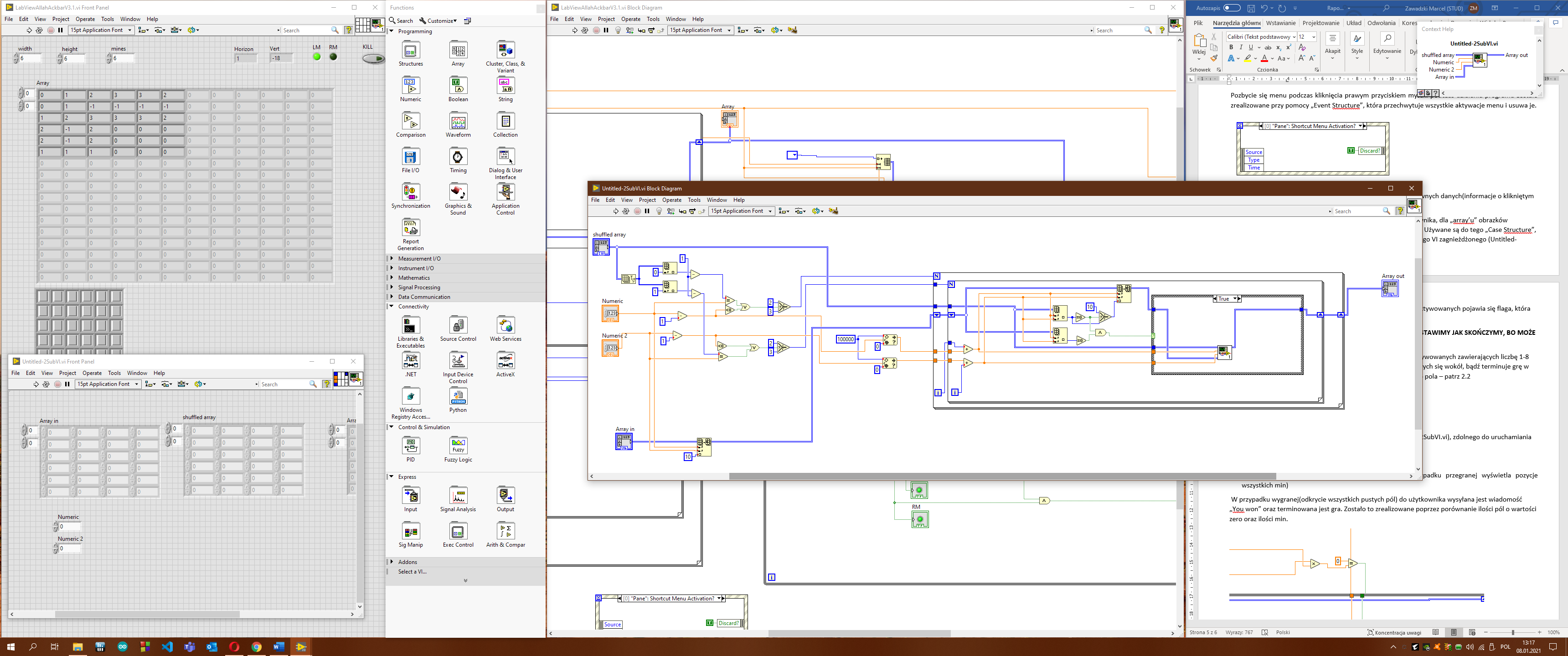
If the left mouse button is clicked on non-activated fields containing the number 1-8, they change their value to that corresponding to the number of mines around, assigned based on the 'array' from Section 1.3, except for zero, which takes the value 10 in the picture ring. If an empty field is pressed (value 0) - see 2.3

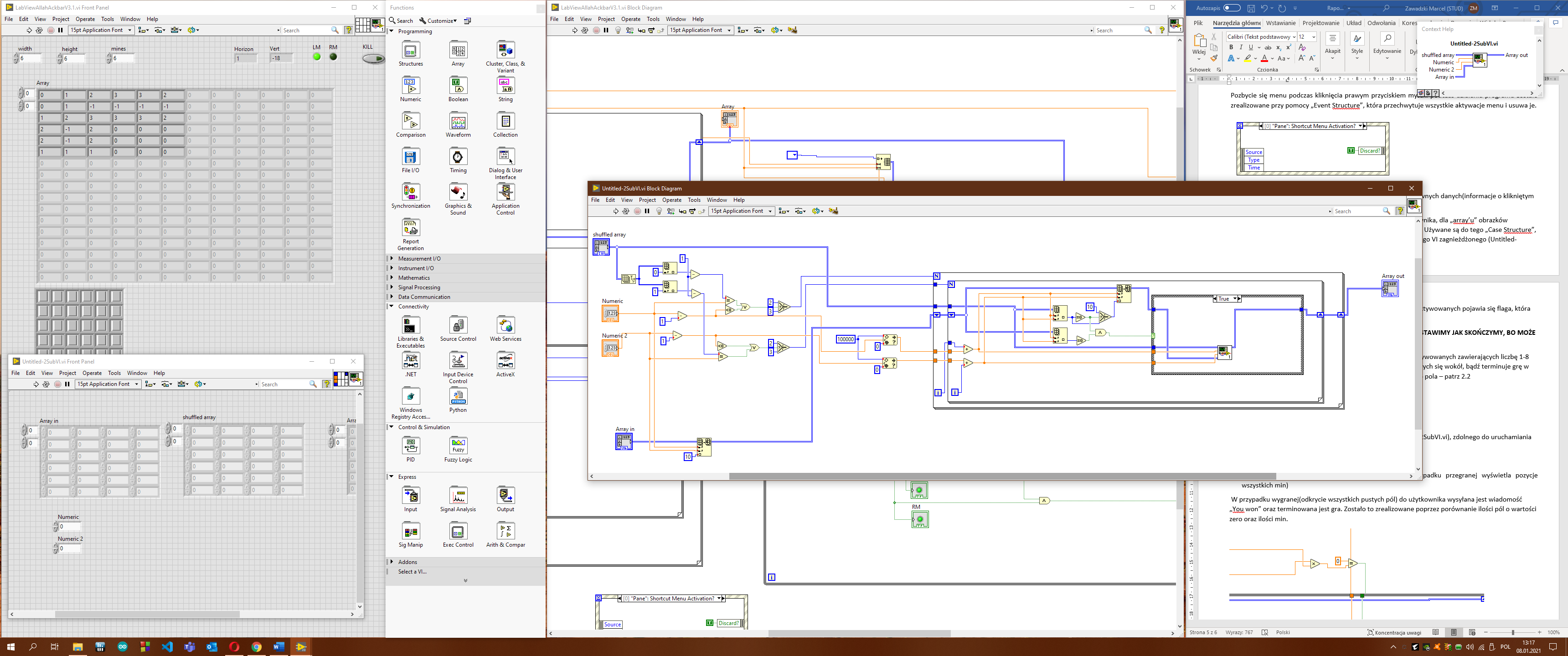


If you click on a mine, the game is terminated by checking whether the field you clicked on contains a mine - see 2.4

* 1. Revealing the corresponding fields around the clicked field.

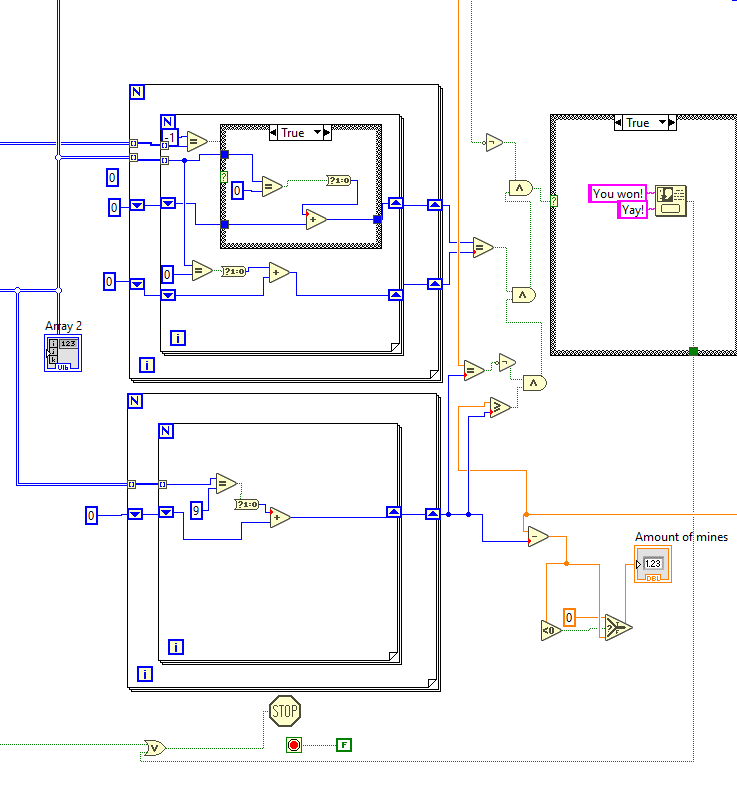
Implemented using a separate nested VI (Untitled-2SubVI.vi) capable of recursive triggering. The field for which the VI is run changes its value to an empty discovered field (10) and each adjacent field (following similar rules as in Section 1.3) is discovered if it is 1-8 or the same VI is called for it recursively if it is 0. As a result, a region consisting of empty discovered fields surrounded by an edge of discovered numbers 1-8 is discovered.





* 1. Display information about success, loss. (In case of loss, displays the positions of all mines)

The prerequisite for winning is to clear all fields without mines, and there may be flags on the mines, or they may not be clicked on at all. This is done by counting the number of empty fields and the number of empty fields on mines and comparing these values with each other. The number of flags on the field is also taken into account, so that random marking does not result in a win. We have also implemented the classic sapper's mine counter, which informs us how many mines are left to be marked with flags, each time a flag is placed the number of mines decreases, no matter how it was placed (classic sapper's rules, not to make the game easier)



Clicking on a mine (a field with a value of "-1" that is not marked by a flag) causes the "You clicked the mine" notification to appear, terminating the game and displaying all the mines that were on the board by iterating through all the fields and changing the value of the "ring array" to the value corresponding to the mine.

