Group Project Documentation

Technical documentation of the project

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology

{model document version: version 2/2023}

Project name and acronym: {name of the project, e.g.: Port security system against terrorist threats - SZP} **Redundant coding visualization app**	**Principal:** {customer name} **Bartosz Czaplewski, PhD**	
Order number: {number of the project team within the Group Project according to the SPG system, e.g. 13@KSSR'2022} **5@KSTI'2023/24**	**Project manager:** {project team leader} **Bartosz Kołakowski**	**Opiekun project:** {opiekun projektu} **Bartosz Czaplewski, PhD**

Document name/code: **Technical documentation of the product – DTP**	**Version No.:** {document version e.g. 1.00} **5.00**
Responsible for the document: {surname, first name} **Kołakowski Bartosz**	**Date of first drafting:** {date of first version of the document} **09.01.2024**
	Last updated date: {date of execution of the current version of the document} **08.06.2024**
	Semester of the Group Project: {enter 1 or 2} 2

Historia dokumentu

Version	Description of the modification	Chapter / page	Author of the modification	Date
1.00	{description, e.g. draft version} Preliminary	{e.g. whole} whole	{surname, first name} Kołakowski Bartosz	{date modified} 09.01.2024
2.00	{description e.g. correction in p.2.2} Changing Drawing 5 and Drawing 25	{e.g. point 2.2} 2.2.1, 2.2.5	{surname, first name} Kołakowski Bartosz	{date modified} 09.01.2024
3.00	Updated general description, added description of Reed-Solomon code	2.1, 2.3	Jastrzębski Paweł	22.05.2024
4.00	Description of exe generation	2.1	Jastrzębski Paweł	29.05.2024
5.00	Rewording of chapter 2.2. Added chapter 2.2.4 describing windows when choosing the Reed-Solomon algorithm. Added chapter 2.5 describing the translation of the application.	2.2 and its subsections, 2.5	Noga Piotr	08.06.2024

{NOTE: in the second semester, the documentation can be an extension of the documentation from semester I (new version of the document), it can also be a new file;

NOTE: If the documentation was created using another tool, the file with the documentation in this document should be indicated as an attachment to this document*; creating documentation in any other way does not exempt from the obligation to create this document, but instead of a documentary description, it is sufficient to indicate a document created in another way}*

Table of contents

1 Introduction - About the document

1.1 Full Document

{do not change}

The purpose of the document is to document information about the product, its functional features, technical parameters, flowcharts, software, results, product photos, measurements, tests and other elements required by the account manager and the customer.

1.2 Document scope

{specifying what is included in the scope of the document and what is not, or indicating related documents}
Documenting the application from the user's perspective, including documenting how Hamming encoding works.

1.3 Customer

{specification of the addressees of the document, it can be the type of recipient; here: the contractor (Department), members of the project team and named persons to whom the document is to reach}
Principal - Bartosz Czaplewski, Ph.D. (KSTI)
Team Members:
Bartosz Kołakowski
Michał Mróz
Paweł Jastrzębski
Maxim Novak
Piotr Noga

1.4 Terminology

{explanation of terms and abbreviations used in the document, designations used inside the document, e.g. requirements designations}
GUI (graphical user interface)
Frontend – the code responsible for displaying/presenting data, contained in files. qml
Backed – code encoding messages and correcting errors passed by the frontend; implementation of coding. Included in .cpp
GF – Finite Field/Galois Field

2 Technical documentation of the project

{the scope of documentation is decided by the supervisor, here you should start with the description of the technical solution according to the supervisor's recommendations, in an editorial layout that best reflects the nature of the project – subsequent chapters, subchapters, points}

2.1 General:

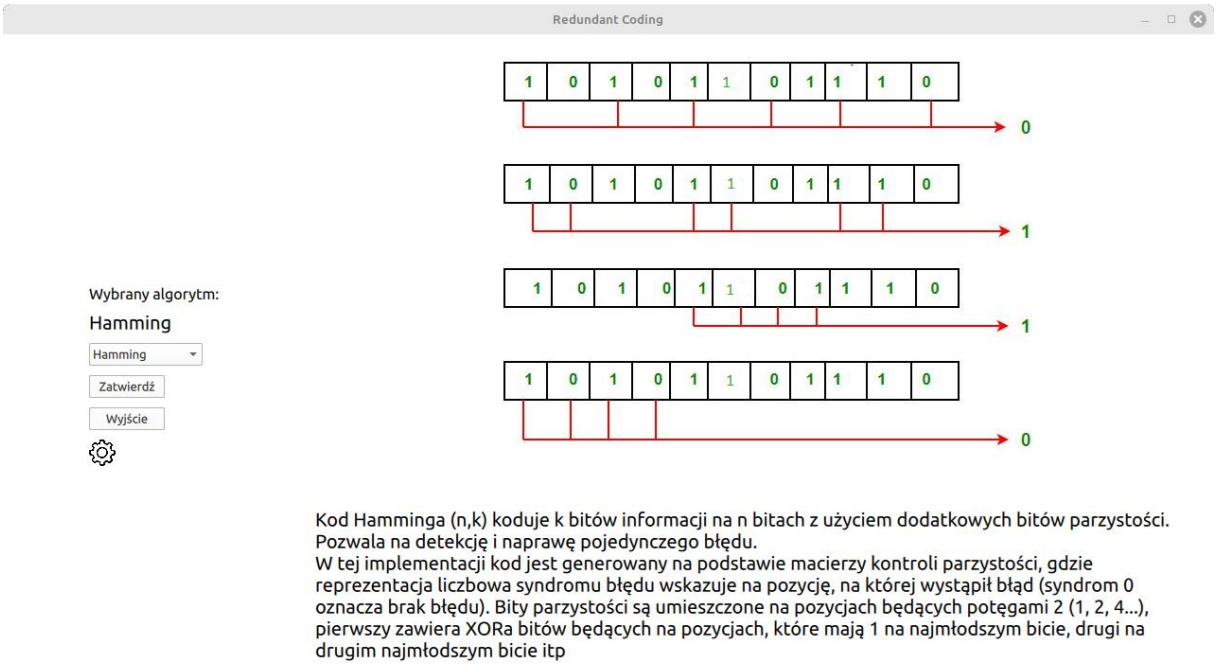
The project is made in C++, in the Qt framework. 2 types of files are used – typical .cpp files that contain the "backend" of the application and files. qml; "frontend", contain information about what is actually to be displayed on the user's screen. The preferred development environment is Qt Creator. The project can be developed on both Linux and Windows. In order to develop the application, you need to install the Boost library on the system. Any other libraries you use (like GoogleTest) are installed automatically.

The final application is exported as a .exe file. In order to generate it, you first need to build the project in QtCreator (preferably on the release profile, not debug), which will generate a directory with a name similar to "build-Redundant-Coding-Visualization-Desktop_Qt_6_6_2_MinGW_64_bit-Release". Next, turn on the MinGW console (you should be able to search for a name like "Qt 6.6.2 (MinGW 11.2.0 64-bit)"), navigate to this directory and run the command "windeployqt appRedundantCoding.exe". You should distribute the entire directory (e.g. as a zip), not the exe file itself, because the directory contains the resources needed to run – such as dll libraries.

2.2 Frontend/Appearance:

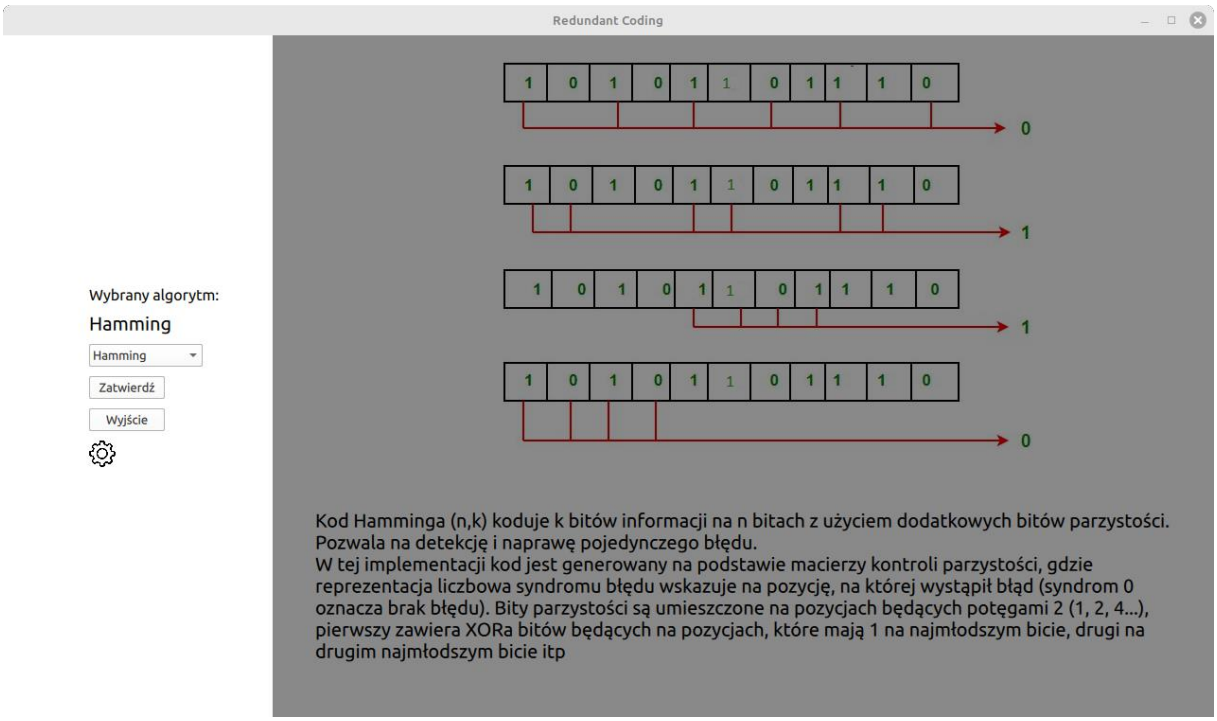
The appearance of individual sections of the program will be presented, i.e. the main menu, data entry for a given algorithm, presentation of its operation. The elements that make up each of the pages appearing in the program will be explained.

2.2.1 Main Menu



Drawing 1 Appearance of the main menu of the program

After starting the program, the user is presented with the main menu (Drawing 1), the source code of which is contained in the file "src_gui/Main.qml". It consists of two areas – "Descriptive" and "Interactive".

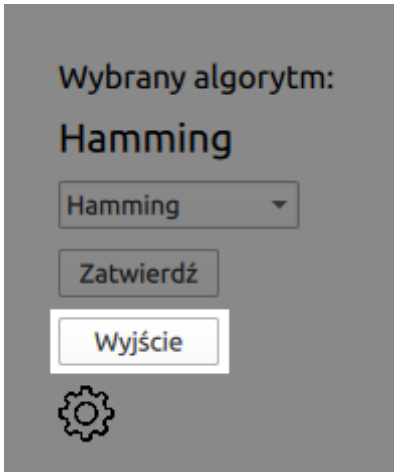


Drawing 2 Highlighted area "Interactive"

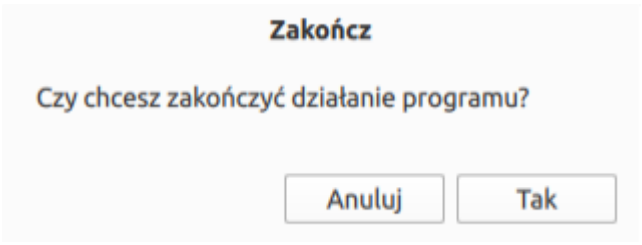
Area 'Interactive' (Drawing 2) contains three buttons – "Settings", "Exit" and "Confirm", a drop-down list and text.

Drawing 3 "Settings" button

The "Settings" button, represented by a gear icon (Drawing 3), takes you to the window with program settings, such as selecting the program language. This window will be presented later in the document.



Drawing 4 "Exit" button



Drawing 5 Confirmation of program closure

The "Exit" button (Drawing 4) causes a window (Drawing 5), in which the user is asked to confirm their decision to close the program.

Drawing 6 "Confirm" button

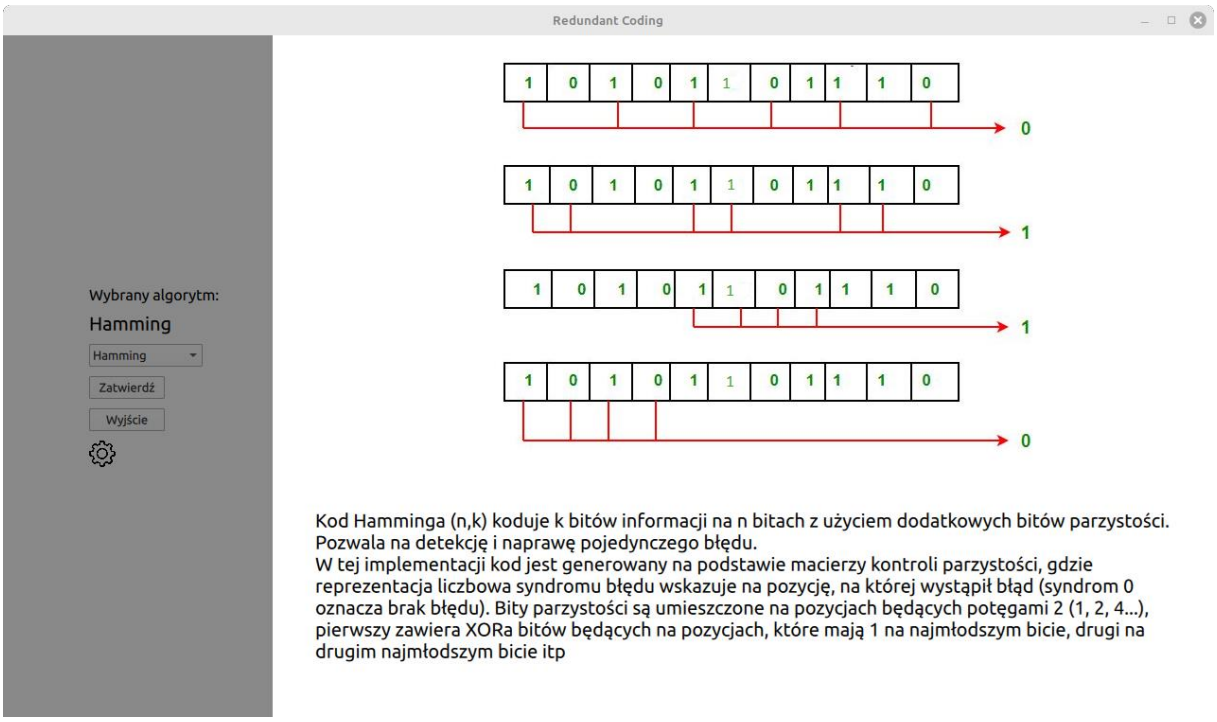
Drawing 7 Text indicating the selected algorithm

Drawing 8 Drop-down list

The last button you see in the main menu is "Confirm" (Drawing 6). After clicking on it, the program goes to the next program window, hereinafter called "Startup data", in which the data on which the program will operate is entered.

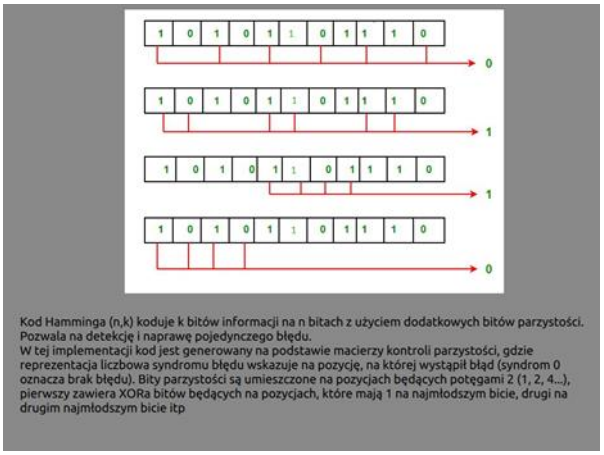
Above the buttons is a drop-down list (Drawing 7), which contains a choice of algorithms that will be presented at a later stage of the programme's operation. In the current state of the application, there is only one working algorithm to choose from – the Hamming algorithm and two non-working options, which are only intended to visualize the possibility of choosing more algorithms in the future.

The text above it is associated with the list (Drawing 8), which displays which algorithm has been selected at the moment.

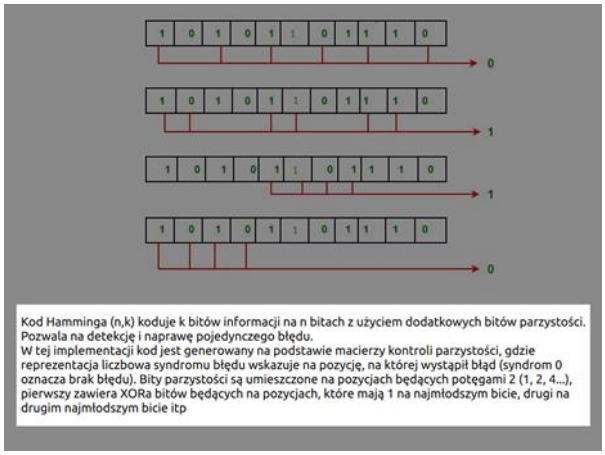


Drawing 9 Highlight of the "Descriptive" area

Area "Descriptive" (Drawing 9) is designed to familiarize the user with the algorithm selected by him in the "Interactive" area. It is made of two elements – an image and a description.



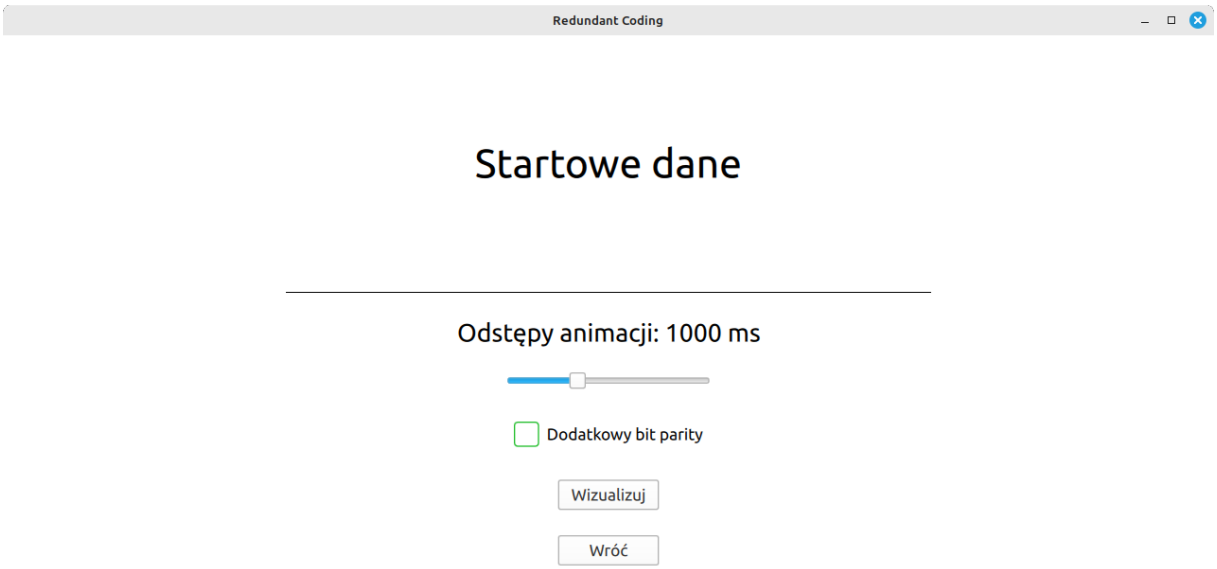
Drawing 10 Graphical presentation of the algorithm



Drawing 11 Algorithm description

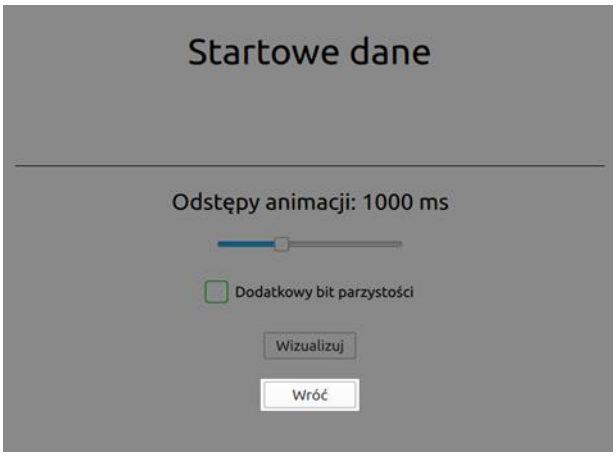
Image (Drawing 10) shows a visualization of the selected algorithm, while the description (Drawing 11) explains how it works.

2.2.2 Starting data

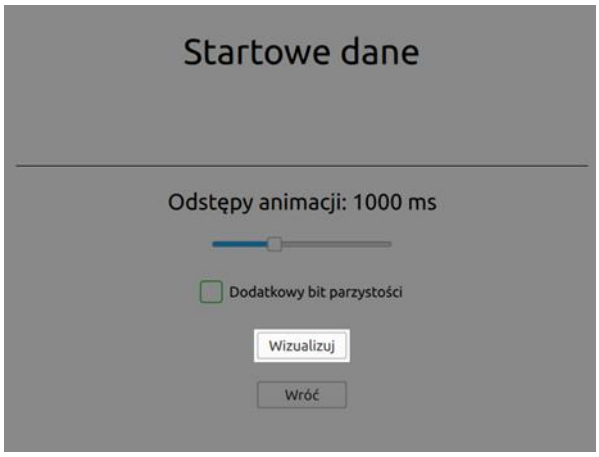


Drawing 12 The "Start Data" window

The next window in the application is "Startup Data" (Drawing 12), which allows you to enter the data on which the program will operate. It contains two buttons - "Back" and "Visualize", a Checkbox button, a slider, text informing about the animation spacing, a text field and a window title.



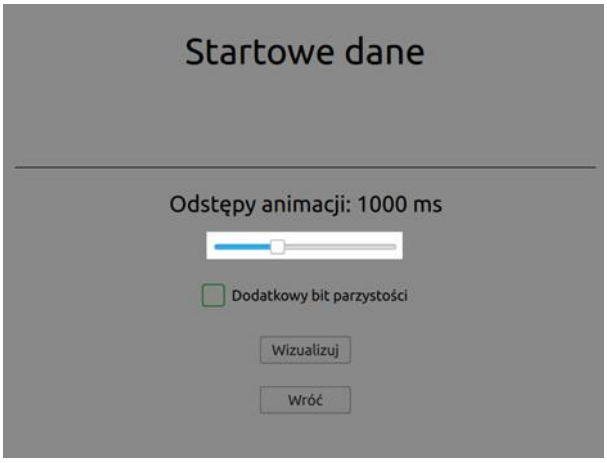
Drawing 13 "Back" button



Drawing 14 "Visualize" button

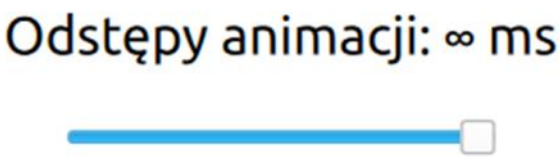
The "Back" button (Drawing 13) allows you to return to the main menu, while the "Visualize" button (Drawing 14) goes to the main part of the application, i.e. the visualization of the algorithm.

Drawing 15 Checkbox button



Drawing 16 Slider

"Extra parity bit" (Drawing 15) introduces an additional parity bit in the algorithm. Slider (Drawing 16) sets the time it takes to wait for the next step of the algorithm to be executed. The time that can be set is in the range (0 seconds - 2.9 seconds).



Drawing 17 ∞ value on the slider

It is also possible to enable the transition to the next step of the algorithm through a button. In this case, set a special value on the slider ∞ (Drawing 17), which is located at the end of the scale of the slider on the right.

Drawing 18 Text box

Drawing 19 Animation spacing

Animation spacing (Drawing 18) inform the user about the time needed to wait to move to the next step of the animation showing how the algorithm works. The time is in milliseconds. Text field (Drawing 19) allows you to enter a numeric string in binary form, which will then be

redundantly encoded. It accepts only zeros and ones. Other characters are omitted, so the application is protected against entering an invalid string.

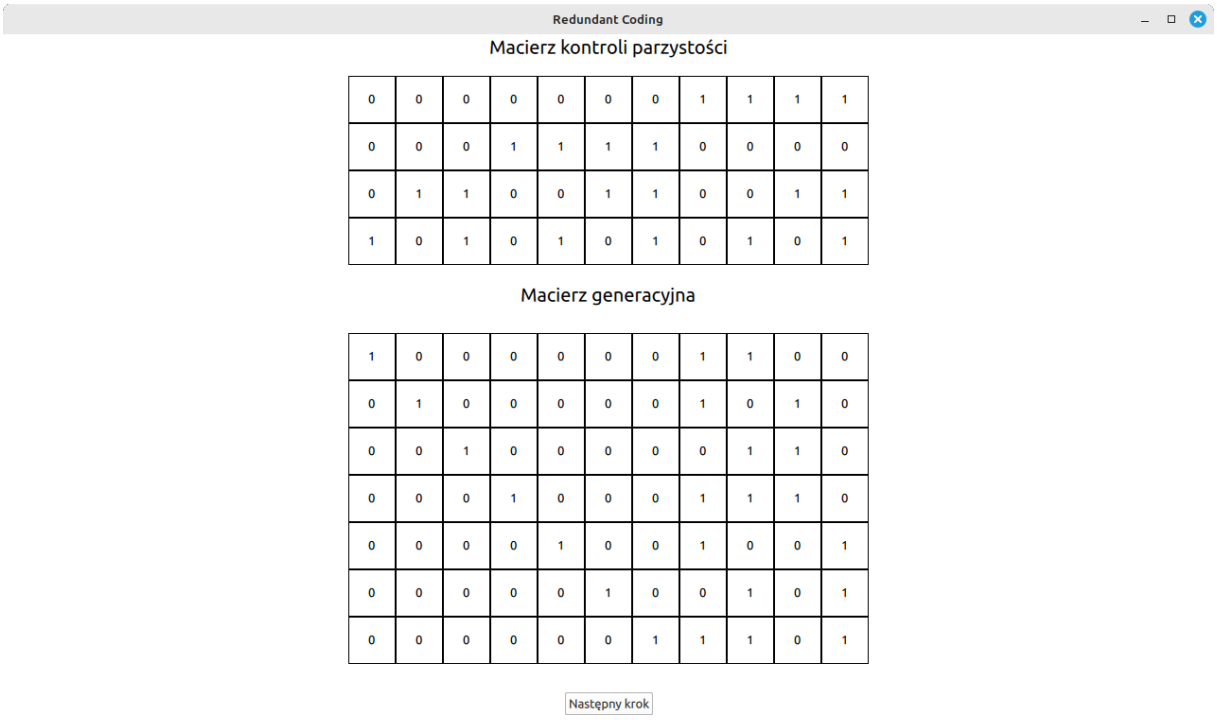
Drawing 20 Window title

The last visible element of the window is its title (Drawing 20).

2.2.3 Algorytm Hamminga

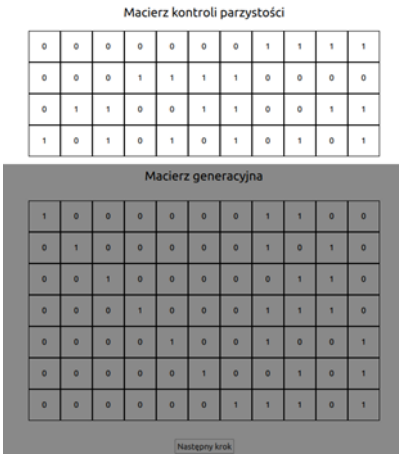
In the following subsections, all windows occurring during the operation of the algorithm will be presented, including individual elements of these windows.

2.2.3.1 Matrices

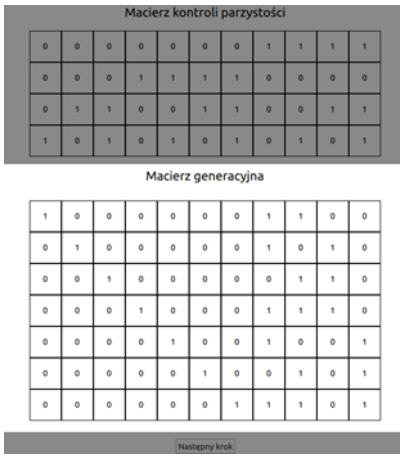


Drawing 21 The "Matrices" window

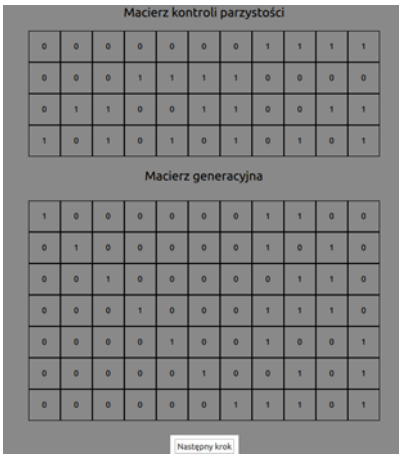
The next window in the program is the "Matrices" window (Drawing 21).



Drawing 22 Parity Control Matrix



Drawing 23 Generation matrix



Drawing 24 "Next step" button

It mainly shows the parity control matrix (Drawing 22) and the generation matrix (Drawing 23). The code responsible for this window is located in the file "src-gui/HammingGenerationMatrix.qml". Both matrices shown in this window are dynamically generated based on the numerical sequence provided by the user in the previous window. Creating them in a graphic form is presented in the chapter Matrix visualization. In addition to matrices, the window also contains the titles of the matrices above them. At the very bottom of the window, there is a "Next step" button (Drawing 24), which takes the user to a window that demonstrates the operation of the algorithm.

2.2.3.2 Plik Hamming.qml

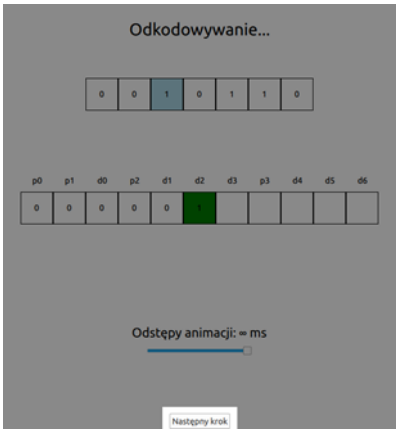
After typing a numeric string in the "Starting data" and move on, a significant part of the program related to the visualization of the Hamming algorithm is located in the "src_gui/Hamming.qml" file. It is the most advanced window related to this algorithm, because apart from the last window "", the rest of the application is executed based on this file. In the following sections, unless otherwise noted, the section will deal with the content of "src_gui/Hamming.qml".

2.2.3.3 Decode

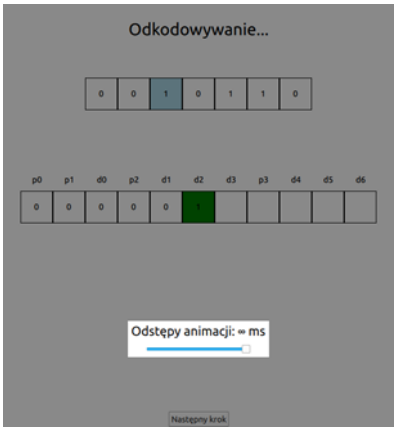
Drawing 25 "Decode" window before saving the redundant matrix

Drawing 26 "Decode" window after saving the redundant matrix

The actual presentation of the algorithm's operation begins in this window (Drawing 25 and Drawing 26). This window is now made up of two matrices, a slider and a "Next step" button if the slider is set to ∞.



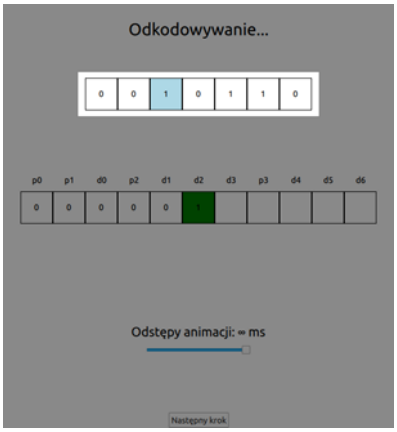
Drawing 27 "Next step" button



Drawing 28 Slider

If you see a "Next step" button in the window (Drawing 27), it allows you to move to the next step in the operation of the algorithm. Slider (Drawing 28) as in the "Starting data" sets the time between the steps of the algorithm.

Drawing 29 Redundancy Coding Matrix



Drawing 30 Original number sequence matrix

The matrix in the center of the window is the redundancy coding matrix (Drawing 29). Above the individual cells of the matrix there are their markings, "pX" or "dX". "p" stands for the bit, which is the parity bit, and "d" for the bit in the number sequence. "X" should be understood as the number of the bit in its type. Examples of notation look like this:

- "p3" should be read as: the parity bit being the third in the order.
- "d5" stands for, respectively: the bit of the numerical sequence, which is the fifth in the order.

It is worth mentioning that the number "X" **does not mean** that the bit is at a given position in the matrix. For example, the bit "d5" shown in Figure 29 is not in position number 5, but in position number 9, because it is also preceded by parity bits from 0 to 3.

Above the redundancy matrix is a matrix with a numeric string previously entered by the user (Drawing 30).

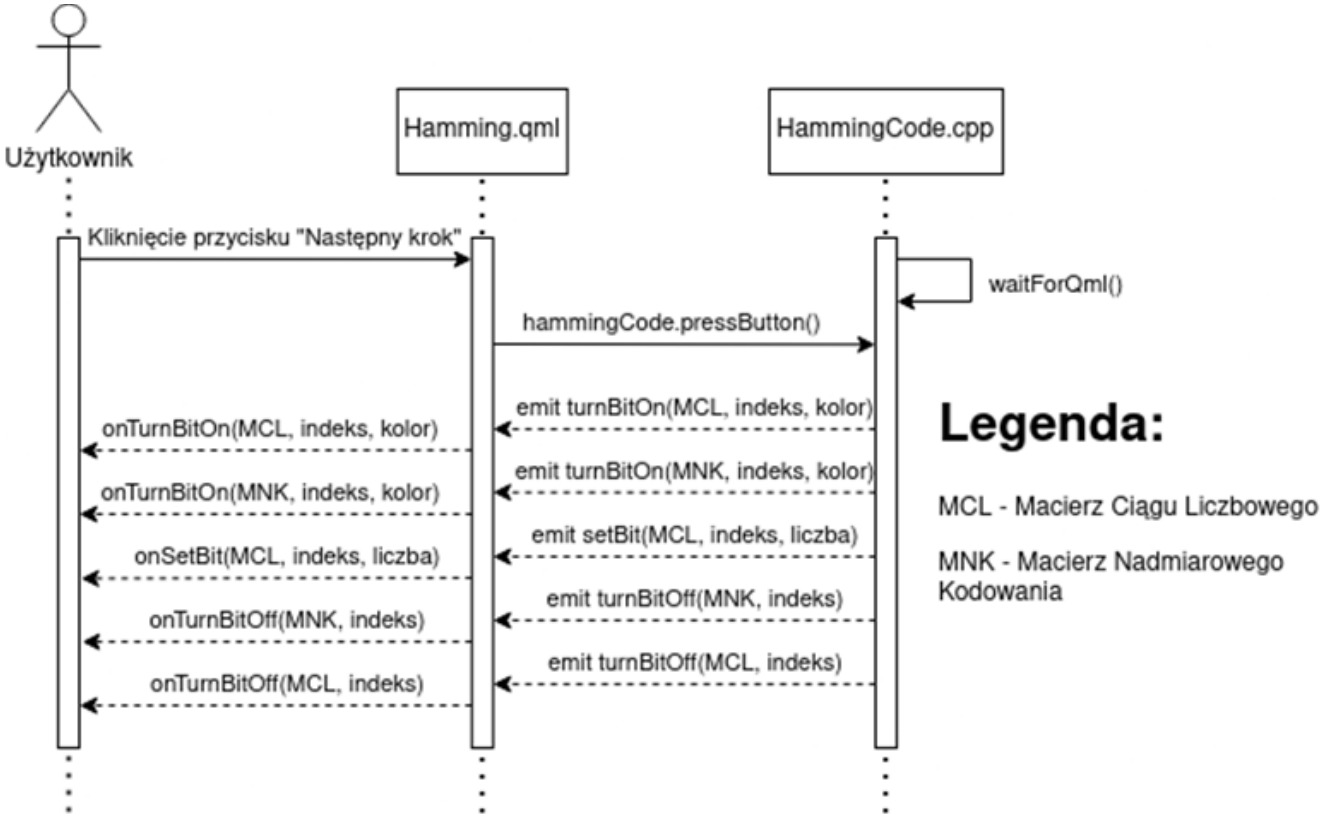
The general way of visualizing the matrix is presented in the chapter "Matrix visualization", while this window also applies an additional procedure to the redundant encoding matrix. As you can see in Figures 25 and 26, the matrix is not filled in at the beginning, but at the end it is fully written. The method of filling in the book is presented in the chapter "Matrix populating in Hamming.qml".

2.2.3.4 Matrix populating in Hamming.qml

The backend function is responsible for populating the matrix at this stage of the program's operation *encodeDataAsync* located in the "src/HammingCode.cpp" file. It uses the functions contained in the current window file *onTurnBitOn, onTurnBitOff* and *onSetBit*. With the *onTurnBitOn* Backend feature *encodeDataAsync* sets the appropriate color for a given matrix cell, i.e. for bits that are parity bits, sets the color red, and for bits that mark the next bit in the numerical sequence, the color is green. Function *onSetBit* allows you to enter the value of a given bit into the matrix. *OnTurnBitOff* sets the background of the given matrix cell back to white when you need to move to the next cell.

As the bits are written to the redundant encoding matrix, the above-mentioned backend *encodeDataAsync* function also changes the background color of the corresponding cells of the numeric sequence matrix, on exactly the same principles as before, without typing the bit values

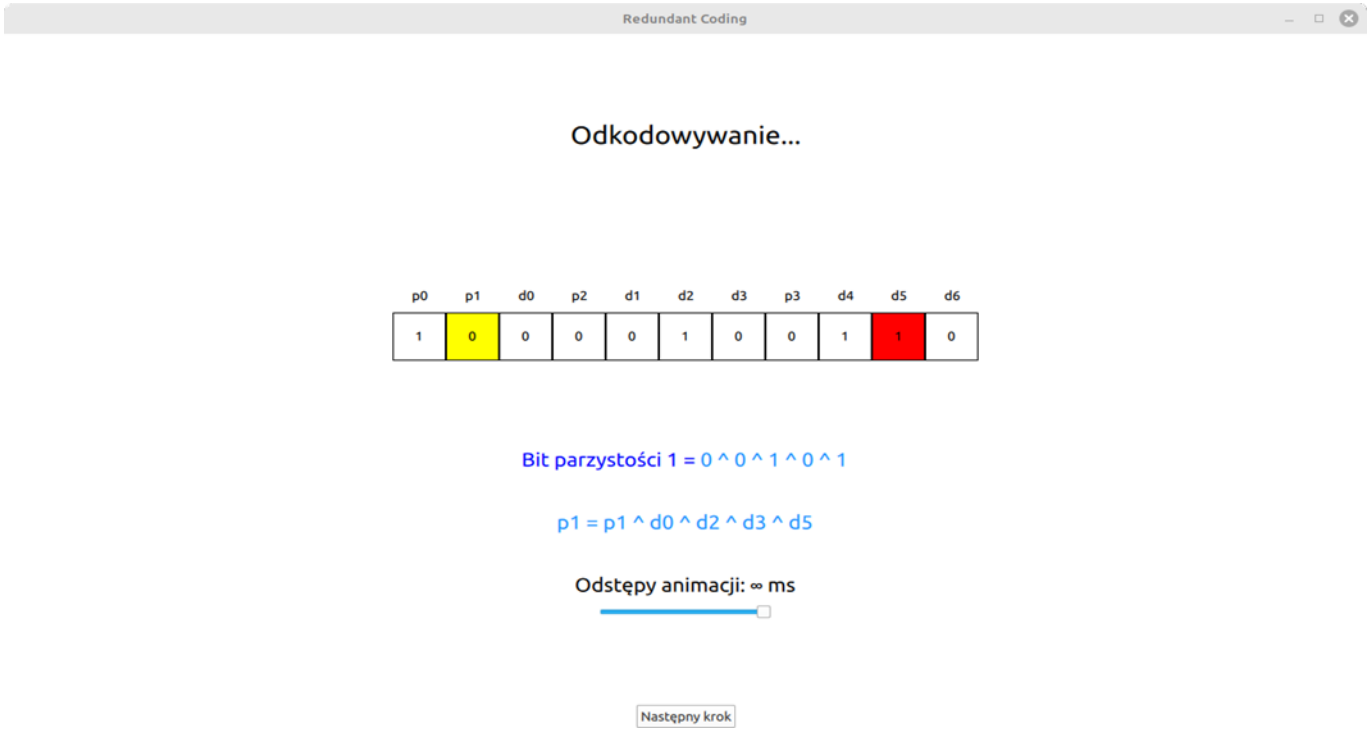
with *onSetBit*, i.e. *onTurnBitOn* changes the background color to light blue and *onTurnBitOff* undoes this change.



Drawing 31 Simplified animation sequence diagram of filling matrix cell redundant encoding along with number sequence matrix animation

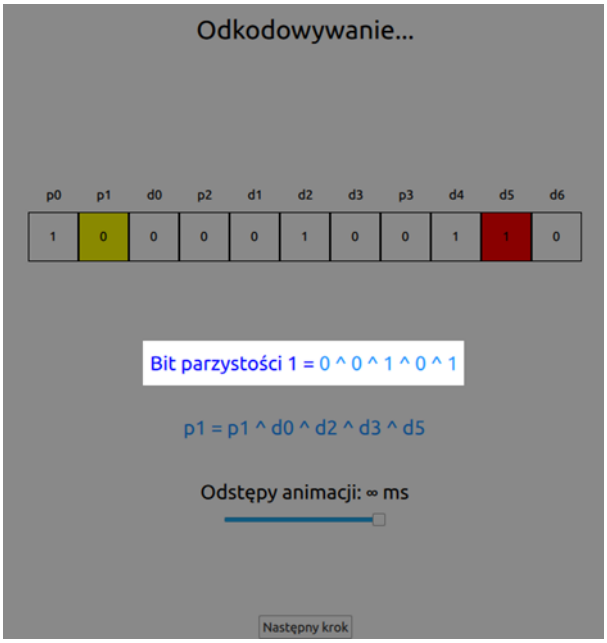
On Drawing 31 A simplified sequence diagram is presented, which shows how the application works when it animates the filling of a given cell of the redundant coding matrix.

2.2.3.5 Calculating parity bits

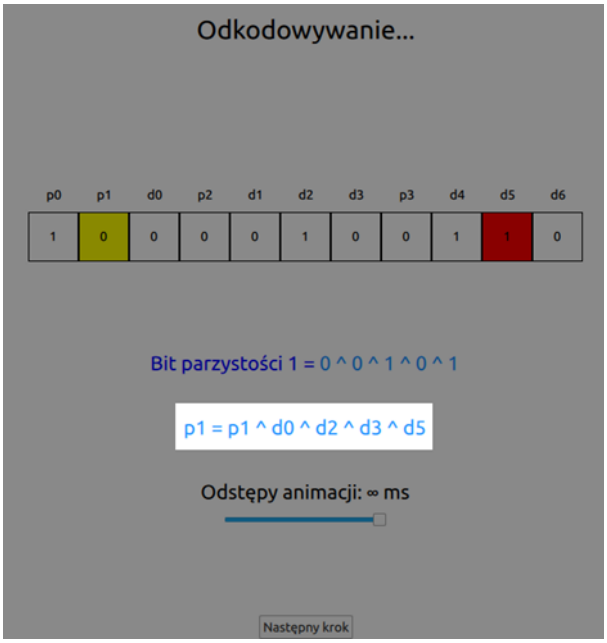


Drawing 32 The "Parity Bits Calculation" window

After populating the matrix with redundant encoding, you now need to calculate the values of each parity bit, because zeros were typed in the population by default. The window itself has not changed, because it is still "Decoding", as the title in it says, but the appearance of this window is slightly different from the previous one (Drawing 32). The number sequence matrix disappeared from the view and only the matrix with the redundant code remained.

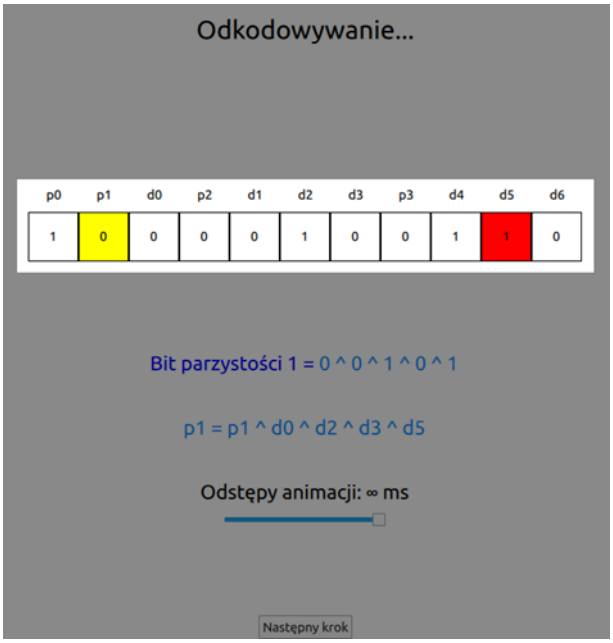


Drawing 33 The text "Calculations"



Drawing 34 Text "Template"

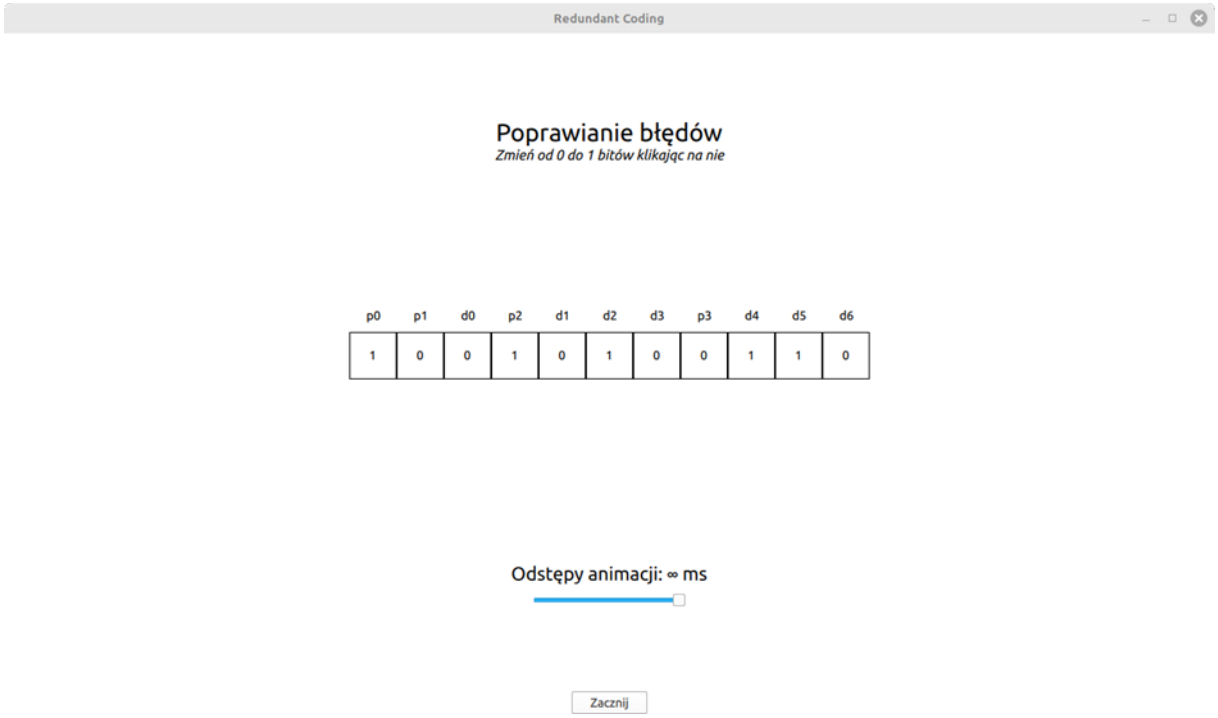
Two texts appeared under it. The first text of "Calculations" (Drawing 33) contains the calculation of the parity bit in question. The second text, called "The Pattern" (Drawing 34) contains the formula used in the text 'Calculations'.



Drawing 35 Parity bit calculation animation

Both texts interact with an animation that shows the calculation of the parity bit value (Drawing 35). Changing the colors of individual cells works in a similar way to changing colors in a numerical sequence matrix, which is described in the chapter "Matrix populating in Hamming.qml" with minor differences, which will now be briefly outlined. At this point, it is two cells at once that have changed color. The parity bit under consideration has its background color changed to yellow, while the bit taken into account at a given moment, depending on its value, changes its color to red (when the value is 0) or to green (when the value is 1).

2.2.3.6 Correcting errors



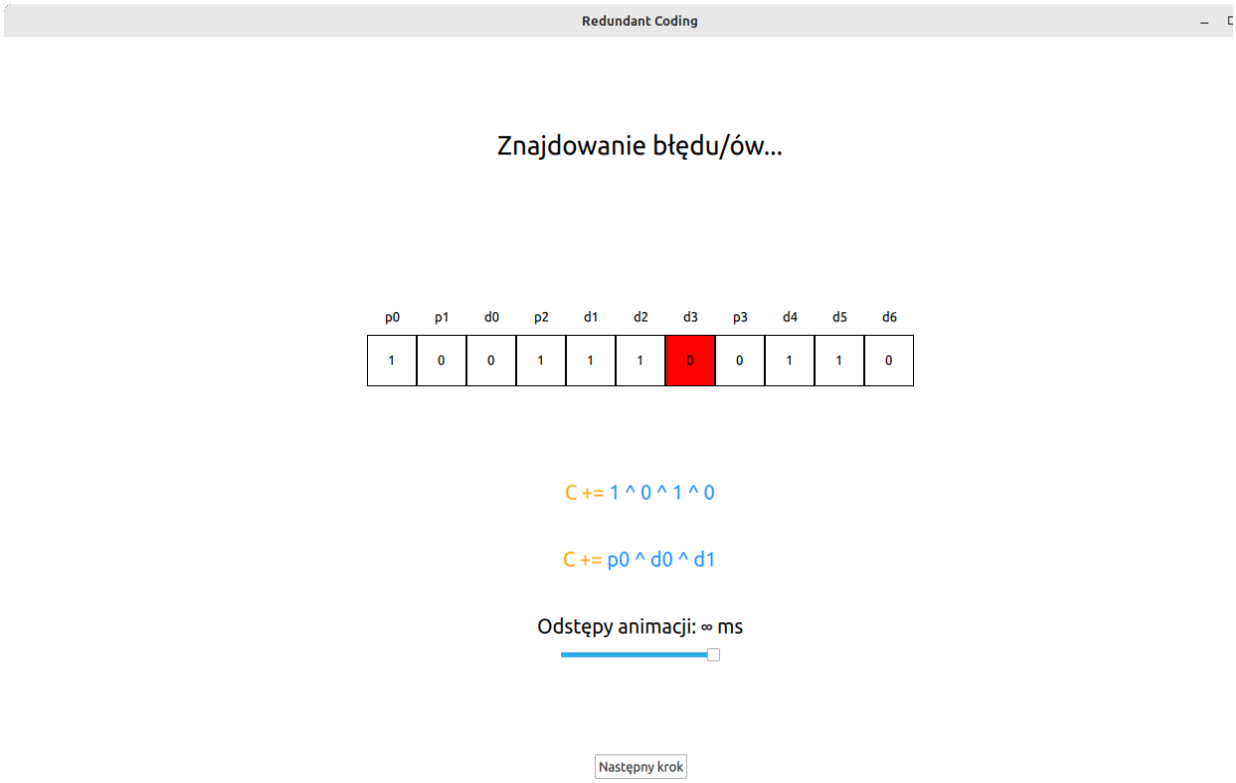
Drawing 36 The "Correcting Errors" window

After changing the parity bits in the redundant encoding matrix in the previous window appearance, the user has the option to change one bit (or two bits if an additional parity bit is selected). In the chapter "Matrix visualization" has been mentioned that the structure of a single element that makes up a matrix contains information about mouse operation. At this point, it is used, so that after clicking on the indicated cell of the matrix, the user changes its value.

Drawing 37 Changed bit with index 5

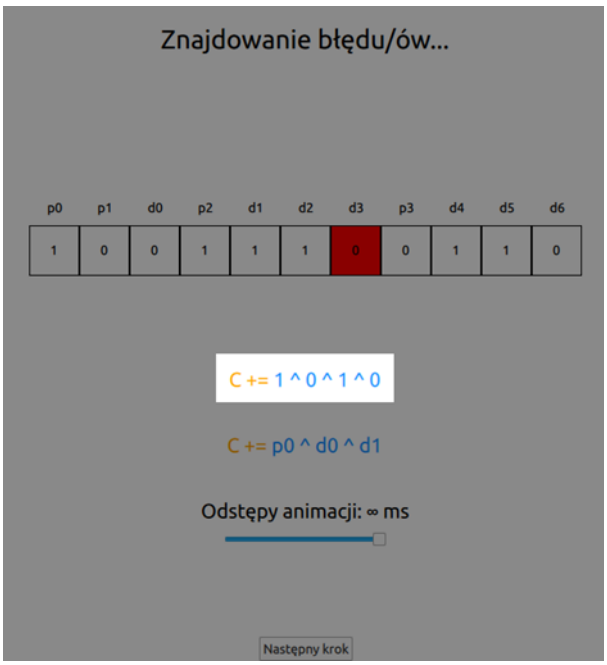
For demonstration purposes, the bit "d1" has been changed, which is located at the position with index 5 (Drawing 37).

2.2.3.7 Find the error(s)

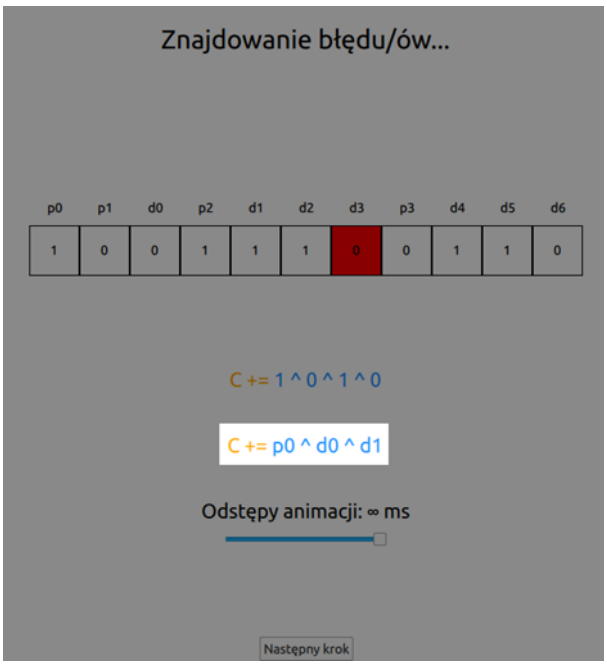


Drawing 38 "Find Error(s)" Window

This window appearance is almost the same as in the chapter "Calculating parity bits" with a slight difference in the text below the matrix and in the animation, where only one cell of the matrix is changed color at a time.



Drawing 39 The text "Calculations"



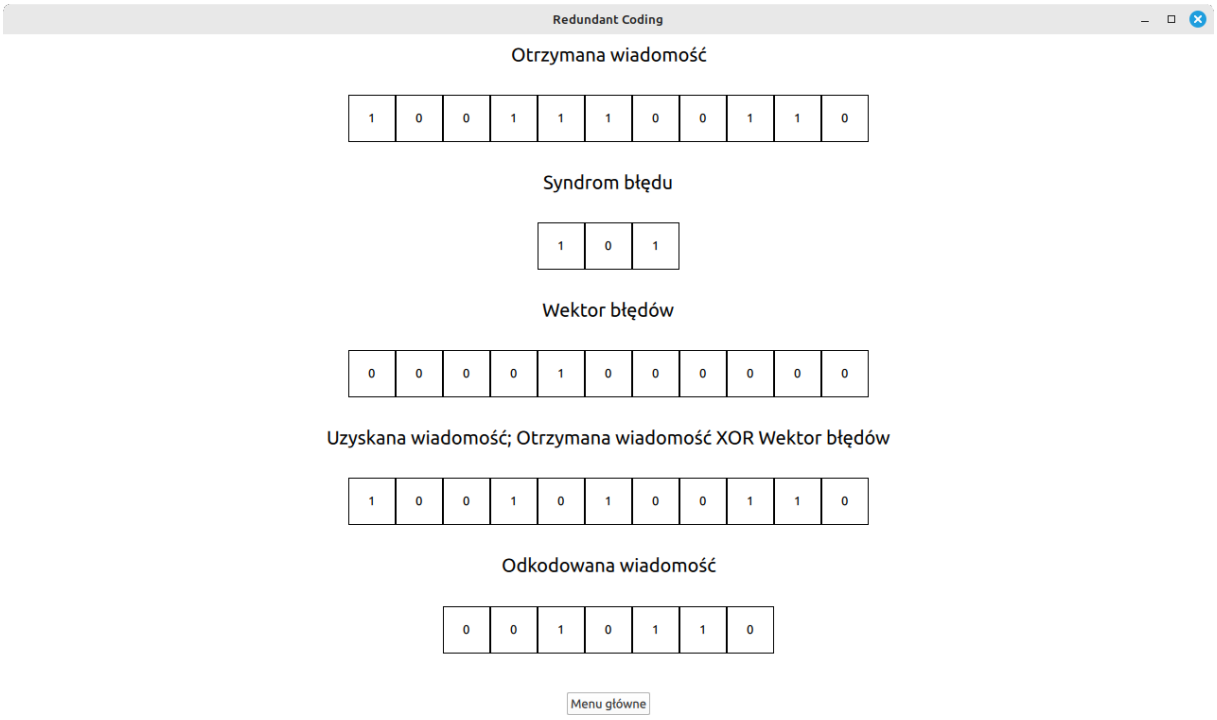
Drawing 40 Text "Template"

As in the aforementioned chapter, the first text 'Calculations' (Drawing 39) contains the calculation, but this time the number C is calculated. This number indicates the index at which the bit was changed by the user in the previous window appearance. The second text 'Model' (Drawing 40) contains the formula used in the text "Calculations".

Drawing 41 End screen of the "Hamming.qml" file

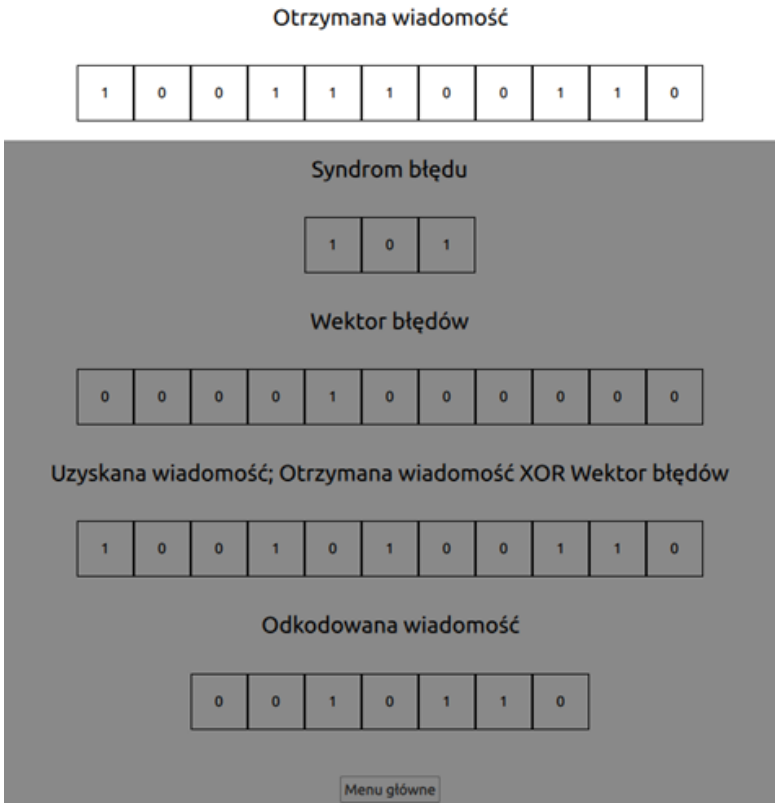
After the error or error search animation is complete (Drawing 41), in place of the text "Template", text appears informing on which position the error is.

2.2.3.8 End screen



Drawing 42 End screen of the Hamming algorithm

The last window that appears in the process of presenting the Hamming algorithm is located in the "src_gui/HammingSyndrome.qml" file. It shows the 5 matrices, the titles above them, what each matrix means, and a button to return to the main menu.

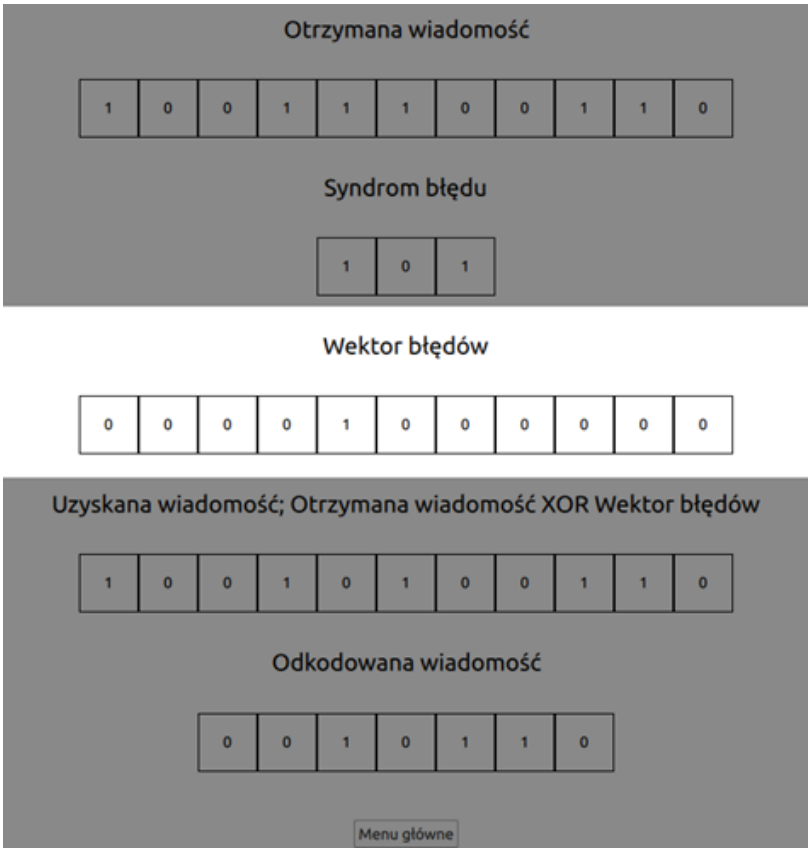


Drawing 43 Message Received Matrix

The "Message Received" matrix (Drawing 43) means redundant encoding in which a bit has been deliberately changed by the user.

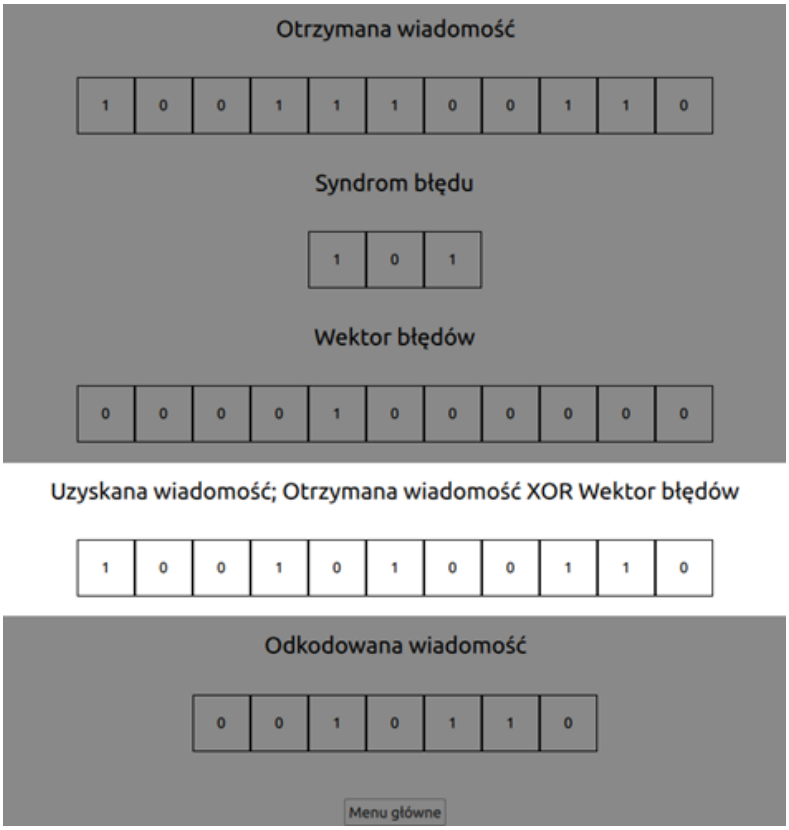
Drawing 44 The "Error Syndrome" matrix

'Error syndrome' (Drawing 44) is a matrix in which the index of the position is stored in binary form, in which the changed bit by the user is contained.



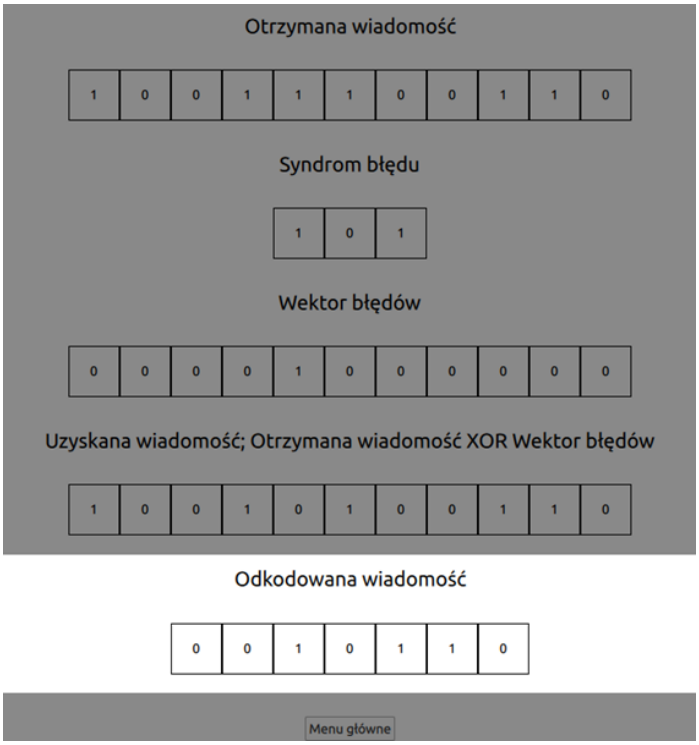
Drawing 45 Error Vector Matrix

'Error vector' (Drawing 45) is a matrix of the same size as the original redundant encoding matrix, which is completely zero-filled, except for the bit at index that the original redundancy code failed to mention.



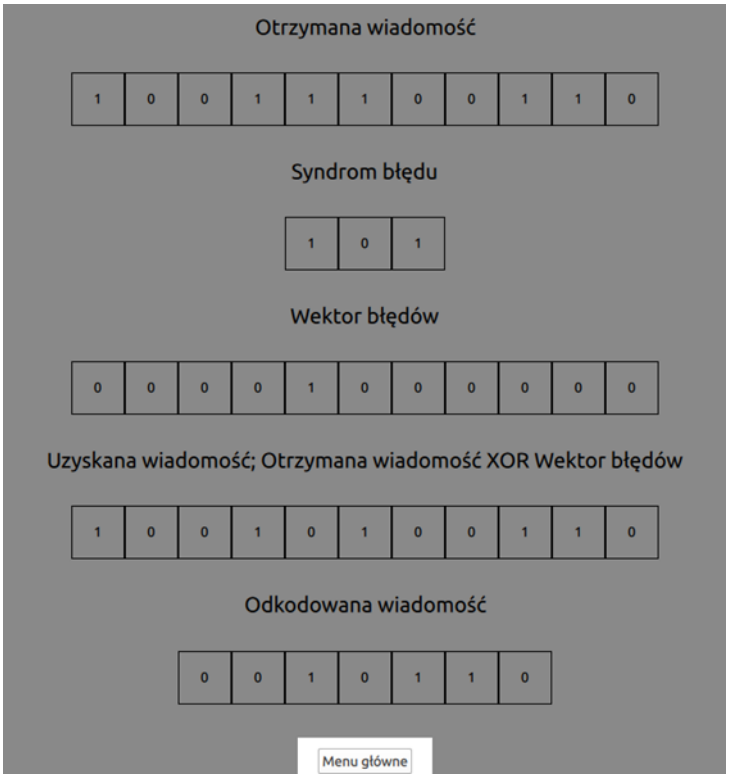
Drawing 46 Message Received Matrix

'Message obtained' (Drawing 46) is the matrix resulting from the disable alternative operation of the "Message Received" matrix with the "Error Vector" matrix.



Drawing 47 "Decoded message" matrix

The last of the matrices in this window is the "Decoded message" (Drawing 47), This is the result of the algorithm. The numerical sequence formed from the bits of this matrix should be equal to the one specified by the user at the beginning of the algorithm's operation, if the redundancy encoding, detection and repair of the error were carried out correctly.



Drawing 48 "Main Menu" button

At the bottom of the window there is a "Main Menu" button (Drawing 48) that takes you back to Start.

2.2.3.9 Matrix visualization

The matrix is the basis of the presented program. In many application windows, they are generated in various forms, be it a single numerical sequence or in the form of complex generation matrices, occupying a significant part of the window in which they are located. The files responsible for creating them in a graphical form, visible to the user, are located in the "src_gui/VisualizeComponents" folder. The process of creating their visualization in the "ArrayRowLayout.qml" file is as follows:

1. By calling the file, the value of the variable of type string *myArr*, which stores the numeric string for a given order of the matrix, is changed.
2. On the basis of the length of the binary sequence (how many numbers are contained in a given sequence, in this case the digits 0 and 1), which is entered in the above-mentioned variable, the width of the matrix is determined.
3. The Repeater function repeats the drawing of given elements as many times as the value of *the model variable*. It is this function that allows you to graphically represent a given order of matrices. The model variable therefore takes the value of the matrix width.
4. The element that is cyclically drawn by the above function is the *Rectangle type*. It contains the following information:

1. what is the size of the rectangle that represents a given cell of the matrix,
2. the text that is contained in the rectangle, i.e. the number located in a given place,
3. how the area of the drawn rectangle reacts when you click on it with the mouse, which allows you to change the bit contained in a given matrix cell (MouseArea type and onClicked() function).

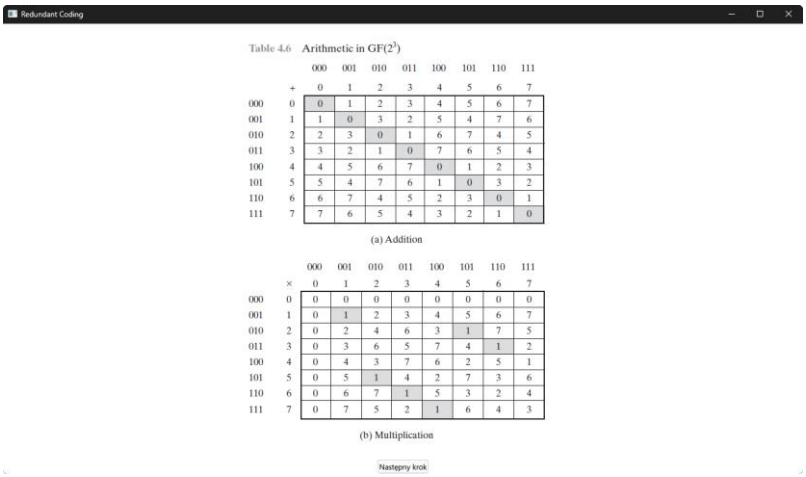
That element is therefore a cell of a given matrix, and the composition of them in the Repeater function constitutes a row of matrices.

At this point, the matrix row is already visible on the screen. If an application is tasked with displaying a matrix with more rows, it calls the file as many times as its height, so for example, for a matrix that is five times high, the file "ArrayRowLayout.qml" will be called five times, giving the impression that the matrix drawn in the window looks uniform.

2.2.4 Algorytm Reeda-Solomon

In the following subsections, we will present the individual screens that occur during the operation of the Reed-Solomon algorithm. All screens except the first one are in the "ReedSolomon.qml" file, which will be further noted in the following subsections.

2.2.4.1 Galois



Galois Screen Drawing

This is the first screen available only in the Reed-Solomon algorithm. It presents the arithmetic used in the rest of the program. The screen consists of two parts - two tables and a "Next Step" button. This is the only screen that is **not** in the "ReedSolomon.qml" file. It is in "Galois.qml".

2.2.4.2 General overview of "ReedSolomon.qml"

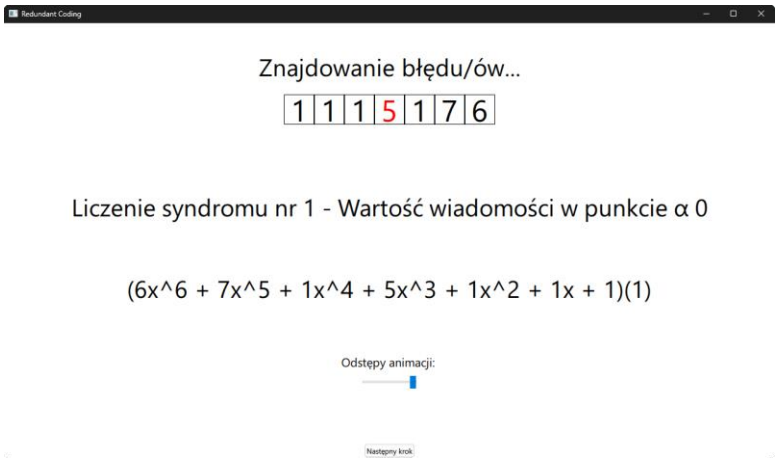
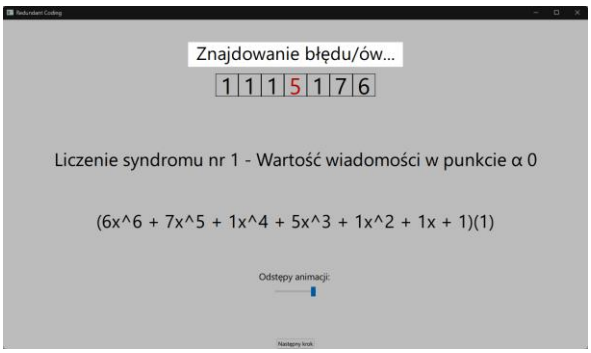


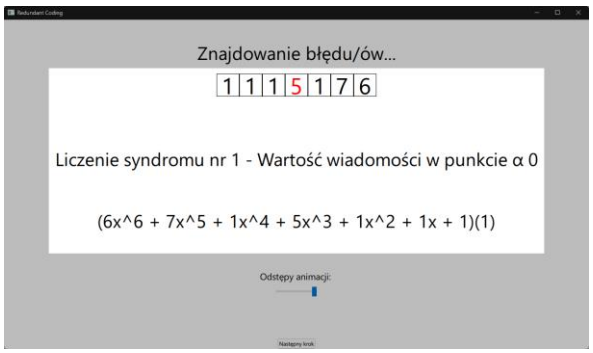
Figure Sample screen from the file "ReedSolomon.qml"

The screen consists of the following parts: Title ("Find Error(s)"), "Workspace", slider ("Animation Spacing:") and if you select the Animation Spacing to ∞ the "Next Step" button.

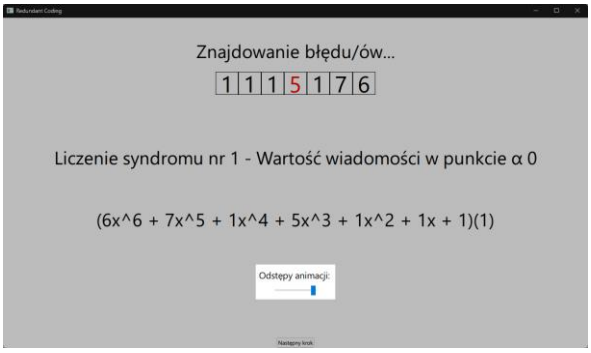
- Title: Identifying what general action is taking place on the screen, e.g. "Decoding", "Finding error(s)", "Finding the position of the error", etc. (Drawing)
- "Workspace": a field in which the detailed operation of the algorithm is presented, i.e. the formulas used are shown, what is currently being calculated, operations on numbers. (Drawing)
- "Animation intervals" slider: allows you to set the time between successive steps performed in the algorithm. (Drawing)
- Next step button: Move to the next section of the program if you have selected a value ∞ the animation interval. (Drawing)



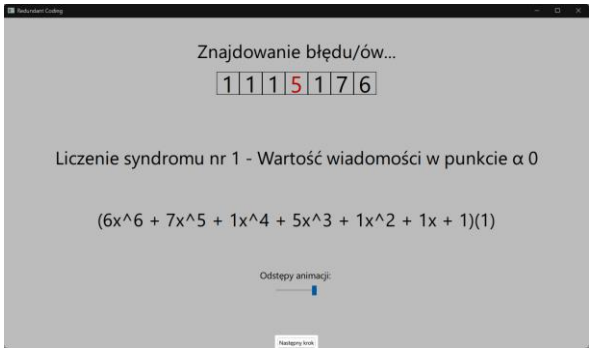
Title Figure



Drawing "Workspace"



Drawing "Animation Spacing" Slider



Drawing "Next Step" Button

2.2.4.3 Decode

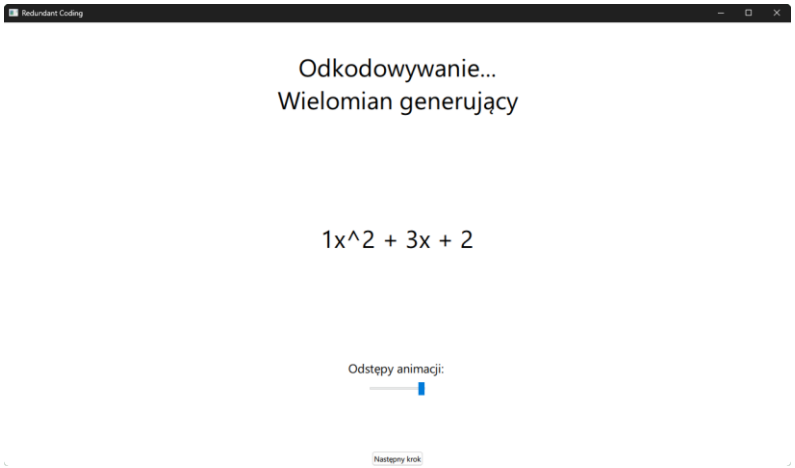


Figure Decoding Screen with subtitle "Generating Polynomial"

At the beginning is the generating polynomial, which is used in the algorithm. The polynomial formula is presented in the "Working Area".

Figure Rewrite Input Data

Then the program rewrites the values provided by the user in the "Starting data" screen to the matrix on which the Reed-Solomon algorithm will be performed. This matrix appears at the top of the screen, just below the title.

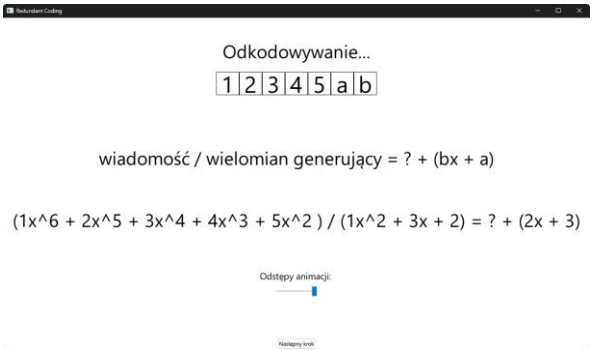


Figure message / polynomial generating before calculating the values of the numbers *a* and *b*

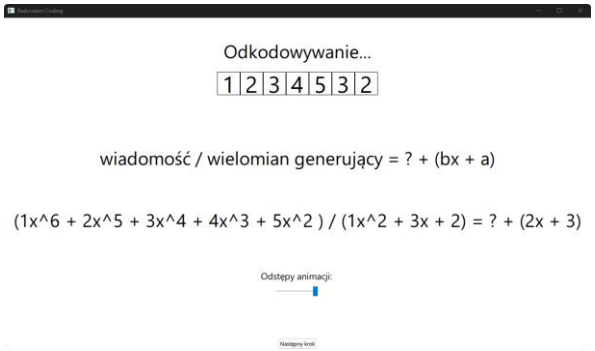


Figure Calculated values *of a* and *b*

A screen now appears in which the values *a* and *b* are calculated. The screenshot (Figure) on the left shows the situation before these values were calculated, while the screenshot on the right (Figure) shows the values already calculated.

2.2.4.4 Correcting errors

Figure Correcting errors with a set error

In this screen, the matrix appears along with the additional numbers that were calculated in the previous screen. You can choose in which position of the message you want the error to appear, or you can leave it unchanged.

Later in the discussion of screens, we will show an example (message 12345) in which the user set the wrong number to the fourth position (he changed the correct number 4 to the wrong number 1). The changed number is highlighted in red compared to the other numbers, which are colored black.

2.2.4.5 Finding bugs

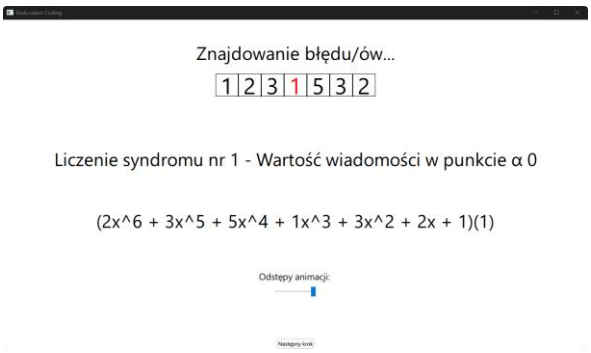


Figure Counting Syndrome No. 1

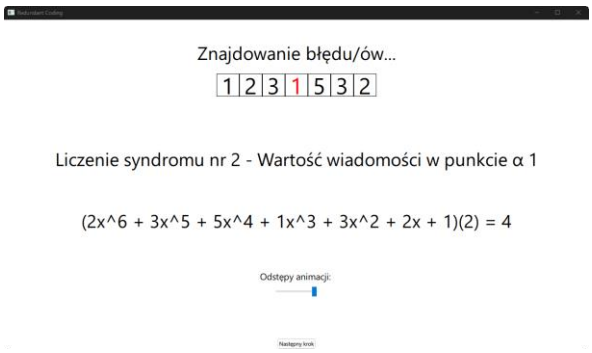


Figure Counting syndrome No. 2

At this point, the program checks to see if there is an error in the message. Counting of individual syndromes is shown.

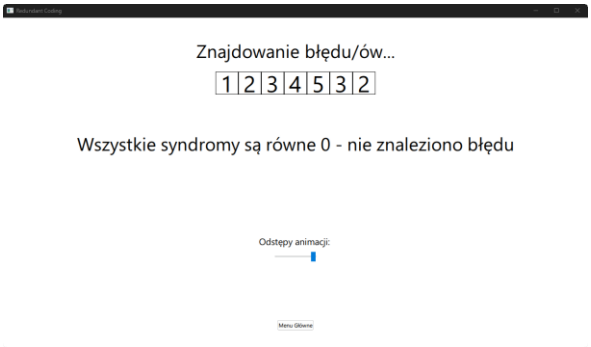
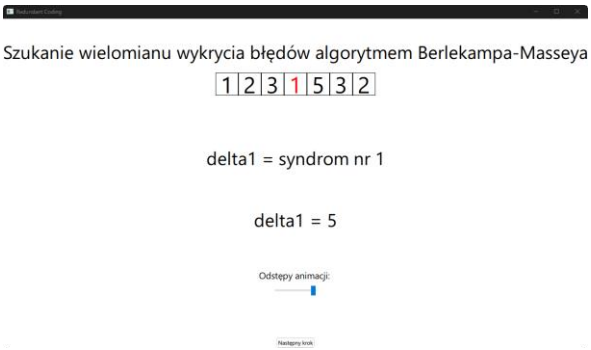


Figure Valid Message - No Errors Found

Figure Wrong Message - Errors Found

Depending on whether the program found the error in the message, the corresponding information appears, as you can see in the screenshots above. If the user has not set any error in the message, the algorithm terminates its operation at the moment and after pressing the "Main menu" button, the program is returned to the start screen.

2.2.4.6 Finding the polynomial of error detection



Delta 1 calculation figure

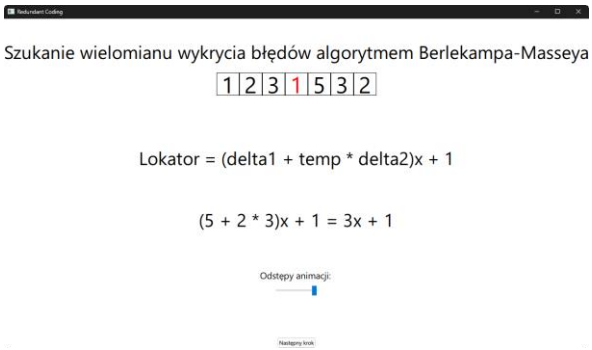


Figure Calculating Tenant

In the next screen, the polynomial of error detection is searched. In the "Workspace" you can list the patterns or variables from these formulas one by one. Apart from the changes in the aforementioned "Workspace", nothing changes, so they will only be described in a few sentences. First, delta 1 is calculated, then a temporary temp variable. Then the delta of 2 is calculated, which, together with the previous variables, is used to calculate the Tenant. After calculating the variables, the error detection polynomial (i.e. Locator) appears and we move on to the next part of the screen presented in the next chapter.

2.2.4.7 Finding the Error Position

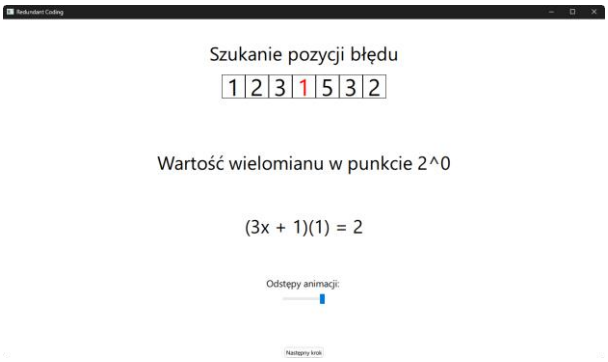


Figure Calculating polynomial values in points

The program already knows that there is an error in the message. In this screen, he goes to search for which position he is in. It uses the Chien algorithm for this. As in the previous screen, only the content of the "Workspace" changes, so the changes in the mentioned field will also be presented. First, the values of the polynomial are calculated at subsequent points until the value of one of the points is 0.

Figure Enumeration of 0

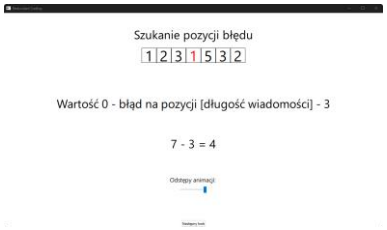


Figure Error Position Calculation

Figure Result of Chien's algorithm

After the program shows the incorrect position according to Chien's algorithm, the program tries to find the correct value at the place of the error, which is presented in the next chapter.

2.2.4.8 Looking for the size of the error

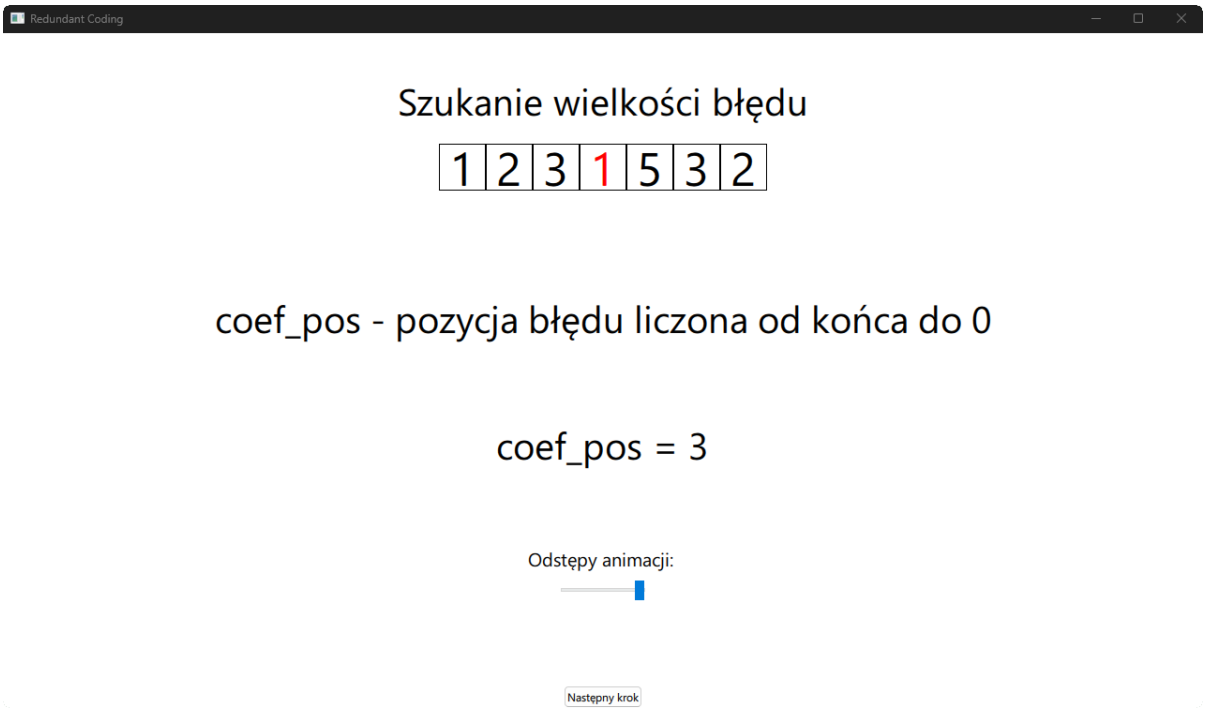


Figure Error Position

Another part of the algorithm that is performed in the program is to find the correct value in the position marked as incorrect. As in the previous chapters, the screen content changes only in the "Workspace", so here too we will only show how the algorithm works. First, the coef_pos variable takes the value of the position determined by the error algorithm. A errata_locator, or polynomial, is then calculated, which is used to identify the location of both errors and corrections in the codeword being submitted. After this step, you can calculate the error_evaluator, which will be used to calculate the error value itself in the place previously indicated by the algorithm. Next, the values of xi and y are calculated. Finally, the size of the error is determined, followed by the correct message.

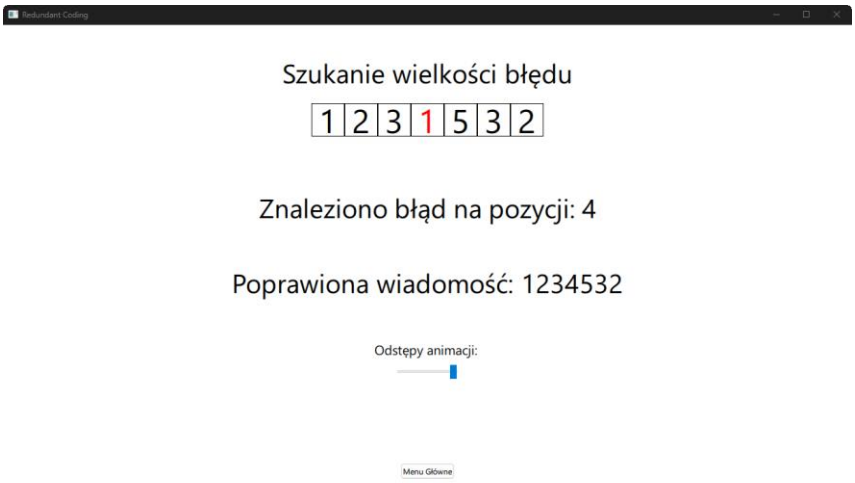


Figure End screen of the algorithm. Correct indication of incorrect position and corrected message

At the very end of the algorithm, the user can see where the program found the error and what the correct message is in their opinion.

2.3 Implementation of the codes:

The general structure of the implementation of both encodings is similar – there is a class responsible for encoding containing the following methods:

1. setInitialData – setting the initial message that will be encoded
2. encodeData – encoding the message set in *setInitailData*
3. sendCode – setting the received message (it can be identical to the one returned by *encodeData*, or changed – simulating errors in transmission)
4. correctError – searching for errors in the message and correcting them

The *encodeData* and *correctError methods* have the forQML flag in the parameters – to distinguish between a call by the frontend and by tests. For tests, it is enough to perform coding/decoding, and in the frontend version, additional messages are sent informing the frontend about the actions being performed in order to show them to the user.

The mechanism from Qt is used for this, where in the .cpp file you can use the emit syntax *FunctionName*, and in .qml you need to implement *the onFunctionName* function.

After each message, the backend waits for the time set in the GUI slider and then resumes its work.

Calculations for the frontend must be called asynchronously, on a new thread, for which the QtConcurrent::run mechanism is used. If the window is closed, the program informs the thread

about it in the background and waits for the end of the operation so that no zombie process is left in the system.

2.3.1 Hamming

The implementation of the Hamming code is in the src/HammingCode.cpp file, and the header is in include/HammingCode.hpp. The Hamming code supports 2 versions – with or without an additional parity bit. The algorithm is implemented using a parity control matrix in which the decimal representation of the syndrome points to the position at which the error occurred, for example for Hamming(7,4) code

0	0	0	1	1	1	1
0	1	1	0	0	1	1
1	0	1	0	1	0	1

Syndrome 011 in the decimal representation is 3, i.e. the error is in position 3 (counting from 1). This eliminates the need to generate and remember the error parity control matrix and the generation matrix.

Parity bits are set to powers of 2 (1, 2, 4, ...). The remaining bits are data, for example:

Position	1	2	3	4	5	6	7
p – parity bit, d – data bit	p1	p2	d1	p3	d2	d3	d4

Consecutive parity bits can be enumerated as XOR bits at those positions that have "1" in the binary representation on the youngest bit (p1), second youngest bit (p2), etc.

Example for p1:

Position	1	2	3	4	5	6	7
Binary representation	001	010	011	100	101	110	111
Designation	p1	p2	d1	p3	d2	d3	d4
Should you use it in XOR?	✓		✓		✓		✓

Example for p2:

Position	1	2	3	4	5	6	7
Binary representation	001	010	011	100	101	110	111
Designation	p1	p2	d1	p3	d2	d3	d4
Should you use it in XOR?		✓	✓			✓	✓

When decoding, XORs are checked in an analogous way, the XOR result for p1 is the youngest bit of the syndrome, so if the results are p1=1, p2=1, p3=0, it means that the syndrome is 011 and as it was mentioned before, it means that an error has been detected at

position 3 of the code word. This error is corrected and you can then decode the message by extracting only the data bits from the corrected codeword

In the case of an additional parity bit, the encoding is initially the same, at the end only a new p0 bit is added in the first position, which contains the XOR of all bits. When decoding, if:

- Decimal representation of the syndrome (C) = 0, XOR of all bits (P) = 1; there was an error in the extra bit
- C=0, P=0; no errors found
- C != 0, P = 1; found 1 error at position C
- C != 0, P = 0; 2 bugs found, can't be corrected

Coding:

```
void HammingCode::encodeDataAsync(){
    int n = this->m + this->p, dataPtr{};
    QBitArray dataEncoded(n);
    for(int i = 0; i < n; i++){
        bool isParity = isPowerTwo(i + 1),
             dataBit = data[dataPtr];


        if(!isParity) dataEncoded[i] = data[dataPtr]; //copying non-parity bits


        if(!isParity) dataPtr++;
    }
    int bit = 0;
    for(int i = 1; i <= n; i *= 2){ //calculating parity bits
        int xorVal = 0; //counting number of 1's for each parity bit, xor just signals even/odd count
        for(int j = i + 1; j <= n; j++){
            if(j & (1 << bit)){ //bit manipulation trick
                xorVal ^= dataEncoded[j - 1];
            }
        }
        dataEncoded[i - 1] = xorVal; //keeping even number of 1's in the code
        bit++;
    }
    else data = dataEncoded; //just copy the rest without extending the bit
}
```

Decoding:

```
int HammingCode::correctErrorStandard()
{
    int n = this->m + this->p, C{}, bit{};
    for(int i = 1; i <= n; i *= 2){ //calculate parity bits and add them up with formula: p1 * 1 + p2 * 2 + p3 * 4 + ... = C
        Int sarval = 0;
        xorVal ^= receivedCode[i - 1]; //xor is same as counting 1's
        for(int j = i + 1; j <= n; j++){
            if(j & (1 << bit)){
                xorVal ^= receivedCode[j - 1];
            }
        }
        C += xorVal * i;
        bit++;
    }
    if(C == 0) return -1;
    else return C - 1;
}
```

2.3.2 Reed-Solomon

The implementation of the Reed-Solomon code is in the src/ReedSolomonCode.cpp file, and the header is in include/ReedSolomonCode.hpp

Reed-Solomon coding is much more difficult to understand, so it was created on the basis of a sample implementation in Python and was ported to C++. The implementation used with an explanation of how it works can be found at https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders

The coding uses finite field theory (so-called Galois Bodies - GF), which have been implemented in a general version (with any number of elements), but only 8-element ones are used for visualization purposes, so that they can be easily shown as 1 digit, simulating a block of 3-bit data.

In this body, the arithmetic is as follows:

Table 4.6 Arithmetic in GF(2^3)

	+	000	001	010	011	100	101	110	111
		0	1	2	3	4	5	6	7
000	0	0	1	2	3	4	5	6	7
001	1	1	0	3	2	5	4	7	6
010	2	2	3	0	1	6	7	4	5
011	3	3	2	1	0	7	6	5	4
100	4	4	5	6	7	0	1	2	3
101	5	5	4	7	6	1	0	3	2
110	6	6	7	4	5	2	3	0	1
111	7	7	6	5	4	3	2	1	0

(a) Addition

	×	000	001	010	011	100	101	110	111
		0	1	2	3	4	5	6	7
000	0	0	0	0	0	0	0	0	0
001	1	0	1	2	3	4	5	6	7
010	2	0	2	4	6	3	1	7	5
011	3	0	3	6	5	7	4	1	2
100	4	0	4	3	7	6	2	5	1
101	5	0	5	1	4	2	7	3	6
110	6	0	6	7	1	5	3	2	4
111	7	0	7	5	2	1	6	4	3

(b) Multiplication

The GaloisField class, which is also in the ReedSolomonCode.cpp file, is responsible for these operations.

Operations on polynomials represented by the Poly class (also in ReedSolomonCode.cpp) containing fields n – degree of the polynomial and int* coef – an array with polynomial coefficients, starting from the lowest (free coefficient) are also used and implanted.

Encoding is done on messages with 5 data blocks and 2 control blocks, allowing to correct 1 erroneous block of unknown position (so in GF(8) this can mean an error on 3 bits). First, 2 control blocks are calculated by counting the polynomial division of the selected message by the generating polynomial. The coefficients of the remainder from division are used as control blocks.

Error correction begins with counting the syndromes, which are the value of the received message (in the form of a polynomial) at the appropriate points. If all syndromes are equal to 0, it means that no error has occurred.

In the opposite situation, the polynomial error_locator is calculated, used to locate the error using the Berlekamp-Massey algorithm. It is used in the Chien algorithm to find at which positions transmission errors occurred.

Error locations are used to compute errata_locator, a polynomial that searches for errors, which differs from error_locator in that it takes into account the indicated error positions. Such functionality has not been implemented in this project, but typically, if you know that an error has occurred at a particular position, you can pass such a position to the algorithm, which improves its corrective power.

Errata_locator then allows the calculation of error_evaluator – the polynomial that is used to calculate the magnitude of the error (e.g. whether instead of 2 there should be 3 or maybe 4).

After calculating the error value for each item, the message can be corrected. The decoded word is the first 5 coefficients of the corrected message.

2.4 Tests:

The application contains tests to check whether the introduced changes did not cause errors in the implementation of the coding algorithm. Tests are performed automatically when the application is launched during development and the developer will be informed if any of the tests does not work. The test files are located in the tests directory. To change this directory or add a new one, you need to modify the CMakeLists.txt file in the main directory – currently there is

```
add_subdirectory(tests)
```

And in this folder there is another CMakeLists.txt file:

```
file(GLOB FOLDER_FILES RELATIVE ${CMAKE_CURRENT_SOURCE_DIR} *.cpp)

target_sources(appRedundantCoding PRIVATE

   ${FOLDER_FILES}

)
```

It is enough that the variable FOLDER_FILES points to all files with tests.

The tests themselves use the GTest library.

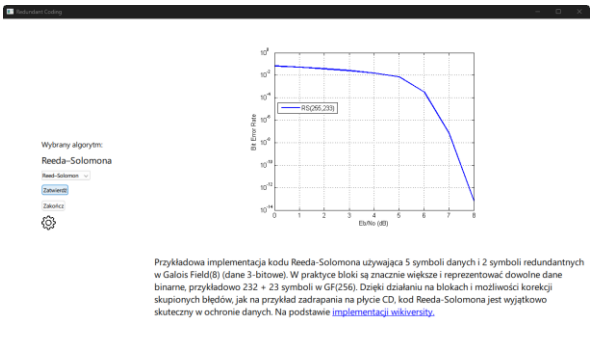
Sample test:

```
////hamming's encoding testing without additional parity bit

TEST(Hamming, hammingEncoding){


    //data, expected encoded data

    QList<QPair<QString, QString>> dataAndResult{{"1110", "0010110"},
{"11010101100111011111010110", "11111010010110011111011111010110"}, {"11100001000",
"001111010001000"},

                                        {"00101010110", "110101001010110"},
{"01111000100000001001010001", "11001111100010000000001001010001"},
{"11100001011010010001110100", "11111101000101100100100001110100"},


{"01100001001000111011010010000110100100111011100100011110 1",
"01001101000100100001110110100100000110100100111011100100011110 1"},
{"11101101110000101010110100000010110110000100010010110110 0",
"00101100110111010001010101101000000010110110000100010010110110 0"},


{"10000111110010110000001001001100011001110001010111111000110111011111010100100
01000000100001101011110110101000000100010010",
"10110001011111010101100000010010001100011001110001010111111000101011101111101
01001001000000100001101011110110101000000100010010"},


{"11010001100011001111110100000101110100000100100000101000100100111001101111011
00010111001001000101000000101101011100111000111000000001011011011110110000100
00010010111101101101110111111011111001010101010001110100110000101000010011011
0111000101110100",
"01111010000110010110011111101001000101110100000100100000101000110010011100111
01111011000101110010010001010000001011010111001110010111000000010110110111101
10000100000100101111011011011101111110111110010101010100011101001100001010000
10011011001110001011101 00"}};


    foreach(auto pair, dataAndResult){


        QString data = pair.first, expectedResult = pair.second, result{};


        QBitArray bits(data.size());


        for(int i = 0; i < data.size(); i++) bits[i] = (data[i] == '1');


        auto hammingCode = QSharedPointer<HammingCode>(new HammingCode());
```

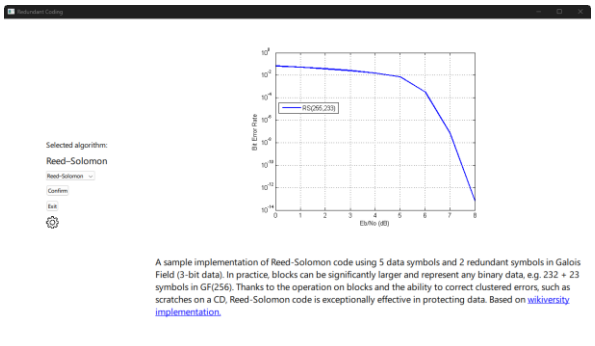
```
        hammingCode.data()->setInitialData(bits, false);
        hammingCode.data()->encodeData(false);
        QBitArray encoded = hammingCode.data()->getData();
        for(int i = 0; i < encoded.size(); i++){
            result.append(encoded.testBit(i) ? '1' : '0');
        }
        ASSERT_STREQ(result.toStdString().c_str(), expectedResult.toStdString().c_str());
    }
}
```

Each test should start with a Test *(arg1, arg2) block {* arguments are the name of the test suite and the name of the specific test. In such a block, you can write regular C++ code and ASSERT methods from GTest. In this case, the test has encoded several possible inputs and their correctly encoded messages and for each of these pairs, it first tries to encode the input using our code (HammingCode class, encodeData method) and then checks if the result is consistent with the correct one (ASSERT_STREQ – compares if 2 strings are identical).

2.5 Translating the app



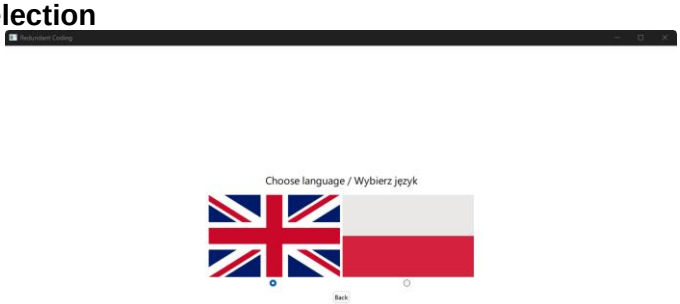
Drawing Main Menu in Polish



Drawing Main Menu in English

It is possible to operate the application in two languages - Polish and English. The source code provides step-by-step instructions on how to add any additional languages if necessary. This option will also be somewhat presented in this chapter, because it will show how language packs work in the program.

2.5.1 Language selection



Drawing Options Screen

In the program, the language can be selected in the "Options" screen, marked in the main menu with a gear icon. The user can choose between English, marked with the flag of the United Kingdom, and Polish, marked with the flag of Polish. Under the flags there are appropriate radio type buttons. The language changes when the corresponding button under the flag is pressed. It is immediately saved to the configuration file. Thanks to this treatment, the application works in the appropriate language even after restarting the application, so there is no need to change it every time. The program's configuration file containing language information is stored in Microsoft Windows in the key registry at the path: *\HKEY_CURRENT_USER\Software\Redundant Coding\App*. The variable that stores information about the set language of the program is called *Language*. When its value is 0, the program runs in English, and when the *Language* variable is set to 1, the application is running in English.

2.5.2 Language Packs

```
selectedAlgorithmText: "Wybrany algorytm: "
confirmButton: "Zatwierdź"
confirmButton.ToolTip: "Potwierdź wybór algorytmu"
exitButton: "Zakończ"
messageDialog.title: "Zakończ"
messageDialog: "Czy chcesz zakończyć działanie programu?"

backButton.ToolTip: "Wróć do menu głównego"
backButton: "Wróć"

selectedAlgorithmText: "Startowe dane"
delayText: "Odstępy animacji: "
animationDelay.ToolTip: "Zmień czas"
additionalBit.text: "Dodatkowy bit parzystości"
vizualizeButton.ToolTip: "Przejdź do wizualizacji algorytmu"
vizualizeButton: "Wizualizuj"
errorMsg: "Wpisz ciąg składający się z 0 i 1!"
backButton.ToolTip: "Wróć do menu głównego"
backButton: "Wróć"

controlMatrixText.text: "Macierz kontroli parzystości"
generationMatrixText.text: "Macierz generacyjna"
nextStepButton.text: "Następny krok"

stageTextExt.text: "Zmień od 0 do "
stageTextExt.text: " bitów klikając na nie"
animationDelayText.text "Odstępy animacji: "
animationDelay.ToolTip.text: "Zmień czas"
visualiseButton.text: "Zacznij"
mainMenuButton.text: "Menu Główne"
nextStepButton.text: "Następny krok"
encoding.text: "Odkodowywanie..."
encodingEnd.text: "Poprawianie błędów"
correctingErrors.text: "Znajdowanie błędu/ów..."
correctErrorExtended1.text: "Błąd występuje w rozszerzonym bicie parzystości (0)"
correctErrorExtended2.text: "Nie znaleziono błędów!"
correctErrorExtended3.text: "Błąd znajduje się na pozycji: "
correctErrorExtended4.text: "Występuje podwójny błąd, który nie może zostać wyliczony, gdzie występuje"
parityBit.text: "Bit parzystości: "
additionalParityBit: "Dodatkowy bit parzystości ="

receivedMessageText.text: "Otrzymana wiadomość"
syndromeText.text: "Syndrom błędu"
vectorText.text: "Wektor błędów"
xorText.text: "Uzyskana wiadomość; Otrzymana wiadomość XOR Wektor błędów"
decodedText: "Odkodowana wiadomość"
mainMenuButton.text: "Menu główne"
```

```
selectedAlgorithmText: "Selected algorithm: "
confirmButton: "Confirm"
confirmButton.ToolTip: "Confirm your choice of algorithm"
exitButton: "Exit"
messageDialog.title: "Close"
messageDialog: "Do you want to close the program?"

backButton.ToolTip: "Go back to Main Menu"
backButton: "Back"

selectedAlgorithmText: "Starting data"
delayText: "Animation intervals: "
animationDelay.ToolTip: "Change the time"
additionalBit.text: "Additional parity bit"
vizualizeButton.ToolTip: "Go to the visualization of the algorithm"
vizualizeButton: "Visualize"
errorMsg: "Enter a string consisting of 0 and 1!"
backButton.ToolTip: "Return to Main Menu"
backButton: "Back"

controlMatrixText.text: "Parity check matrix"
generationMatrixText.text: "Generation matrix"
nextStepButton.text: "Next step"

stageTextExt.text: "Change from 0 to "
stageTextExt.text: " bits clicking on them"
animationDelayText.text "Animation intervals: "
animationDelay.ToolTip.text: "Change the time"
visualiseButton.text: "Begin"
mainMenuButton.text: "Main Menu"
nextStepButton.text: "Next step"
encoding.text: "Encoding..."
encodingEnd.text: "Correcting errors"
correctingErrors.text: "Finding error(s)..."
correctErrorExtended1.text: "Error is in extended parity bit (0)"
correctErrorExtended2.text: "There is no error!"
correctErrorExtended3.text: "Single error occured at the position: "
correctErrorExtended4.text: "There is a double error, but it can't be calculated where"
parityBit.text: "Parity bit: "
additionalParityBit: "Additional parity bit ="

receivedMessageText.text: "Received message"
syndromeText.text: "Error syndrome"
vectorText.text: "Error vector"
xorText.text: "Message obtained; Message received XOR Error vector"
decodedText: "Decoded message"
mainMenuButton.text: "Main Menu"
```

Figure Visible Language Packs - Polish and English

The program uses language packs, i.e. files containing the translation of each element of the program. These files are located in the */assets/lang/* folder. Each of the files is in *.lang* format. In this file, the content of each page is translated one by one. It is important to write them in the right order and in the same format, as shown in the figure above (Figure). Translation format: *variable: "translation".* This format resembles a dictionary, which is a data type in popular programming languages that allows you to assign a meaning to a text. The translation cannot contain quotation marks (*"*), which will be explained in the next chapter, which provides a detailed discussion of how the translation of elements in the application works.

2.5.3 How translation works in the app

In order for the translation to be implemented in the program, a separate class called Settings has been created, which allows you to store information about the program settings. It is stored in the *Settings.cpp* and *Settings.hpp* files. This class in its final form allows you to read *getLanguage()* and write setLanguage(*int language)* the value of the set language in the configuration file and read from the language packs the appropriate translations of the elements visible on individual readFile(*int page) screens*.
The key of the three functions just mentioned in the previous paragraph is *readFile(int page).* It works on the principle that it is called each time before moving to the next screen. In the instruction file *Readme.md*, included with the language packs, the screen numbers are presented, which must be provided as an int page argument to get their translation. In the .qml files of individual screens, appropriate functions are prepared, which insert the translation into the elements of the indicated screen. These functions are automatically called when you use *the readFile(page)* function from the *Settings class on the previous screen*. It is important not to confuse the number of the screen being called, otherwise it will lead to indeterminate behavior of the program, in the best case there will only be a mistranslation of the elements, but in the worst case the program will stop working.
If you delve into the details of how the translation works from the very beginning, it looks like this:

- From the .qml file of screen X, the *Settings::readFile(y) function is called*, where y is the number specifying screen Y
- The *readFile* function in the *Settings.cpp* calls *the getLanguage()* function from the *Settings class*
- The getLanguage() *function* retrieves information from the configuration file (in the case of Microsoft Windows, the key registry) about what language is set
- Based on the language value obtained from *the getLanguage()* function , *readFile(y)* opens the appropriate language pack.
- When the language pack is opened correctly, the function reads the text line by line
- The y value given as an argument to *readFile(y)* tells it how many newlines the function needs to read before it starts writing text from the file
- Once the function has encountered the appropriate number of newline characters, it starts to take the translation from the read line of text by extracting only the text between the quotation marks (hence the translated text cannot contain quotation marks)
- The extracted translation is added to the output variable of type QString, followed by a newline "\n"
- If the function encounters an empty character or at the end of a file, it finishes reading the text from the file and returns a variable that stores the translation.
- The readFile(y) *function* when returning a non-empty variable emits a signal that will allow the translation to be set on the correct page when it is loaded
- In the .qml file of a given page, there is a function written that receives an output variable from the emitted signal, which stores the translation in itself
- This function separates individual translations of elements that are separated by a newline and writes them to the QString table
- On the basis of the established order in this array, translations are assigned for each of the elements

3 Attachments

{all documents that cannot be easily integrated into the text should be attached in separate files, and their list should be presented in the form of a table, for example:}