

ACCA4017_1700994_Bartosz_G alantowicz

by Bartosz Galantowicz

FILE	2275_BARTOSZ_GALANTOWICZ_ACCA4017_1700994_BARTOSZ_GALANTOWICZ_178138_1218048861.DOCX (1.37M)		
TIME SUBMITTED	05-APR-2019 02:56PM (UTC+0100)	WORD COUNT	4693
SUBMISSION ID	104208586	CHARACTER COUNT	26461

Programming in C++

Bartosz Galantowicz

1700994

Software Development – ACCA4017

Mike Dacey / Tim Bashford

Table of Contents

1. Abstract.....	3
2. Introduction	3
3. Design.....	3
4. Exercises.....	7
4.1. Exercise 1	7
4.1.1 Constraints and Requirements	7
4.2. Exercise 2	23
4.2.1. Constraints and Requirements	23
5. Bibliography	39

1. Abstract

The report is an evidence of the knowledge in the field of software development that has been obtained for first-year academic studies at the University of Wales Trinity Saint David. Furthermore, the main purpose of the report is to show and implement programming skills for given examples. Therefore, the creation of a working application is not the aim. The objective is to find out how does the code would work before will be implemented in the code. To solve the problem in a logical order and then use a programming language for developing a program in an organised way. Those proper programming practice methods are shown referring to three exercises from the assignment.

2. Introduction

C++ as a modern high-level programming language was chosen to accomplish the assigned tasks. Furthermore, the IDE is Visual Studio 2017, and the software has a form of Windows Desktop Wizard application. The tasks are based on building three different games referring to programming skills and knowledge gained during the university studies. Each game is an examination of various difficulties levels and methods on how to deal with them. However solved exercises have been tested if they work correctly, and the results are included in the report. Moreover, in the assignment, each step of program development is described and analysed. To perform required tasks and to make the work with code easy the `iostream`, `string`, `sstream` and `vector` libraries are used.

3. Design

The application design idea is to combine all exercises in one menu, which is displayed to a user. Accordingly, one from three exercises can be chosen by the user and then tested. Moreover, each exercise may be repeated many times until it is decided to stop. Namely after finished one game, the menu is displayed again, and the program is waiting for the user decision.

```
int main()
{
    int userChoice{ 0 };

    do
    {
        DisplayMenu();
        userChoice = ReadIntValue(1, 3, "Enter the exercise number: ");
        ChoseExercise(userChoice);

    } while (userChoice != 3);

    return 0;
}
```

```
void DisplayMenu()
{
    cout << "*****" << endl;
    cout << endl;

    cout << "Exercise menu: " << endl;
    cout << "1. EXERCISE 1" << endl;
    cout << "2. EXERCISE 2" << endl;
    cout << "3. Exit" << endl;
}
```

```
// the function reads the user input and checks that is a number
double ReadNumber(string prompt)
{
    double value{};
    string input{};
    stringstream stream{};

    cout << prompt;
    getline(cin, input);
    stream.str(input); // Empty the contents
    stream.clear(); // Empty the bits flags

    while ((stream >> value && stream.eof()) == false)
    {
        cout << prompt;
        stream.clear();
        getline(cin, input);
        stream.str(input);
    }

    stream >> value; // convert the string into a integer value

    return value;
}
```

```
// reading integer values
// passing min and max value allows for setting the function
int ReadIntValue(int minValue, int maxValue, string prompt)
{
    double userValue{ 0 };

    // avoiding confusion when a user enters decimal values
    do
    {
        userValue = ReadNumber(prompt);

        // checking that the chosen value is not a decimal number
    } while ((userValue - int(userValue) > 0) || (userValue < minValue || userValue > maxValue));

    // casting double into int
    int userIntValue = static_cast<int>(userValue);

    return userIntValue;
}
```

```
void ChoseExercise(int userChoice)
{
    switch (userChoice)
    {
        // Exercise 1
        case 1:
        {
            vector<double> setOfSevenValues;

            cout << "Welcome in EXERCISE 1" << endl;

            cout << "Player 1, please enter seven values without repetition:" << endl;

            setOfSevenValues = StoringPlayerOneValues();

            cout << "Player 2 is your turn! The first value is " << setOfSevenValues.at(0) << endl;

            CheckingGuess(setOfSevenValues);
            break;
        }
    }
}
```

```
// Exercise 2
case 2:
{
    const int numberOfTreasures{ 3 };

    cout << "Welcome in EXERCISE 2" << endl;

    cout << "Firstly enter a row (X) and then enter a column (Y)." << endl;

    cout << "Player 1 place treasures now:" << endl;

    // locating the treasures
    vector<int> treasuresCoordinates;

    for (int i{ 0 }; i < numberOfTreasures; i++)
    {
        treasuresCoordinates = StoreCoordinates(treasuresCoordinates);
        cout << "Treasure " << i + 1 << " has been placed" << endl;
    }

    // initialising the grid
    string grid[rows][columns]
    {
        {"-","-","-","-","-","-","-","-","-","-"}, //0 row
        {"-","-","-","-","-","-","-","-","-","-"}, //1st row
        {"-","-","-","-","-","-","-","-","-","-"}, //2nd row
        {"-","-","-","-","-","-","-","-","-","-"}, //3rd row
        {"-","-","-","-","-","-","-","-","-","-"}, //4th row
        {"-","-","-","-","-","-","-","-","-","-"}, //5th row
        {"-","-","-","-","-","-","-","-","-","-"}, //6th row
        {"-","-","-","-","-","-","-","-","-","-"} //7th row
    };

    DisplayGrid(grid);

    StartGameCore(numberOfTreasures, treasuresCoordinates, grid);

    break;
}

case 3:
    cout << endl;
    cout << "Exiting the application...";
    break;
}
```

Fig.3.1.1 Presented functions are vital for the program organisation. Namely, a user can choose a specific operation or exit the program. What is more, it is informed about valid options by displaying the menu and its content in the application box. The management of the program is based on a switch statement, which is a coded template for the menu. Furthermore, a few kinds of taking input mechanics was coded because of the requirements of specific tasks. For example ReadNumber() function allows for obtaining only numerical values, whereas ReadIntValue() function takes only integers.

4. Exercises

4.1. Exercise 1

The first exercise is a game for two players, where seven numbers are provided by one player and then the first value from the set is displayed to the second player. Furthermore, the second player is obligated to guess that the next value from the set, is lower or higher than the previous number, and so on. The game finishes when all guesses are correct, or a mistake is committed by the guessing player.

4.1.1 Constraints and Requirements

- a) Seven values provided by player one cannot be repeated.
- b) The values cannot be a letter or a special sign.
- c) The letters “h” and “H” stand for “higher”, or “l” and “L”, stand for “lower”, can be used only by player two as an input.
- d) If a mistake will be made by player two, then the game is ended. The winner is the player one.
- e) If all values are revealed by player two, then the game is ended. The winner is player two.

1.1. Stages of development

- a) Arranging the event system for the game (display order and what is displayed).
- b) Creating an input system, taking the data from the users.
- c) Taking values from player one.
- d) Displaying first value from the given set.
- e) Taking input from the player two.
- f) Comparing that the next value is lower or higher than the previous one.
- g) Displaying the result of the operation.

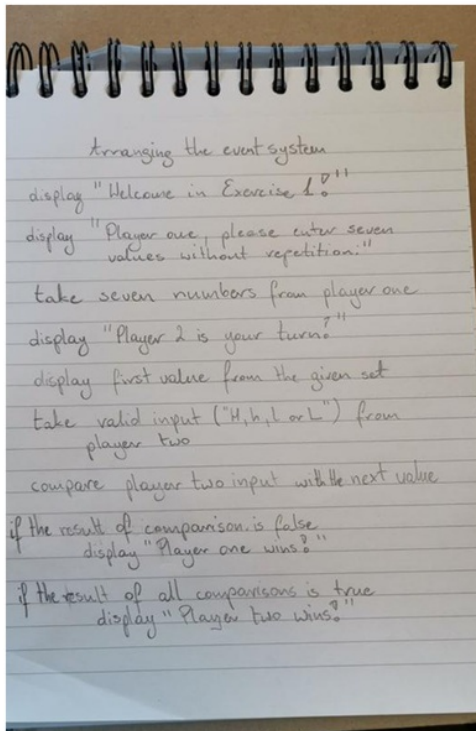


Fig.4.2.1 Arranging the event system for the game has been written down in order of work.

```
void ChoseExercise(int userChoice)
{
    switch (userChoice)
    {
        // Exercise 1
        case 1:
        {
            vector<double> setOfSevenValues;

            cout << "Welcome in EXERCISE 1" << endl;

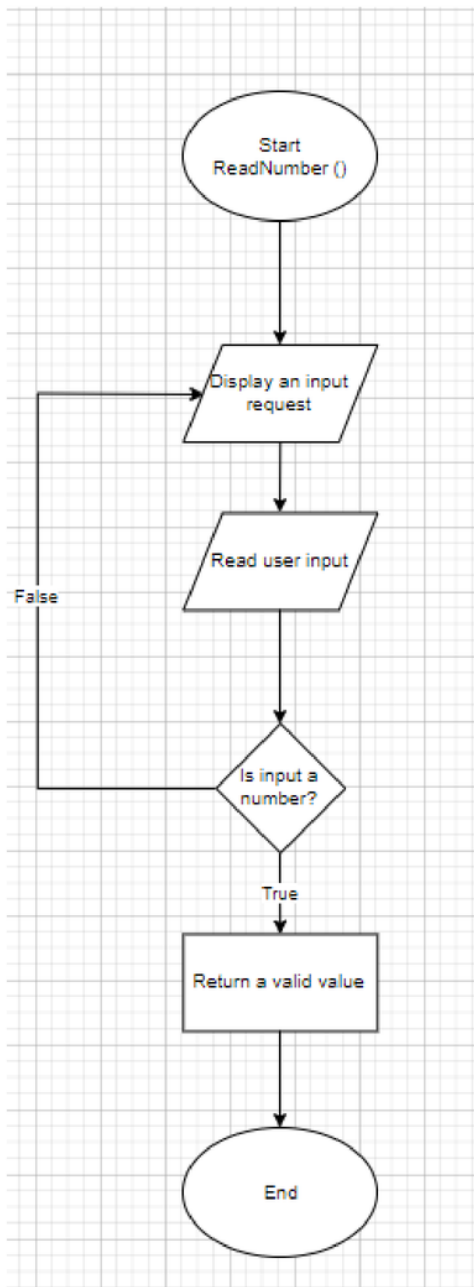
            cout << "Player 1, please enter seven values without repetition:" << endl;

            setOfSevenValues = storingPlayerOneValues();

            cout << "Player 2 is your turn! The first value is " << setOfSevenValues.at(0) << endl;

            CheckingGuess(setOfSevenValues);
            break;
        }
    }
}
```

Fig.4.2.2 Exercise 1 is allocated in the switch statement for the first case.



```

// the function reads the user input and checks that is a number
double ReadNumber(string prompt)
{
    double value{};
    string input{};
    stringstream stream{};

    cout << prompt;
    getline(cin, input);
    stream.str(input); // Empty the contents
    stream.clear(); // Empty the bits flags

    while ((stream >> value && stream.eof()) == false)
    {
        cout << prompt;
        stream.clear();
        getline(cin, input);
        stream.str(input);
    }

    stream >> value; // convert the string into a integer value

    return value;
}

```

Fig.4.2.3 This logic system is used as a principle for reading input from a user. However, it is implemented in function ReadNumber(), which the primary purpose is to take valid values (numbers) and passing them forward. Furthermore, this function is used in all exercises as universally.

```

Player 1, please enter seven values without repetition:
Enter number 1>6
Enter number 2>-2
Enter number 3>96
Enter number 4>-96
Enter number 5>-1
Enter number 6>1
Enter number 7>8

```

```

Player 1, please enter seven values without repetition:
Enter number 1>0.001
Enter number 2>-0.02
Enter number 3>12.96
Enter number 4>0.1
Enter number 5>-0.1
Enter number 6>0.99
Enter number 7>-0.999
Player 2 is your turn! The first value is 0.001

```

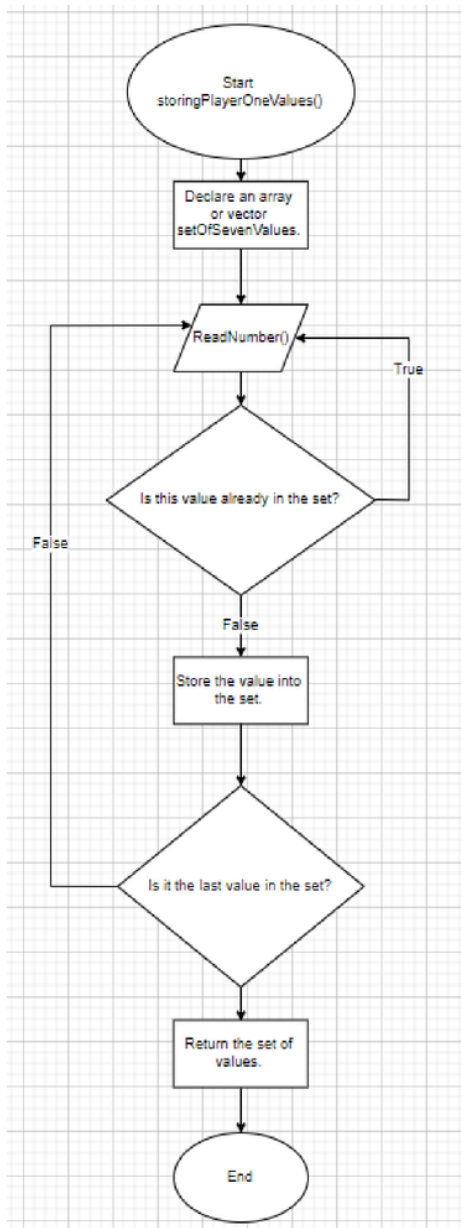
```

Player 1, please enter seven values without repetition:
Enter number 1>g
h
w
A
&
$
1
Enter number 2>

```

Attempts	Test Values	Description	Expected Result	Actual Result	Pass / Fail
1	integer values – 6, -2, 96, -96, -1, 1, 8	Reading a number from the user.	6, -2, 96, -96, -1, 1, 8	6, -2, 96, -96, -1, 1, 8	Pass
2	double values – 0.001, -0.02, 12.96, 0.1, -0.1, 0.99, -0.999	Reading a number from the user.	0.001, -0.02, 12.96, 0.1, -0.1, 0.99, - 0.999	0.001, -0.02, 12.96, 0.1, -0.1, 0.99, -0.999	Pass
3	alphabetic and special signs inputs – h, w, A, &, \$	Reading a number from the user.	Ignoring alphabetical inputs and special signs until a numerical value is entered.	Ignoring alphabetical inputs and special signs until a numerical value is entered.	Pass

Fig.4.2.4 The examples of using the ReadNumber() function in tests. However, the function is used as a supporting one. Hence it often occurs in other functions.



```

vector<double> StoringPlayerOneValues()
{
    vector<double> setOfSevenValues;
    int counter{ 0 };
    bool isRepeated{ false };

    for (int i{ 0 }; i < maxNumberOfValues; i++)
    {
        counter = 0; // resetting the counter

        do
        {
            cout << "Enter number " << i + 1 << ">";
            setOfSevenValues.push_back(ReadNumber("")); // getting value from the user

            for (unsigned int j{ 0 }; j < setOfSevenValues.size(); j++)
            {
                if (setOfSevenValues.at(i) == setOfSevenValues.at(j))
                {
                    counter++; // updating the counter when given value have been already entered
                }

                if (counter > 1) // is more than one examples of given values
                {
                    isRepeated = true;
                    setOfSevenValues.pop_back(); // discarding last implement value
                    cout << "Sorry but this values is already in the set. Try again. ";
                    counter = 0; // resetting the counter
                    break;
                }
                else
                {
                    isRepeated = false; // if in the set are not repetitions break do...while loop condition
                }
            }

        } while (isRepeated == true); // keep going until the input is valid

        counter = 0;
    }
    return setOfSevenValues;
}

```

Fig.4.2.5 The figures are showing the process of taking input from the player one by function StoringPlayerOneValues (). Moreover, the operation is repeated until the given value is valid (is not repeated in the set of values) and when all seven values are stored in the array or vector. The assumption is that the numbers cannot be repeated because, if next value is the same as previous player two is not able to say that those values are equal as in the game only two options (is higher or lower) can be chosen.

```
Enter the exercise number: 1
Welcome in EXERCISE 1
Player 1, please enter seven values without repetition:
Enter number 1>6
Enter number 2>-2
Enter number 3>96
Enter number 4>-96
Enter number 5>-1
Enter number 6>1
Enter number 7>8
Player 2 is your turn! The first value is 6
Higher (H) or Lower (L)>
```

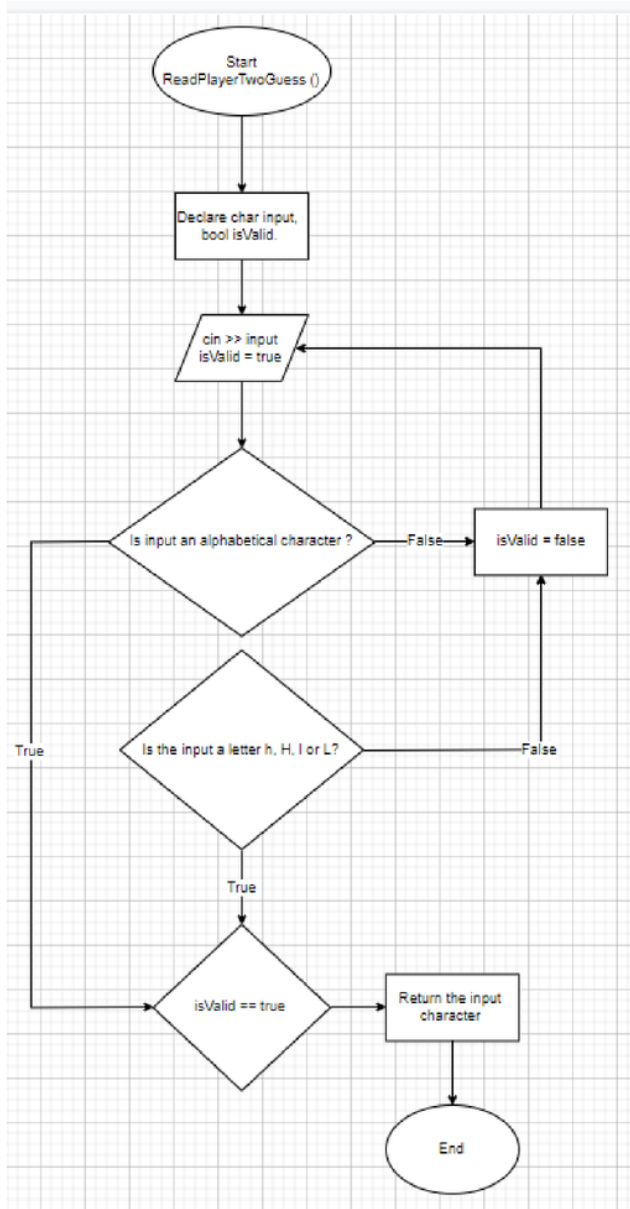
```
Enter the exercise number: 1
Welcome in EXERCISE 1
Player 1, please enter seven values without repetition:
Enter number 1>0.001
Enter number 2>-0.02
Enter number 3>12.96
Enter number 4>0.1
Enter number 5>-0.1
Enter number 6>0.99
Enter number 7>-0.999
Player 2 is your turn! The first value is 0.001
Higher (H) or Lower (L)>
```

```
Enter the exercise number: 1
Welcome in EXERCISE 1
Player 1, please enter seven values without repetition:
Enter number 1>4
Enter number 2>3
Enter number 3>2
Enter number 4>1
Enter number 5>2
Sorry but this values is already in the set. Try again. Enter number 5>3
Sorry but this values is already in the set. Try again. Enter number 5>4
Sorry but this values is already in the set. Try again. Enter number 5>5
Enter number 6>6
Enter number 7>3
Sorry but this values is already in the set. Try again. Enter number 7>5
Sorry but this values is already in the set. Try again. Enter number 7>7
Player 2 is your turn! The first value is 4
Higher (H) or Lower (L)>
```

Attempts	Test Values	Description	Expected Result	Actual Result	Pass / Fail
1	integer values – 6, -2, 96, -96, -1, 1, 8	Store seven values of the first player.	6, -2, 96, -96, -1, 1, 8	6, -2, 96, -96, -1, 1, 8	Pass

2	double values – 0.001, -0.02, 12.96, 0.1, -0.1, 0.99, -0.999	Store seven values of the first player.	0.001, -0.02, 12.96, 0.1, -0.1, 0.99, -0.999	0.001, -0.02, 12.96, 0.1, -0.1, 0.99, -0.999	Pass
3	repeated values – 4, 3, 2, 1, 2, 3, 4	Store seven values of the first player.	Asking the user to enter a non- repeating value.	Asking the user to enter a non- repeating value.	Pass

Fig.4.2.6 The examples of StoringPlayerOneValues() function in tests. Firstly, a new number is taken from the player one and then it is checked that was not repeated. If it was, the user is asked to enter a different value. Since all seven values are saved, the vector containing them can be returned.



```

char ReadPlayerTwoGuess()
{
    char input;
    bool isValid{ true };

    do
    {
        cout << "Higher (H) or Lower (L)>";
        cin >> input; // get a line of input
        isValid = true; // assume it is valid
        cin.clear(); // clear the error flags
        cin.ignore(1000, '\n'); // discard no needed characters from the input stream until a new line

        if (!isalpha(input)) // is it an alphabetical character?
        {
            isValid = false; // if not, mark it as invalid

            // print an error to the user
            cout << "Invalid input. Please input only alphabetical characters." << endl;
        }
        else if (input != 'h' && input != 'l' && input != 'H' && input != 'L') // is it h, H, l, or L?
        {
            isValid = false; // if not, mark it as invalid
            cout << "Invalid input. Please input ""h"" or ""H"" for higher and ""l"" or ""L"" for lower." << endl;
        }
    } while (!isValid); // keep going until the input is valid

    return input;
}

```

Fig.4.2.7 The flowcharts are presenting a logic way to receive a valid input from player two. Thereby the function ReadPlayerTwoGuess () is responsible for checking that entered values are letters: “l”, “L” (stand for “lower”), “h” or “H” (stand for “higher”) and they are not numbers. If the input is valid then is passed further.

```

Enter the exercise number: 1
Welcome in EXERCISE 1
Player 1, please enter seven values without repetition:
Enter number 1>1
Enter number 2>2
Enter number 3>3
Enter number 4>4
Enter number 5>5
Enter number 6>6
Enter number 7>7
Player 2 is your turn! The first value is 1
Higher (H) or Lower (L)>72
Invalid input. Please input only alphabetical characters.
Higher (H) or Lower (L)>104
Invalid input. Please input only alphabetical characters.
Higher (H) or Lower (L)>76
Invalid input. Please input only alphabetical characters.
Higher (H) or Lower (L)>108
Invalid input. Please input only alphabetical characters.
Higher (H) or Lower (L)>

```

```
Enter the exercise number: 1
Welcome in EXERCISE 1
Player 1, please enter seven values without repetition:
Enter number 1>1
Enter number 2>2
Enter number 3>3
Enter number 4>-2
Enter number 5>-1
Enter number 6>6
Enter number 7>7
Player 2 is your turn! The first value is 1
Higher (H) or Lower (L)>h
Correct, next value is 2
Higher (H) or Lower (L)>H
Correct, next value is 3
Higher (H) or Lower (L)>l
Correct, next value is -2
Higher (H) or Lower (L)>L

Incorrect, sorry player 1 won!
*****
```

```
Exercise menu:
1. EXERCISE 1
2. EXERCISE 2
3. EXERCISE 3
4. Exit
Enter the exercise number: 1
Welcome in EXERCISE 1
Player 1, please enter seven values without repetition:
Enter number 1>1
Enter number 2>2
Enter number 3>3
Enter number 4>4
Enter number 5>5
Enter number 6>6
Enter number 7>7
Player 2 is your turn! The first value is 1
Higher (H) or Lower (L)>q
Invalid input. Please input h or H for higher and l or L for lower.
Higher (H) or Lower (L)>w
Invalid input. Please input h or H for higher and l or L for lower.
Higher (H) or Lower (L)>e
Invalid input. Please input h or H for higher and l or L for lower.
Higher (H) or Lower (L)>r
Invalid input. Please input h or H for higher and l or L for lower.
Higher (H) or Lower (L)>H
Correct, next value is 2
Higher (H) or Lower (L)>
```

Attempts	Test Values	Description	Expected Result	Actual Result	Pass / Fail
1	Numbers: 72 – ASCII Code for “H” 104 – ASCII Code for “h” 76– ASCII Code for “L” 108 – ASCII Code for “l”	Reading a valid input of the second player.	Display “Invalid input. Please input only alphabetical characters.” – re-enter the input.	Display “Invalid input. Please input only alphabetical characters.” – re-enter the input.	Pass
2	Letters “h” and “H” or “l” and “L”.	Reading a valid input of the second player.	Go to the next step of the program.	Go to the next step of the program.	Pass
3	Other letters and special signs.	Reading a valid input of the second player.	Display “Invalid input. Please input h or H for higher and l or L for lower.” – re-enter the input.	Display “Invalid input. Please input h or H for higher and l or L for lower.” – re-enter the input.	Pass

Fig.4.2.8 The examples of ReadPlayerTwoGuess() function in tests. Thanks to the function the second player is able to enter only a valid character. If the input is valid “do...while” loop is skipped, and a letter is returned. In the case when the input is a number or invalid character, it is required to enter “h”, “H”, “l” or “L” again.

```

+ ... X
This function is a game core that implements a passed set of values and uses the input of the second
player to evaluate the result of the game.

function CheckingGuess(setOfSevenValues)
{
    Printing first value from the set.
    Taking the input from player two by using the function ReadPlayerTwoGuess().
    For (Count until the game is over or all values are guessed)
    {
        if (the next element of the set is higher, but the given input indicates that is lower)
        {Print "Player one won!" and finish the game.}
        else if (the next element of the set is lower, but the given input indicates that is higher)
        {Print "Player one won!" and finish the game.}
        else
        {
            Print "Correct, next value is ", display next value from the set and increase the counter.

            If (all values are guessed and counted by a counter)
            {Print "Congratulations! Player two won!" and finish the game}
        }
    }
}

```

```

void CheckingGuess(vector<double> setOfSevenValues)
{
    bool isGuessed = true;
    int counter(0);
    char playerTwoInput[];

    while(counter < maxNumberOfValues && isGuessed == true)
    {
        playerTwoInput = ReadPlayerTwoGuess();

        if ((setOfSevenValues.at(counter + 1) > setOfSevenValues.at(counter)) && (playerTwoInput == 'l' || playerTwoInput == 'L'))
        {
            // if the value is different than user input then player 2 loose
            cout << endl;
            cout << "Incorrect, sorry player 1 won!" << endl;
            isGuessed = false;
        }
        else if ((setOfSevenValues.at(counter + 1) < setOfSevenValues.at(counter)) && (playerTwoInput == 'h' || playerTwoInput == 'H'))
        {
            // if the value is different than user input then player 2 loose
            cout << endl;
            cout << "Incorrect, sorry player 1 won!" << endl;
            isGuessed = false;
        }
        else
        {
            // when user input is correct go to next value
            cout << "Correct, next value is " << setOfSevenValues.at(counter + 1) << endl;
            counter++;

            // player 2 wins if all values are guessed
            if (isGuessed == true && counter == maxNumberOfValues - 1)
            {
                cout << endl;
                cout << "Congratulations! Player 2 won!" << endl;
                break;
            }
        }
    }
}

```

Fig.4.2.9 The picture is showing the pseudo code for the core game function and the function itself. From the written note in this way is better to imagine the logic and mechanics of the

game. Namely, the set of values is passed to this function, and then player two can guess next value from the set, basing only on the first displayed number. Summing up the game is ended when a one guess is wrong, or all values are revealed by the second player.

```
Enter the exercise number: 1
Welcome in EXERCISE 1
Player 1, please enter seven values without repetition:
Enter number 1>7
Enter number 2>6
Enter number 3>5
Enter number 4>4
Enter number 5>3
Enter number 6>2
Enter number 7>1
Player 2 is your turn! The first value is 7
Higher (H) or Lower (L)>1
Correct, next value is 6
Higher (H) or Lower (L)>L
Correct, next value is 5
Higher (H) or Lower (L)>H
Incorrect, sorry player 1 won!
*****
```

```
Player 1, please enter seven values without repetition:
Enter number 1>1
Enter number 2>2
Enter number 3>3
Enter number 4>4
Enter number 5>5
Enter number 6>6
Enter number 7>7
Player 2 is your turn! The first value is 1
Higher (H) or Lower (L)>h
Correct, next value is 2
Higher (H) or Lower (L)>H
Correct, next value is 3
Higher (H) or Lower (L)>L
Incorrect, sorry player 1 won!
*****
```

```

Player 1, please enter seven values without repetition:
Enter number 1>3
Enter number 2>2
Enter number 3>1
Enter number 4>4
Enter number 5>0
Enter number 6>6
Enter number 7>8
Player 2 is your turn! The first value is 3
Higher (H) or Lower (L)>L
Correct, next value is 2
Higher (H) or Lower (L)>L
Correct, next value is 1
Higher (H) or Lower (L)>H
Correct, next value is 4
Higher (H) or Lower (L)>L
Correct, next value is 0
Higher (H) or Lower (L)>h
Correct, next value is 6
Higher (H) or Lower (L)>H
Correct, next value is 8
Congratulations! Player 2 won!
*****

```

Attempts	Test Values	Description	Expected Result	Actual Result	Pass / Fail
1	Player two input: "h" or "H" when the guessing value is lower.	Checking correctness of the game mechanics.	Display "Incorrect, sorry player 1 won!" – end the game.	Display "Incorrect, sorry player 1 won!" – end the game.	Pass
2	Player two input: "l" or "L" when the guessing value is higher.	Checking correctness of game mechanics.	Display "Incorrect, sorry player 1 won!" – end the game.	Display "Incorrect, sorry player 1 won!" – end the game.	Pass
3	All guessing inputs are correct.	Checking correctness of the game mechanics.	Display "Congratulations! Player 2 won!" – end the game.	Display "Invalid input. Please input h or H for higher and l or L for lower." – re-enter the input.	Pass

Fig.4.2.10 The examples of CheckingGuess() function in tests. Furthermore, the tested function is a fundamental part of the program because the most important mechanics which making a game, are implemented at this point. Thus, a guess of the second player is compared to real values entering by player one, and that is a base for this game. Therefore, if player two is wrong,

then player one is a winner, but if all numbers are revealed, it is meant that the first player is defeated.

4.2. Exercise 2

Next, the form of the second exercise is another competition game. It is all about hiding treasures by one player and finding them by the second player. Moreover, hidden treasures must be allocated on the provided grid and then the second player is obligated to discover them by using simple coordinates, having only three chances.

4.2.1. Constraints and Requirements

- a) The grid dimension is 8x8 (8 rows and 8 columns), hence 64 elements can be placed there.
- b) The entered coordinates have to be in the range between 0 and 7 for a specific row and column.
- c) Each coordinate for a guess and placing a treasure must be unique to avoid cases when all treasures are hidden at this same position, or live points are lost because entering already checked localisation.
- d) There are only three treasures to hide by the first player.
- e) There are only three lives for the player two.
- f) The life points are lost only when a guess of the second player is not right.
- g) The game is ended when life points of the second player are depleted, or all treasures are revealed.

1.2. Stages of development

- a) Arranging the event system of the game (display order and what is displayed)
- b) Declaring a grid and creating a grid displaying function.
- c) Coding a coordinate reading function.
- d) Implementing coordinates taken from users and storing them.
- e) Introducing a checking coordinates system. Comparing inputs of the player two against the coordinates of treasures.
- f) Organising a core game function, which is responsible for counting lives, treasures and controlling other implemented functions required for finishing the gameplay. Displaying the result of the game.

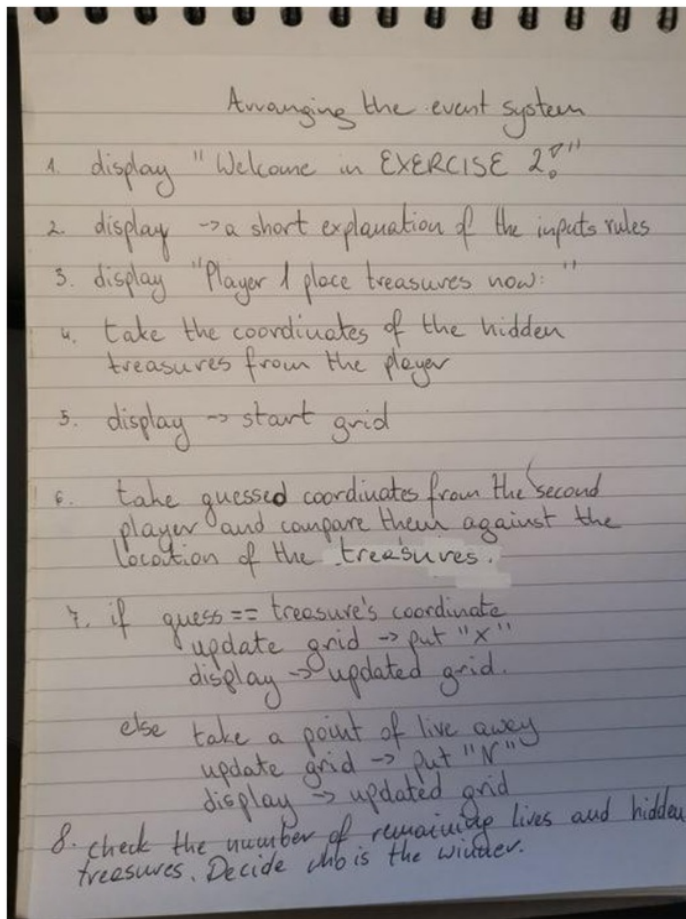


Fig.5.2.1 Arranging the event system for the game has been written down in order to performing.

```
// Exercise 2
case 2:
{
    const int numberOfTreasures{ 3 };

    cout << "Welcome in EXERCISE 2" << endl;

    cout << "Firstly enter a row (X) and then enter a column (Y)." << endl;

    cout << "Player 1 place treasures now:" << endl;

    // locating the treasures
    vector<int> treasuresCoordinates;

    for (int i{ 0 }; i < numberOfTreasures; i++)
    {
        treasuresCoordinates = StoreCoordinates(treasuresCoordinates);
        cout << "Treasure " << i + 1 << " has been placed" << endl;
    }

    // initialising the grid
    string grid[rows][columns]
    {
        {"-","-","-","-","-","-","-","-"}, //0 row
        {"-","-","-","-","-","-","-","-"}, //1st row
        {"-","-","-","-","-","-","-","-"}, //2nd row
        {"-","-","-","-","-","-","-","-"}, //3rd row
        {"-","-","-","-","-","-","-","-"}, //4th row
        {"-","-","-","-","-","-","-","-"}, //5th row
        {"-","-","-","-","-","-","-","-"}, //6th row
        {"-","-","-","-","-","-","-","-"}, //7th row
    };

    DisplayGrid(grid);

    startGameCore(numberOfTreasures, treasuresCoordinates, grid);

    break;
}
```

Fig.5.2.2 Exercise 2 is allocated in the switch statement for the second case.

```
// Display the grid to users
void DisplayGrid(string arrayGrid[rows][columns])
{
    cout << " ";

    for (int i{ 0 }; i < columns; i++)
    {
        cout << " " << i;
    }

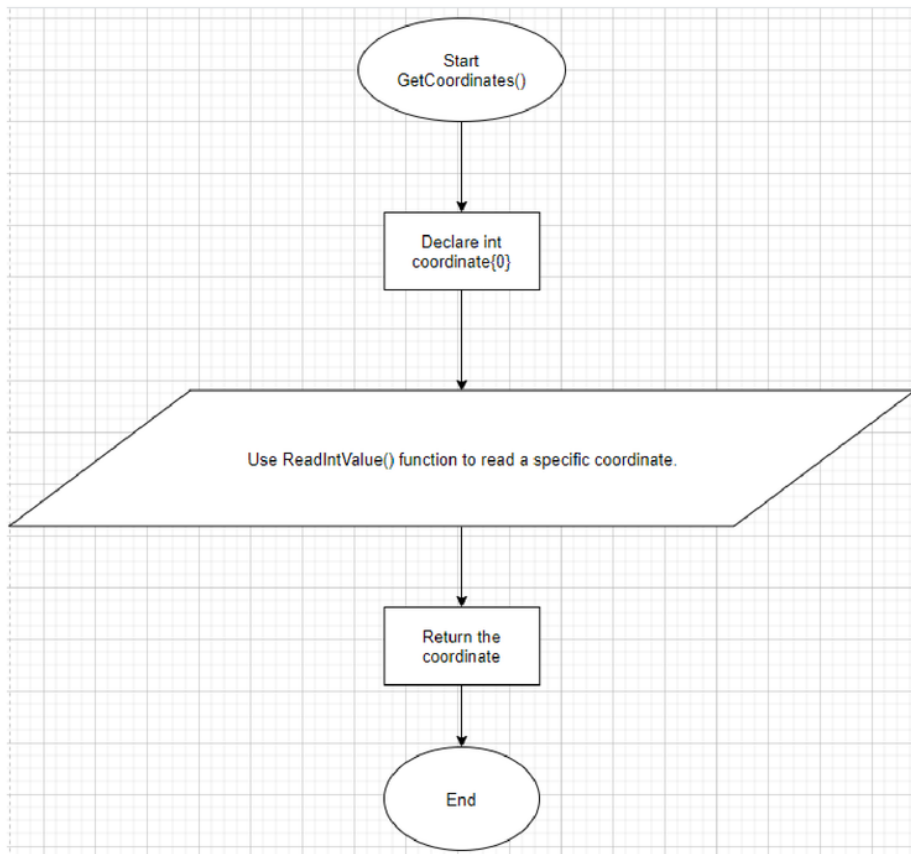
    cout << endl;

    for (int j{ 0 }; j < rows; j++)
    {
        cout << j << " ";

        for (int k{ 0 }; k < columns; k++)
        {
            cout << arrayGrid[j][k] << " ";
        }

        cout << endl;
    }
}
```

Fig.5.2.3 The declared grid is passed to DisplayGrid() function which is used in generating the visualisation of the game grid map.



```
// Get coordinates from players and pass them further
// The ReadIntValue() function is used to read coordinates,
// and the maximum and minimum values are in the boundaries of array elements
int GetCoordinates(string prompt)
{
    int coordinate( 0 );
    coordinate = ReadIntValue(0, 7, prompt);

    return coordinate;
}
```

Fig.5.2.4 A basic function for passing the value of a coordinate, which is set the boundaries of valid inputs.

```
Player 1 place treasures now:
X:0
Y:1
Treasure 1 has been placed
X:2
Y:3
Treasure 2 has been placed
X:4
Y:5
Treasure 3 has been placed
  0 1 2 3 4 5 6 7
0 - - - - - - -
1 - - - - - - -
2 - - - - - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - - -
7 - - - - - - -
X:6
Y:7
```

```
Welcome in EXERCISE 2
Firstly enter a row (X) and then enter a column (Y).
Player 1 place treasures now:
X:-1
X:-2
X:8
X:9
X:7
Y:-1
Y:-2
Y:3
Treasure 1 has been placed
```

```
Welcome in EXERCISE 2
Firstly enter a row (X) and then enter a column (Y).
Player 1 place treasures now:
X:A
X:z
X:$
X:4
Y:q
Y:w
Y:@
Y:2
Treasure 1 has been placed
```

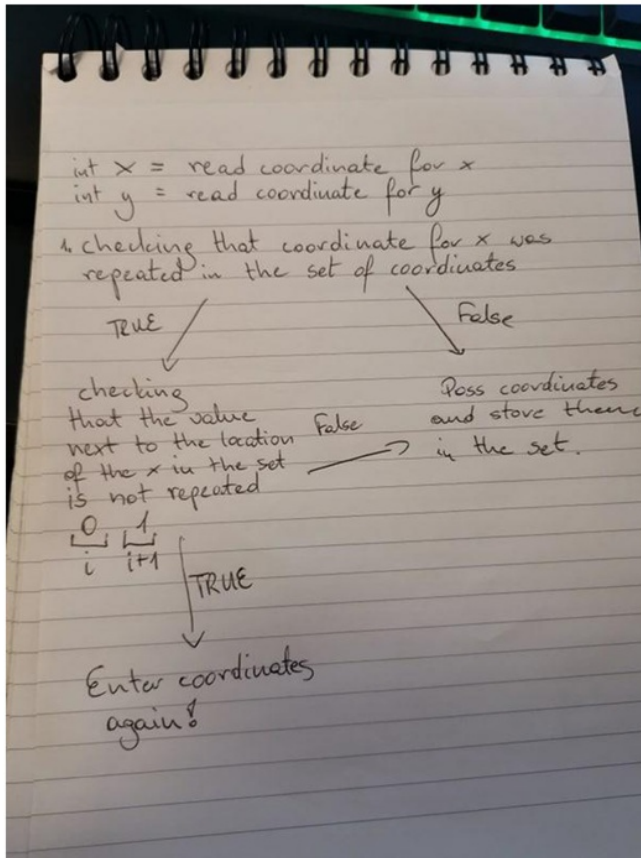
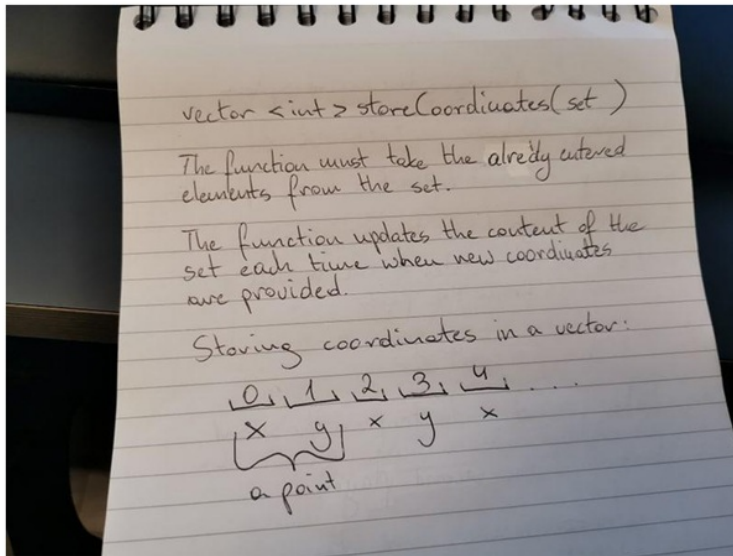
```

Enter the exercise number: 2
Welcome in EXERCISE 2
Firstly enter a row (X) and then enter a column (Y).
Player 1 place treasures now:
X:0.1
X:-0.1
X:1
Y:7.1
Y:6.9
Y:7
Treasure 1 has been placed

```

Attempts	Test Values	Description	Expected Result	Actual Result	Pass / Fail
1	Integers in the range 0-7.	Reading a valid coordinate for X or Y.	All integers are accepted values.	All integers are accepted values.	Pass
2	Integers out of range: X = -1, -2, 8, 9 Y = -1, -2, 3	Reading a valid coordinate for X or Y.	Demanding a valid value by the program.	Demanding a valid value by the program.	Pass
3	Characters: X = A, z, \$ Y = q, W, @	Reading a valid coordinate for X or Y.	Demanding a valid value by the program.	Demanding a valid value by the program.	Pass
4	Doubles: X = 0.1, -0.1 Y = 7.1, 6.9	Reading a valid coordinate for X or Y.	Demanding a valid value by the program.	Demanding a valid value by the program.	Pass

Fig.5.2.4 The examples of GetCoordinates() function in tests. The outcome is that the coordinates passed further in the program can only be an integer in the range between 0-7.



```

// store the new passed coordinates and check are not repeated
vector<int> StoreCoordinates(vector<int> listOfCoordinates)
{
    int x{ 0 };
    int y{ 0 };
    bool isRepeated{ false };

    do
    {
        isRepeated = false;

        x = GetCoordinates("X:"); // getting a valid value for x coordinate

        y = GetCoordinates("Y:"); // getting a valid value for y coordinate

        // start comparing that the coordinates are not repeated if first coordinates are stored already
        if (listOfCoordinates.size() >= 2)
        {
            // check that the guess value for x is already stored at some point
            for (unsigned int i{ 0 }; i < listOfCoordinates.size(); i += 2)
            {
                if (x == listOfCoordinates.at(i))
                {
                    // check that the value next after the position of x in the set is the same as user input for the y coordinate
                    if (y == listOfCoordinates.at(i+1))
                    {
                        isRepeated = true;
                        cout << "This location was already used. Try again!" << endl;
                        break;
                    }
                }
            }
        }
    } while (isRepeated == true);

    // store a new coordinates in the vector
    listOfCoordinates.push_back(x);
    listOfCoordinates.push_back(y);

    return listOfCoordinates;
}

```

Fig.5.2.5 The picture is showing the StoreCoordinates() method. Its main task is to take and store coordinates of the point of the grid. Furthermore, both users use this function to obtain preferred locations. Additionally, the repetition in provided inputs are tracked, thereby entering these same coordinates by one player more than once is impossible.

```

Welcome in EXERCISE 2
Firstly enter a row (X) and then enter a column (Y).
Player 1 place treasures now:
X:1
Y:0
Treasure 1 has been placed
X:0
Y:1
Treasure 2 has been placed
X:7
Y:0
Treasure 3 has been placed

```

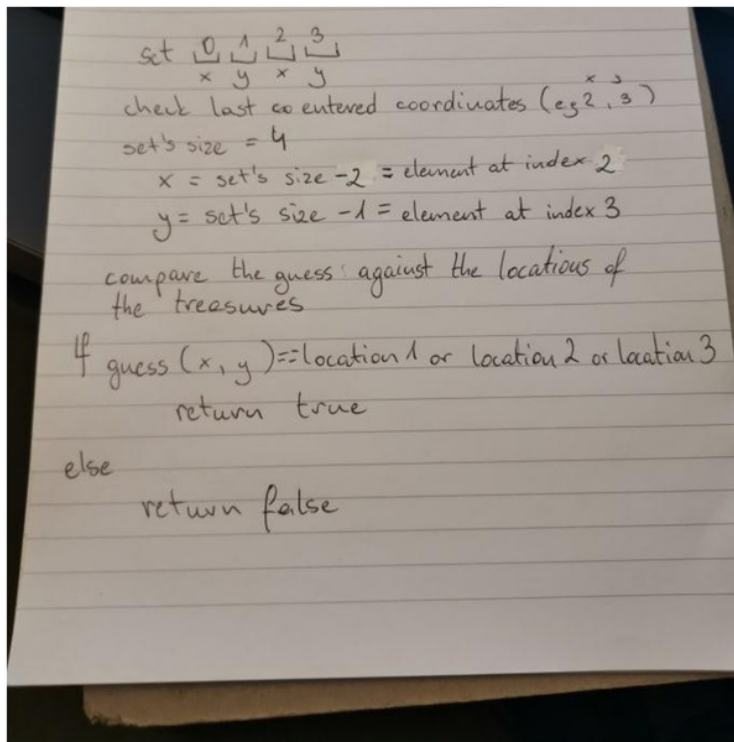
```

Welcome in EXERCISE 2
Firstly enter a row (X) and then enter a column (Y).
Player 1 place treasures now:
X:1
Y:0
Treasure 1 has been placed
X:1
Y:0
This location was already used. Try again!
X:3
Y:0
Treasure 2 has been placed
X:1
Y:0
This location was already used. Try again!
X:3
Y:0
This location was already used. Try again!
X:4
Y:0
Treasure 3 has been placed

```

Attempts	Test Values	Description	Expected Result	Actual Result	Pass / Fail
1	Points: (1,0), (0,1), (7,0)	Storing both coordinates for X and Y as a set of values without repetitions by Players	All values are accepted.	All values are accepted.	Pass
2	Points: (1,0), (1,0), (3,0), (1,0), (3,0), (4,0)	Storing both coordinates for X and Y as a set of values without repetitions by Players/	Demanding a valid value by the program. Display "This location was already used. Try again!".	Demanding a valid value by the program. Display "This location was already used. Try again!".	Pass

Fig.5.2.4 The examples of StoreCoordinates() function in tests. Users can put any point for the grid. However, the repetitions in the set of coordinates are not allowed.



```
// check that guesses are correct or not
// compare coordinates for a guess against coordinates of hidden treasures
bool IsTreasureFound(vector<int> listOfGuesses, vector<int> treasuresCoordinates)
{
    int x = listOfGuesses.at(listOfGuesses.size() - 2); // declare the last stored x
    int y = listOfGuesses.at(listOfGuesses.size() - 1); // declare the last stored y

    bool isFound{ false }; // is a treasure found?

    // is the treasure at the guessing location?
    if (x == treasuresCoordinates.at(0) && y == treasuresCoordinates.at(1)) // 1st location
    {
        isFound = true;
    }
    else if (x == treasuresCoordinates.at(2) && y == treasuresCoordinates.at(3)) // 2nd location
    {
        isFound = true;
    }
    else if (x == treasuresCoordinates.at(4) && y == treasuresCoordinates.at(5)) // 3rd location
    {
        isFound = true;
    }
    else
    {
        isFound = false;
    }

    return isFound;
}
```

Fig.5.2.6 The list of guesses is checked by the system of logic gates thanks to using passed localisations of the treasures. Next, the positive or negative value is returned by the bool function further.

```
Player 1 place treasures now: X:1
X:1 Y:0
Y:0 Treasure 1 has been placed You found the treasure!
```

```
Player 1 place treasures now: X:2
X:2 Y:4
Y:0 No treasure here, you have 2 lives left!
```

Attempts	Test Values	Description	Expected Result	Actual Result	Pass / Fail
1	Player 1 input: (1,0) Player 2 input: (1,0)	Comparing input entered by Player 2 against data entered by Player 1.	Values are matched.	Values are matched.	Pass
2	Player 1 input: (2,0) Player 2 input: (2,4)	Comparing input entered by Player 2 against data entered by Player 1.	Values are not matched.	Values are not matched.	Pass

Fig.5.2.4 The examples of IsTreasureFound() function in tests. If coordinates enter by Player 2, correspond to the location of a treasure the “true” is returned and passed further.

Game Core

set: - number of lives {3}
- number of founded treasures {0}
- list of coordinates (0) → dynamic in size, it will be updated

loop (do until the number of lives is equal to zero or the number of revealed treasures is equal to three)

1. update the list of coordinates
2. check that is found

if (is found)

founded treasures → increases +1
update the grid → put "X"

else

Lives -= 1

update the grid → put "N"

display → grid

check who is the winner

```

// the core game function displaying gameplay and checking which player is a winner
void StartGameCore(int numberOfTreasures, vector<int> treasuresCoordinates, string grid[rows][columns])
{
    int lives[ 3 ]; // number of lives at start
    int foundedTreasures{ 0 }; // number of found treasures
    vector<int> listOfCoordinates(0); // vector for storing guesses of player 2
    bool isFound{ false }; // is a treasure found?

    // do until the number of lives is reduced to 0, or all treasures are revealed
    while (lives > 0 && foundedTreasures < numberOfTreasures)
    {
        listOfCoordinates = StoreCoordinates(listOfCoordinates); // store guesses

        isFound = isTreasureFound(listOfCoordinates, treasuresCoordinates); // check the guess is right or not, then store it

        if (isFound == true) // is the guess right, and a treasure is found?
        {
            foundedTreasures++; // increase a number of found treasures
            cout << "You found the treasure!" << endl;

            // pass the guessing location to the grid and set X
            grid[listOfCoordinates.at(listOfCoordinates.size() - 2)][listOfCoordinates.at(listOfCoordinates.size() - 1)] = "X";
        }
        else
        {
            lives -- 1;
            cout << "No treasure here, you have " << lives << " lives left!" << endl;

            // pass the guessing location to the grid and set Y
            grid[listOfCoordinates.at(listOfCoordinates.size() - 2)][listOfCoordinates.at(listOfCoordinates.size() - 1)] = "N";
        }

        isFound = false; // reset a guess checker
        DisplayGrid(grid);
    }

    cout << endl;

    // checking who is the winner
    if (foundedTreasures == numberOfTreasures)
    {
        cout << "Congratulations! Player 2 wins!" << endl;
    }
    else
    {
        cout << "Congratulations! Player 1 wins!" << endl;
    }
}

```

Fig.5.2.7 The figures are presenting the game core function implementation for the second exercise. Accordingly, the main gameplay is resolved at this point. Each attempt of finding a treasure by the second player is noticed and stored in the set of guesses, which is used to checking that the coordinates are not repeated. Furthermore, in function, a number of lost life points and discovered treasures is counted and displayed. After putting the right or wrong answer the grid is updated and presented to users. At the place of right location, "X" is initialising, while the empty location is marked as "N". The result of the game is obtained by checking that all treasures are found by the second user or when all points of life are equal to zero.

```
X:1
Y:0
You found the treasure!
  0 1 2 3 4 5 6 7
0 - - - - - - -
1 X - - - - - -
2 - - - - - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - - -
7 - - - - - - -
X:2
Y:3
No treasure here, you have 2 lives left!
  0 1 2 3 4 5 6 7
0 - - - - - - -
1 X - - - - - -
2 - - - N - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - - -
7 - - - - - - -
X:6
Y:7
You found the treasure!
  0 1 2 3 4 5 6 7
0 - - - - - - -
1 X - - - - - -
2 - - - N - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - X
7 - - - - - - -
X:2
Y:1
You found the treasure!
  0 1 2 3 4 5 6 7
0 - - - - - - -
1 X - - - - - -
2 - X - N - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - X
7 - - - - - - -

Congratulations! Player 2 wins!
*****
```

```
  0 1 2 3 4 5 6 7
0 - - - - -
1 - - - - -
2 - - - - -
3 - - - - -
4 - - - - -
5 - - - - -
6 - - - - -
7 - - - - -
X:1
Y:1
No treasure here, you have 2 lives left!
  0 1 2 3 4 5 6 7
0 - - - - -
1 - N - - - -
2 - - - - -
3 - - - - -
4 - - - - -
5 - - - - -
6 - - - - -
7 - - - - -
X:2
Y:3
No treasure here, you have 1 lives left!
  0 1 2 3 4 5 6 7
0 - - - - -
1 - N - - - -
2 - - - N - -
3 - - - - -
4 - - - - -
5 - - - - -
6 - - - - -
7 - - - - -
X:6
Y:7
You found the treasure!
  0 1 2 3 4 5 6 7
0 - - - - -
1 - N - - - -
2 - - - N - -
3 - - - - -
4 - - - - -
5 - - - - -
6 - - - - - X
7 - - - - -
X:3
Y:3
No treasure here, you have 0 lives left!
  0 1 2 3 4 5 6 7
0 - - - - -
1 - N - - - -
2 - - - N - -
3 - - - N - -
4 - - - - -
5 - - - - -
6 - - - - - X
7 - - - - -

Congratulations! Player 1 wins!
*****
```

Attempts	Test Values	Description	Expected Result	Actual Result	Pass / Fail
1	Player 1 inputs: (1,0), (2,1), (6,7) Player 2 inputs: (1,0), (2,3), (6,7), (2,1)	Entering coordinates by Player 2 and updating the grid.	“X” at position (1,0), (6,7), (2,1) “N” at position (2,3)	“X” at position (1,0), (6,7), (2,1) “N” at position (2,3)	Pass
2	Player 1 inputs: (1,0), (2,1), (6,7) Player 2 inputs: (1,0), (2,3), (6,7), (2,1)	Checking who is the winner.	Player 2 is the winner.	Player 2 is the winner.	Pass
3	Player 1 inputs: (1,0), (2,1), (6,7) Player 2 inputs: (1,1), (2,3), (6,7), (3,3)	Checking who is the winner.	Player 1 is the winner.	Player 1 is the winner.	Pass

Fig.5.2.4 The examples of StartGameCore() function in tests. The provided data from Player 2 is implemented and compared with the coordinates of treasures. Furthermore, each guess is stored and is resulted in updating the grid. At the end the status of remaining lives and discovered treasures is checked, then a winner is indicated by the application.

2. Conclusions

Summarising, the process of software development was accomplished after finishing the second exercise. However, the methods of obtaining solutions for specific parts of the program have been various. After founding out the underlying constraints and requirements, a logical

order for a particular exercise was considered. Accordingly, tasks were divided into small, more simple objectives to achieve.

Moreover, each stage of application development was thought in an abstract form by using written notes, pseudo code, diagrams or simple sketches and then implemented into a code. The project finalisation was a series of different tests on values and mechanics of the games. Furthermore, the results were included in this report.

5. Bibliography

- [1] M. McGrath, C++ Programming in easy steps, 5th ed., In Easy Steps Limited, 2017.
- [2] cplusplus.com, "cplusplus.com," [Online]. Available: <http://www.cplusplus.com/>.
- [3] GeeksforGeeks, "C++ Programming Language," [Online]. Available: <https://www.geeksforgeeks.org/c-plus-plus/>.

6. Appendix

```
// Bartosz Galantowicz 1700994
// Application of Maths for Software Development
// 27/03/2019

#include<iostream> // standard input and output library
#include<string> // string library
#include<sstream> // input/output from strings library
#include<vector> // container for vectors library

using namespace std;

void DisplayMenu();
int ReadIntValue(int minValue, int maxValue, string prompt);
void ChoseExercise(int userChoice);
double ReadNumber(string prompt);

// Exercise 1 - Functions Prototypes:
vector<double> StoringPlayerOneValues();
char ReadPlayerTwoGuess();
void CheckingGuess(vector<double> setOfSevenValues);
const int maxNumberOfValues{ 7 };

// Exercise 2 - Functions Prototypes:
const int rows { 8 };
const int columns{ 8 };
int GetCoordinates(string prompt);
void DisplayGrid(string arrayGrid[rows][columns]);
vector<int> StoreCoordinates(vector<int> listOfGuesses);
bool IsTreasureFound(vector<int> listOfGuesses, vector<int> treasuresCoordinates);
void StartGameCore(int numberOfTreasures, vector<int> treasuresCoordinates, string
grid[rows][columns]);
```

```
int main()
{
    int userChoice{ 0 };

    do
    {
        DisplayMenu();
        userChoice = ReadIntValue(1, 3, "Enter the exercise number: ");
        ChoseExercise(userChoice);

    } while (userChoice != 3);

    return 0;
}

// SUPPORTING FUNCTIONS

void ChoseExercise(int userChoice)
{
    switch (userChoice)
    {
        // Exercise 1
        case 1:
        {
            vector <double> setOfSevenValues;

            cout << "Welcome in EXERCISE 1" << endl;

            cout << "Player 1, please enter seven values without repetition:"

<< endl;

            setOfSevenValues = StoringPlayerOneValues();

            cout << "Player 2 is your turn! The first value is " <<
setOfSevenValues.at(0) << endl;

            CheckingGuess(setOfSevenValues);
            break;
        }

        // Exercise 2
        case 2:
        {
            const int numberOfTreasures{ 3 };

            cout << "Welcome in EXERCISE 2" << endl;

            cout << "Firstly enter a row (X) and then enter a column (Y)." <<

endl;

            cout << "Player 1 place treasures now:" << endl;

            // locating the treasures
            vector <int> treasuresCoordinates;

            for (int i{ 0 }; i < numberOfTreasures; i++)
            {
                treasuresCoordinates =
StoreCoordinates(treasuresCoordinates);
```

```

        cout << "Treasure " << i + 1 << " has been placed" <<
endl;
    }
    // initialising the grid
    string grid[rows][columns]
    {
        {"-", "-", "-", "-", "-", "-", "-", "-"}, //0 row
        {"-", "-", "-", "-", "-", "-", "-", "-"}, //1st row
        {"-", "-", "-", "-", "-", "-", "-", "-"}, //2nd row
        {"-", "-", "-", "-", "-", "-", "-", "-"}, //3rd row
        {"-", "-", "-", "-", "-", "-", "-", "-"}, //4th row
        {"-", "-", "-", "-", "-", "-", "-", "-"}, //5th row
        {"-", "-", "-", "-", "-", "-", "-", "-"}, //6th row
        {"-", "-", "-", "-", "-", "-", "-", "-"}, //7th row
    };

    DisplayGrid(grid);

    StartGameCore(numberOfTreasures, treasuresCoordinates, grid);

    break;
}
case 3:
    cout << endl;
    cout << "Exiting the application...";
    break;
}
}

void DisplayMenu()
{
    cout << "*****" << endl;
    cout << endl;

    cout << "Exercise menu: " << endl;
    cout << "1. EXERCISE 1" << endl;
    cout << "2. EXERCISE 2" << endl;
    cout << "3. Exit" << endl;
}

// reading integer values
// passing min and max value allows for setting the function
int ReadIntValue(int minValue, int maxValue, string prompt)
{
    double userValue{ 0 };

    // avoiding confusion when a user enters decimal values
    do
    {
        userValue = ReadNumber(prompt);

        // checking that the chosen value is not a decimal number
    } while ((userValue - int(userValue) > 0) || (userValue < minValue || userValue
> maxValue));

    // casting double into int
    int userIntValue = static_cast<int> (userValue);

    return userIntValue;
}

```

```

// the function reads the user input and checks that is a number
double ReadNumber(string prompt)
{
    double value{};
    string input{};
    stringstream stream{};

    cout << prompt;
    getline(cin, input);
    stream.str(input); // Empty the contents
    stream.clear(); // Empty the bits flags

    while ((stream >> value && stream.eof()) == false)
    {
        cout << prompt;
        stream.clear();
        getline(cin, input);
        stream.str(input);
    }

    stream >> value; // convert the string into a integer value

    return value;
}

// EXERCISE 2

// the core game function displaying gameplay and checking which player is a winner
void StartGameCore(int numberOfTreasures, vector<int> treasuresCoordinates, string
grid[rows][columns])
{
    int lives{ 3 }; // number of lives at start
    int foundedTreasures{ 0 }; // number of found treasures
    vector<int> listOfCoordinates(0); // vector for storing guesses of player 2
    bool isFound{ false }; // is a treasure found?

    // do until the number of lives is reduced to 0, or all treasures are revealed
    while (lives > 0 && foundedTreasures < numberOfTreasures)
    {
        listOfCoordinates = StoreCoordinates(listOfCoordinates); // store
guesses

        isFound = IsTreasureFound(listOfCoordinates, treasuresCoordinates); //
check the guess is right or not, then store it

        if (isFound == true) // is the guess right, and a treasure is found?
        {
            foundedTreasures++; // increase a number of found treasures
            cout << "You found the treasure!" << endl;

            // pass the guessing location to the grid and set X
            grid[listOfCoordinates.at(listOfCoordinates.size() -
2)][listOfCoordinates.at(listOfCoordinates.size() - 1)] = "X";
        }
        else
        {
            lives -= 1;
            cout << "No treasure here, you have " << lives << " lives left!"
<< endl;

            // pass the guessing location to the grid and set Y

```

```

        grid[listOfCoordinates.at(listOfCoordinates.size() -
2)][listOfCoordinates.at(listOfCoordinates.size() - 1)] = "N";
    }

    isFound = false; // reset a guess checker
    DisplayGrid(grid);
}

cout << endl;

// checking who is the winner
if (foundedTreasures == numberOfTreasures)
{
    cout << "Congratulations! Player 2 wins!" << endl;
}
else
{
    cout << "Congratulations! Player 1 wins!" << endl;
}
}

// check that guesses are correct or not
// compare coordinates for a guess against coordinates of hidden treasures
bool IsTreasureFound(vector<int> listOfGuesses, vector<int> treasuresCoordinates)
{
    int x = listOfGuesses.at(listOfGuesses.size() - 2); // declare the last stored
x
    int y = listOfGuesses.at(listOfGuesses.size() - 1); // declare the last stored
y

    bool isFound{ false }; // is a treasure found?

    // is the treasure at the guessing location?
    if (x == treasuresCoordinates.at(0) && y == treasuresCoordinates.at(1)) // 1st
location
    {
        isFound = true;
    }
    else if (x == treasuresCoordinates.at(2) && y == treasuresCoordinates.at(3)) //
2nd location
    {
        isFound = true;
    }
    else if (x == treasuresCoordinates.at(4) && y == treasuresCoordinates.at(5)) //
3rd location
    {
        isFound = true;
    }
    else
    {
        isFound = false;
    }

    return isFound;
}

// store the new passed coordinates and check are not repeated
vector<int> StoreCoordinates(vector<int> listOfCoordinates)
{
    int x{ 0 };
    int y{ 0 };
    bool isRepeated{ false };

```

```

do
{
    isRepeated = false;

    x = GetCoordinates("X:"); // getting a valid value for x coordinate
    y = GetCoordinates("Y:"); // getting a valid value for y coordinate

    // start comparing that the coordinates are not repeated if first
coordinates are stored already
    if (listOfCoordinates.size() >= 2)
    {
        // check that the guess value for x is already stored at some
point
        for (unsigned int i{ 0 }; i < listOfCoordinates.size(); i += 2)
        {
            if (x == listOfCoordinates.at(i))
            {
                // check that the value next after the position of
x in the set is the same as user input for the y coordinate
                if (y == listOfCoordinates.at(i+1))
                {
                    isRepeated = true;
                    cout << "This location was already
used. Try again!" << endl;
                    break;
                }
            }
        }
    }

    } while (isRepeated == true);

    // store a new coordinates in the vector
    listOfCoordinates.push_back(x);
    listOfCoordinates.push_back(y);

    return listOfCoordinates;
}

// Display the grid to users
void DisplayGrid(string arrayGrid[rows][columns])
{
    cout << " ";

    for (int i{ 0 }; i < columns; i++)
    {
        cout << " " << i;
    }

    cout << endl;

    for (int j{ 0 }; j < rows; j++)
    {
        cout << j << " ";

        for (int k{ 0 }; k < columns; k++)
        {
            cout << arrayGrid[j][k] << " ";
        }
    }
}

```

```

        cout << endl;
    }
}

// Get coordinates from players and pass them further
// The ReadIntValue() function is used to read coordinates,
// and the maximum and minimum values are in the boundaries of array elements
int GetCoordinates(string prompt)
{
    int coordinate{ 0 };
    coordinate = ReadIntValue(0, 7, prompt);

    return coordinate;
}

// EXERCISE 1

void CheckingGuess(vector<double> setOfSevenValues)
{
    bool isGuessed = true;
    int counter{ 0 };
    char playerTwoInput{};

    while(counter < maxNumberOfValues && isGuessed == true)
    {
        playerTwoInput = ReadPlayerTwoGuess();

        if ((setOfSevenValues.at(counter + 1) > setOfSevenValues.at(counter)) &&
            (playerTwoInput == 'l' || playerTwoInput == 'L'))
        {
            // if the value is different than user input then player 2 loose
            cout << endl;
            cout << "Incorrect, sorry player 1 won!" << endl;
            isGuessed = false;
        }
        else if((setOfSevenValues.at(counter + 1) <
            setOfSevenValues.at(counter)) && (playerTwoInput == 'h' || playerTwoInput == 'H'))
        {
            // if the value is different than user input then player 2 loose
            cout << endl;
            cout << "Incorrect, sorry player 1 won!" << endl;
            isGuessed = false;
        }
        else
        {
            // when user input is correct go to next value
            cout << "Correct, next value is " << setOfSevenValues.at(counter
+ 1) << endl;

            counter++;

            // player 2 wins if all values are guessed
            if (isGuessed == true && counter == maxNumberOfValues-1)
            {
                cout << endl;
                cout << "Congratulations! Player 2 won!" << endl;
                break;
            }
        }
    }
}

char ReadPlayerTwoGuess()

```

```

{
    char input;
    bool isValid{ true };

    do
    {
        cout << "Higher (H) or Lower (L)>";
        cin >> input; // get a line of input
        isValid = true; // assume it is valid
        cin.clear(); // clear the error flags
        cin.ignore(1000, '\n'); // discard no needed characters from the input
        stream until a new line

        if (!isalpha(input)) // is it an alphabetical character?
        {
            isValid = false; // if not, mark it as invalid

            // print an error to the user
            cout << "Invalid input. Please input only alphabetical
characters." << endl;
        }
        else if (input != 'h' && input != 'l' && input != 'H' && input != 'L')
        // is it h, H, l, or L?
        {
            isValid = false; // if not, mark it as invalid
            cout << "Invalid input. Please input ""h"" or ""H"" for higher
and ""l"" or ""L"" for lower." << endl;
        }

        } while (!isValid); // keep going until the input is valid

    return input;
}

vector<double> StoringPlayerOneValues()
{
    vector<double> setOfSevenValues;
    int counter{ 0 };
    bool isRepeated{ false };

    for (int i{ 0 }; i < maxNumberOfValues; i++)
    {
        counter = 0; // resetting the counter

        do
        {
            cout << "Enter number " << i + 1 << ">";
            setOfSevenValues.push_back(ReadNumber("")); // getting value from
the user

            for (unsigned int j{ 0 }; j < setOfSevenValues.size(); j++)
            {
                if (setOfSevenValues.at(i) == setOfSevenValues.at(j))
                {
                    counter++; // updating the counter when given value
have been already entered
                }

                if (counter > 1) // is more than one examples of given
values
                {
                    isRepeated = true;

```

```

                                setOfSevenValues.pop_back(); // discarding last
implement value                cout << "Sorry but this values is already in the
set. Try again. ";              counter = 0; // reseting the counter
                                break;
                                }
                                else
                                {
                                    isRepeated = false; // if in the set are not
repetitions break do...while loop condition
                                    }
                                }
                                } while (isRepeated == true); // keep going until the input is valid
                                counter = 0;
                                }
                                return setOfSevenValues;
}
```

FINAL GRADE

77/100

GENERAL COMMENTS

Instructor

Exercise 1 (50) 48
Exercise 2 (30) 29
Exercise 3 (20) 0

Exercise 1 completed with a clean solution. Evidence of problem solving and testing is quite good.

Exercise 2 is a good solution and you have played with vectors().

Nice evidence of problems solving and testing.

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1

I do not like this syntax, the {} are not required

PAGE 5



Comment 2

The brackets following the case statement are not necessary



Comment 3

Perhaps this code would have been better placed in a function

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40



Comment 4

I much prefer `i = 0`

PAGE 41



Comment 5

This code could be split up more cleanly e.g.

```
PlaceTreasures(treasuresCoords);
```

```
InitialiseGrid(grid);
```

```
Display(grid);
```

```
StartGame(treasureCoords, 3, .grid);
```

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47
