

**Software Development**

**Problem Based Learning**

**“Morra” Game**

**Group J**

Course Title: Higher Diploma in Science in Computing (Software Development)

Date of Submission: 14/04/2020

Lecturer: Adriana E. Chis

|  |  |
| --- | --- |
| Students: | Raminta Kairyte x01416242  Michele Di Benedetto x19189141  Hsiu Hui Huang x19225806 |

**Table of Content**

Problem Based Learning3

Introduction3

PBL Development4

Workload distribution4

Class Diagram6

Input, Main processing and Output (IPO)8

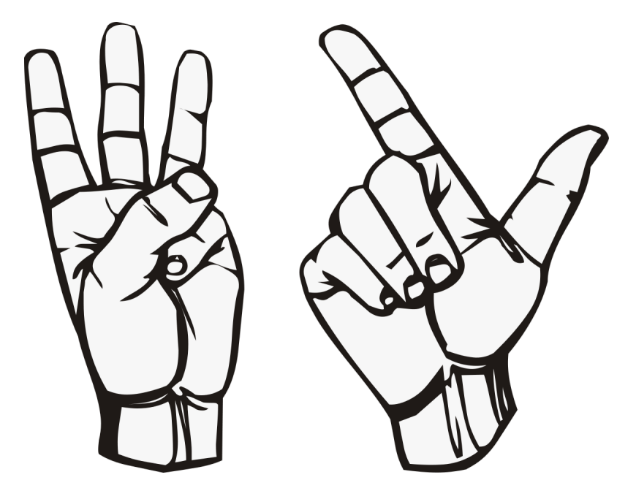
Design and Implementation decisions 14

Manual Testing17

Conclusion 24

**Introduction**

The purpose of this assignment is to acquire practical skills of the software development-based project. The learning process is based on the concept of the “Problem-Solving Learning” (PBL). The project requires all students actively participate in the teamwork, analyze and understand the substance of the problem, autonomously provide the solution of certain parts of the project and engage in the problem-solving tasks. The main goal of the PBL development is to extend the student’s knowledge and to gain specialized skills.



**PBL Development**

The implementation of the project started from the analysis and comprehension of the projects’ scope. The objective of the project was to develop the “Morra” game application. The game would be played by a person against the computer (virtual player). The payer can pick their preferred side (ODD or EVEN) and provide the number between 1 and 10. The sum of the numbers will determine the winning side. The player first to collect the 12 points becomes the winner.

**Workload distribution**

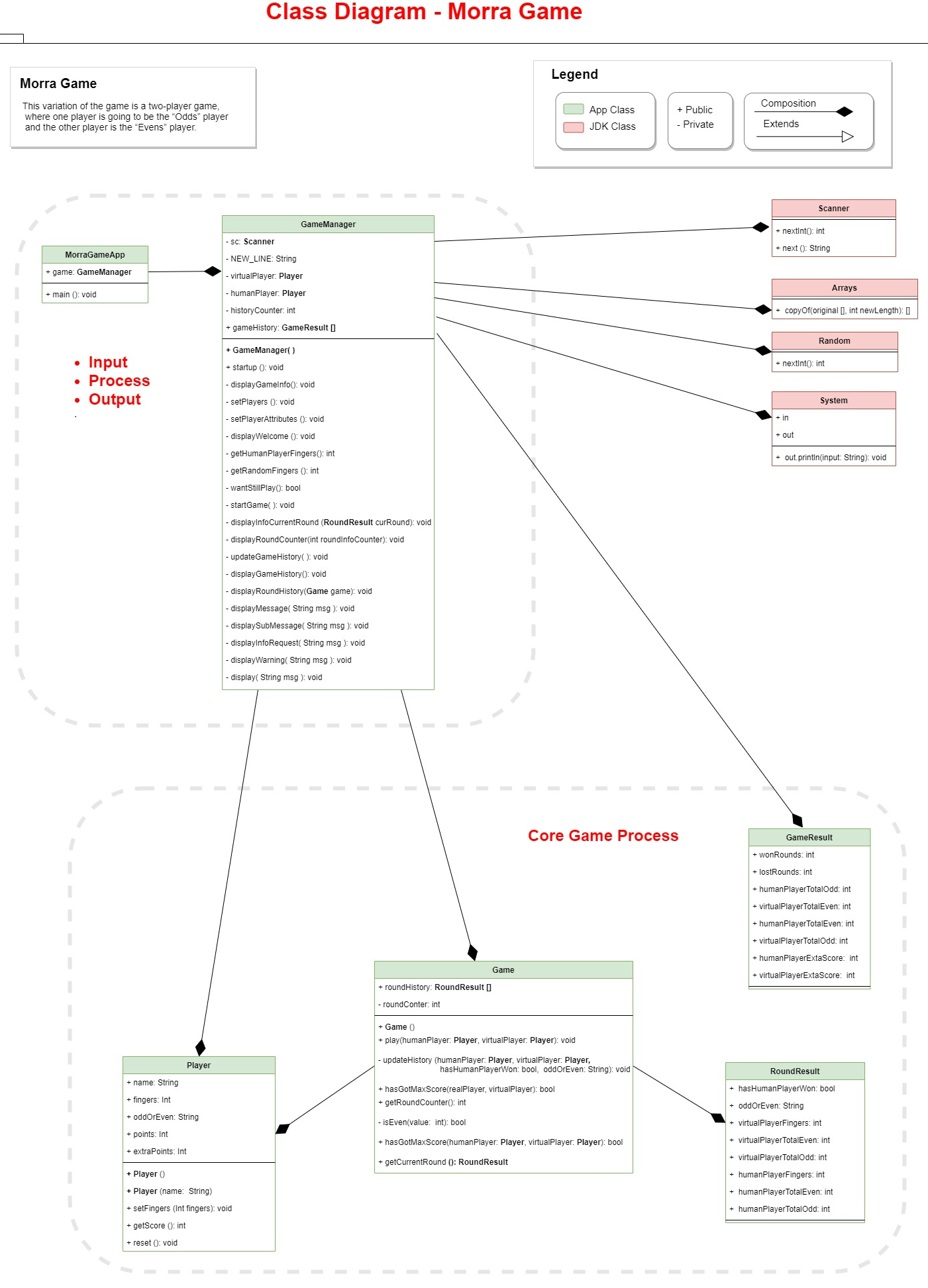
The team of this project implementation consisted of three people. On the first meeting we started the discussion about the project requirements. We defined the possible input, main processing and output of the application. Then we determined the required classes, its attributes and methods, also, the relationship and dependency between them. When the applications’ scope was defined, we discussed the distribution of the workload and assigned individual tasks to each member of the group.

The Scope of the application was divided accordingly:

|  |  |
| --- | --- |
| **Member** | **Tasks** |
| **Michele Di Benedetto** | 1. DESIGN CLASS DIAGRAM  2. GAME RULES  FULL Class implementation  1. MorraGameApp  2. Game  PARTIAL Class implementation  1. GameManager.GameManager()  2. GameManager.startup()  3. GameManager.startGame()  4. GameManager.updateGameHistory()  5. GameManager.displayWelcome()  6. GameManager.displayRoundCounter()  7. GameManager.displayInfoCurrentRound()  8. GameManager.display()  9. GameManager.displayMessage()  10. GameManager.displayInfoRequest()  11. GameManager.displayWarning()  12. GameManager.displaySubMessage() |
| **Hsiu Hui Huang** | FULL Class implementation  1. Player  PARTIAL Class implementation  2. GameManager.getRandomFingers()  3. GameManager.setPlayers()  4. GameManager.setPlayerAttributes()  5. GameManager.getHumanPlayerFingers()  1.TESTING  2.BUG REPORTING |
| **Raminta Kairyte** | FULL Class implementation  1. GameResult  2. RoundResult  PARTIAL Class implementation  1. GameManager.displayGameInfo()  2. GameManager.wantStillPlay()  3. GameManager.displayGameHistory()  4. GameManager.displayRoundHistory()  1.REPORT/DOCUMENTATION  2.TESTING  3.BUG REPORTING |

**Class Diagram**

To visualize the static view and to show the different objects of the application, the Class Diagram was modeled. The Diagram consists of six game application classes and four JDK library classes.



**Input, Main processing and Output (IPO)**

**JDK classes:**

1. **Scanner** – used to take the input from the user. Data types: int, String.
2. **Arrays –** used to display the game’s rounds and history.   
   Data types GameResult, RoundResult
3. **Random –** used to display a random number (virtual player selection). Data types: int.
4. **System –** used to display get the input of the user (from Scanner). Data types: int.

**Application classes:**

1. **MorraGameApp**

This class uses GameManager class to display and start the game.   
Contains the *main* method that is used to run the application.

1. **GameManager**

This class is the main class. It contains:

* Scanner – to take the input from the user (selection of fingers from 1 to 10). Data type: int.
* Player class (instantiable class) - to create two new objects of a type of Player – virtualPlayer and humanPlayer. The object virtualPlayer represents the computer that uses the Random class (from JDK Library) to provide finger selection. The object humanPlayer represents the user who will provide the finger selection via Scanner (from JDK Library).
* historyCounter – counts the games played and used to display the Game history. Data type – int
* GameHistory – stores all game result. A new array from the GameResult is used with a length of 100 (maximum of the games displayed in the history view).
* startup() method – this method is used to manage the entire lifecycle of the application.
* displayGameInfo() method – this method is used to display the Rules of the game and to inform the user how to use the application.
* setPlayers() – method that created two new objects of the type Player – virtualPlayer and humanPlayer. The Virtual Player has a given name – “Baby Yoda”. The real (human) player can input its own name. If no input was provided, the real played will be assigned a name of “Player 1”.
* setPlayerAttributes() – this method assigns the required elements for the player – it asks to input the winning likelihood (ODD or EVEN) and provide the number of fingers.
* displayWelcome() – this method displayed the “Welcome” message for the user.
* getHumanPlayerFingers() – the user provides the number of fingers. Data type for the input: int. The method setFingers() to set the fingers are in the instantiable class Player.
* getRandomFingers() – the virtualPlayer (computer) provides the number of fingers (generated by the Random JDK class). Generated data type: int. The method setFingers() to set the fingers are in the instantiable class Player.
* wantStillPlay() – this method asks the user to continue the game or not and directs the user to the selected channel. Data type: Boolean.
* startGame() – this method processes the game: it takes the input from the user, takes the Random number selection from the virtual player, calculated the winner according to the game rules and displays the winning information.
* displayInfoCurrentRound() – this method displays the number of points that each player has.
* displayRoundCounter() – this method displays the current round number. Takes a variable roundInfoCounter of the data type int.
* updateGameHistory() – this method updates the game history. It adds the newly played game round to the already played ones. It uses instantiable classes of the Game (new object game) and Player (new objects humanPlayer and virtualPlayer). Also, it resizes the array: if the length (of 100) is reached, it doubles it.
* displayGameHistory() – this method displayed the history of all played games and provided information of each round. It uses the historyCounter to keep track of the number of games played. Also, it uses the instantiable class of GameResult to display the information.
* displayRoundHistory() - displayed the data of the currently played game (one round). It uses instantiable class Game to provide and display the information. It sets the array roundHistory of the class RoundResult and uses the roundCounter to keep track of the array size. It displays which player won the round, the number of fingers picked by each player and the winning likelihood winning side chosen by the player.

1. **Player**

This class is used to create new objects of the type Player. In this app there are two objects created: virtualPlayer and humanPlayer. The object virtualPlayer is the computer that uses Random class (from JDK Library) to provide finger selection. The user (human player) can select one of the winning likelihood types: either ODD or EVEN. Also, the user provides the finger selection via Scanner (from JDK Library). The data type used for the fingers is integer (int) and its range is from 1 to 10. The variables name and oddOrEven are String types, the remaining of the variables – fingers, points, extraPoints are the int types. The constructor Player (that takes the String name) sets the Players’ name. The default constructor Player() sets the name to a default value of “Player-1”. A get method getScore() returns the number of points and adds any extra points received from the round and stores the new value in the getScore(). A method setFingers() provides the value of the fingers for the both players. Human player uses Scanner(System.in) to provide the value by the input. The Virtual player uses Random() method to generate a number between 1 and 10. The method reset() is used to set all of the variables back to their starting point: int variables fingers, points, extraPoints are set to 0 (zero), and oddorEven is set to an empty String value (“”).

1. **Game**

This class processes the data of the rounds of the played games. It uses a new array of the class RoudResults and its variables to provide the user with the statistical data of the game. The new array roundHistory is created with a length of 8. This length was chosen because the number of rounds cannot reach over eight until one of the players collects twelve points and wins the game. The roundCounter is set to zero and it keeps track of the rounds played. Variables POINTS and EXTRA\_POINTS were set as final and assigned values of 3 and 2 accordingly. The values will remain the same and cannot be amended in the application. The method play() calculates the points gained by each player. The player who chose the correct wining likelihood will win he round and receive 3 points. Two points will be awarded to the player who would have the number of the chosen fingers closer to the total sum. In a case if both players chose the same number of fingers, no extra points will be given to the players. The method updateHistory() is used to provide the round information to the game. It uses the instantiable classes Player and RoundResult variables to define the winner, the likelihood winning type, and the number of fingers chosen by the players. The method hasGotMaxScore() uses a constant of the final int MAX\_SCORE to set the end of the game once one of the players will reach 12 points.

1. **RoundResult**

This class is used as a part of the display implementation of the played rounds in the game application. This is an instantiable class that is used in the classes Game and GameManager to provide the user with the statistical data of the game. It is used as an array and the values are stored in each of the variable. All variable data types were assigned a public access modifier.

The variables that the RoundResult class contains:

* hasHumanPlayerWon – Boolean data type. It displays the possible outcome of the game: True (Human player won) or False (Virtual Player won).
* virtualPlayerFingers – int data type. Displays the number of fingers that were chosen by the Virtual player (computer).
* virtualPlayerTotalEven – int data type. Displays the number of the likelihood winning type EVEN chosen by the Virtual Player.
* virtualPlayerTotalOdd - int data type. Displays the number of the likelihood winning type ODD chosen by the Virtual Player.
* humanPlayerFingers – int data type. Displays the number of fingers that were chosen by the Human player (user).
* humanPlayerTotalOdd - int data type. Displays the number of the likelihood winning type ODD chosen by the Human Player.
* virtualPlayerTotalEven - int data type. Displays the number of the likelihood winning type EVEN chosen by the Virtual Player.
* oddOrEven – String data type variable. It takes the human players input from the Scanner and sets the likelihood winning type of the players choice (ODD or EVEN).

1. **GameResult**

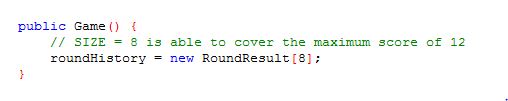
This class is used as a part of the display implementation for the Game History in the application. This is an instantiable class that is used in the GameManager class. It provides the user with the statistical data of the game. When implemented as an array, stores the number of rounds won and lost by the player, the winning likelihood chosen by each player, also any extra points received by each player. All variable data types were assigned a public access modifier.

The variables that the GameResult class contains:

* wonRounds – Displays the total number of rounds won by the Human Player. Data type – integer (int).
* lostRounds - Displays the total number of rounds lost by the Human Player. Data type – integer (int).
* humanPlayerTotalOdd - Displayed the number of the likelihood winning type ODD chosen by the Human Player. Data type – integer (int).
* virtualPlayerTotalEven - Displayed the number of the likelihood winning type EVEN chosen by the Virtual Player. Data type – integer (int).
* humanPlayerTotalEven - Displayed the number of the likelihood winning type EVEN chosen by the Human Player. Data type – integer (int).
* virtualPlayerTotalOdd - Displayed the number of the likelihood winning type ODD chosen by the Virtual Player. Data type – integer (int).
* humanPlayerExtaScore - Displayed the number of extra points received by the Human Player. Data type – integer (int).
* virtualPlayerExtaScore - Displayed the number of extra points received by the Virtual Player. Data type – integer (int).

**Design and Implementation Decisions**

1. **Array size: roundHistory = RoundResult[8]**



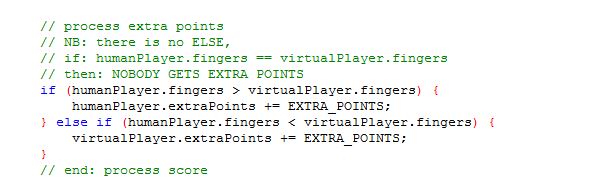
A new array RoundResult is created (of a type of roundHistory) to store and later to display the result of the round. The array has a length of 8 because it would not take more than 8 rounds to collect 12 points and win the game by any player.

1. **Constant variables (final)**



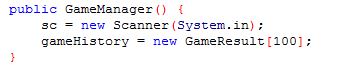
Variables POINTS and EXTRA\_POINTS were set as Constants (defined by the keyword final) and assigned values of 3 and 2 accordingly. The values will remain the same and cannot be amended in the application.

1. **Extra points**



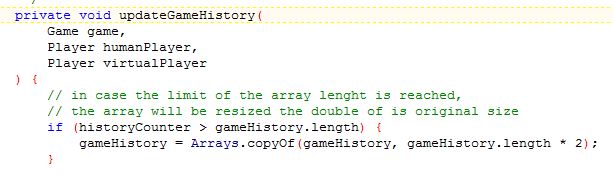
This selection statement defined the situation if the number of chosen fingers by the user matches the randomly generated number by the Virtual player. In this case if both players chose the same number of fingers, none of the players will be awarded by the extra points.

1. **Array size: gameHistory = GameResult[100]**



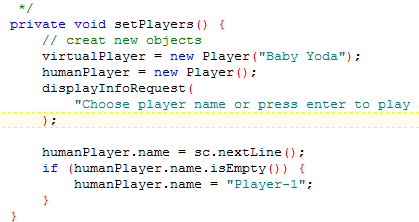
This array stores the History of the Game: it shows all rounds that’ve played by the user. The size of the array indicates how many rounds can be displayed to the user: The user can play up to 100 times.

1. **Limit of the array**



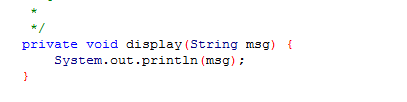
If the player will play more than 100 times, the selection statement will double the length of the array by the copyOf() method.

1. **No name provided by the user**



If the user will not provide the name at the start of the game, the custom name of “Player-1” will be provided by the selection statement. The Virtual Players name will remain “Baby Yoda”.

1. **Message Display**



This method is used to displays the text messages in the application.

**Manual Testing**

The Classes should be compiled in this order:

1. RoundResult

2. Player

3. GameResult

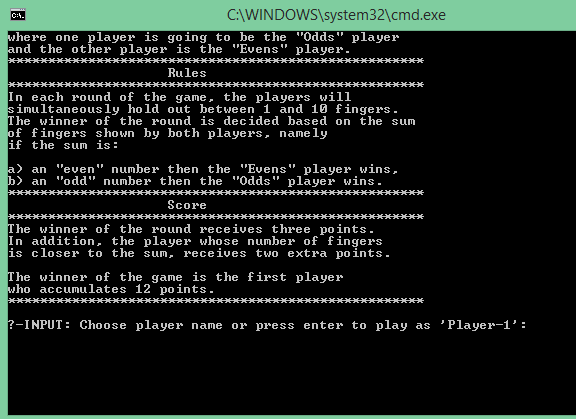
4. Game

5. GameManager

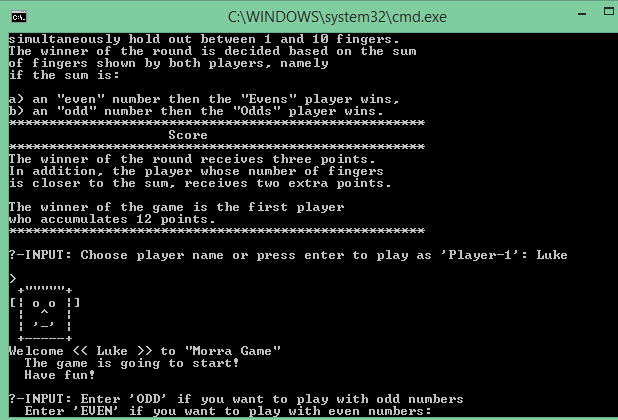
6. MorraGameApp

**Testing the application**

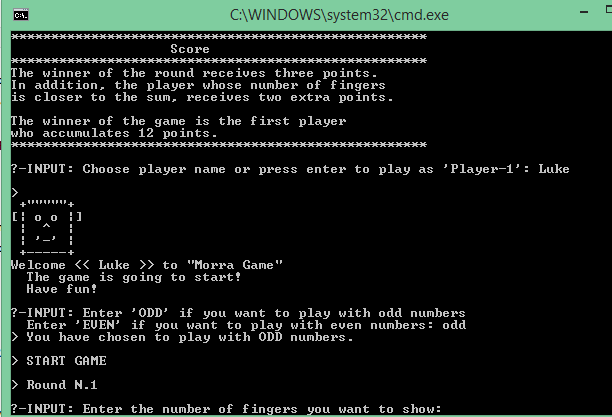
1. When the user runs the app, it is presented with the rules of the game and is asked to provide the name or chose a custom name of “Player 1”.



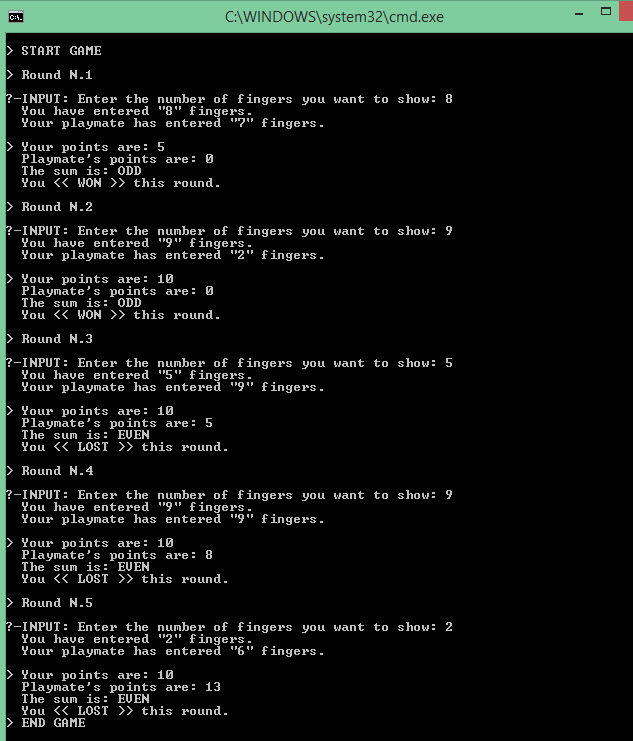
1. The user enters name “Luke”. He is presented with a welcoming message and is asked to pick the likelihood winning type (input: Odd or Even).



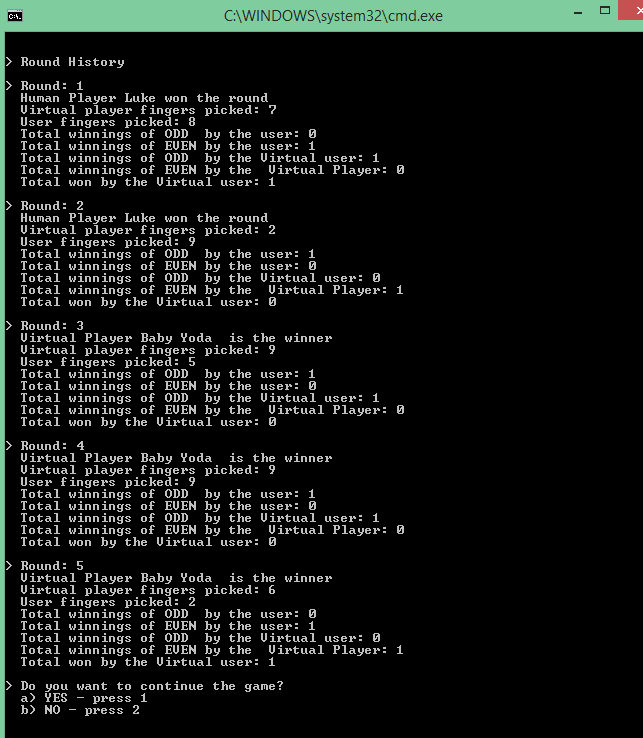
1. Luke choses ODD and is asked to enter the number of fingers.



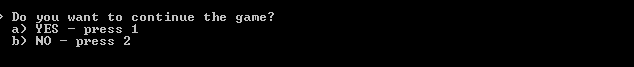
1. The application displays the outcome and asks to provide the number of fingers until one of the players collects 12 points. This game was unfortunate for Luke. He collected 10 points, but his opponent won by the total sum of 13 points.



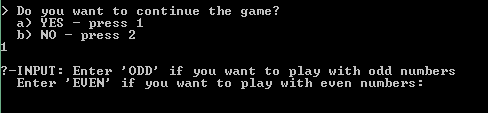
1. After the game, the History of the Round is displayed. It provides the statistical information about the game.



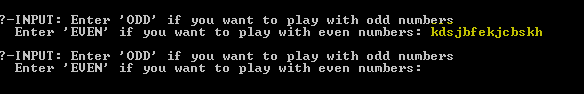
1. Luke is asked if he wants to continue the game. He does, so he provides the input of 1.



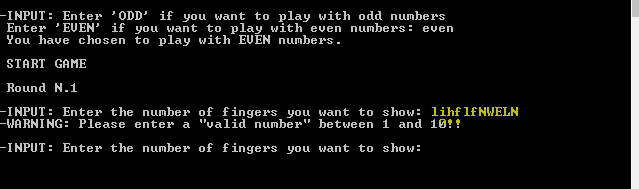
1. Luke is asked to pick ODD or Even



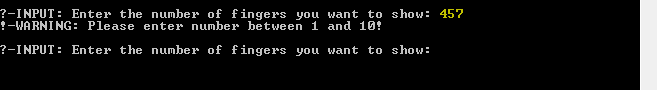
1. This time Luke accidentally presses the random strokes on the keyboard. The Applications rejects his input by asking him the same question till he provides the correct input.



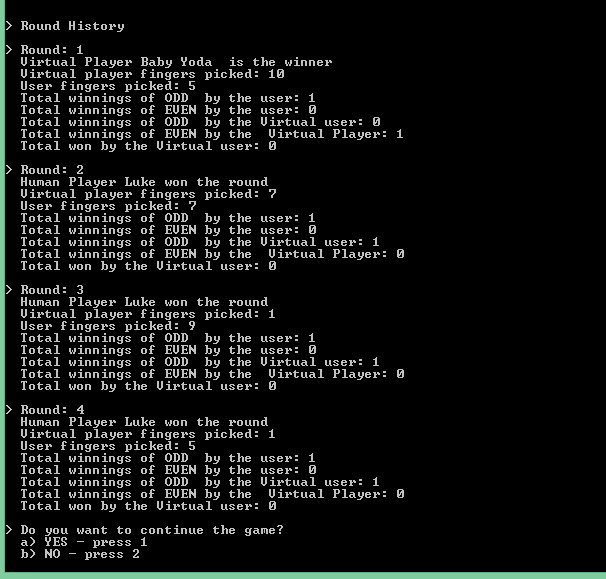
1. Luke choses EVEN and the application asks him to provide the number of fingers. This time he accidentally presses random strokes again and the applications asks him to enter a valid number until the correct input is provided.



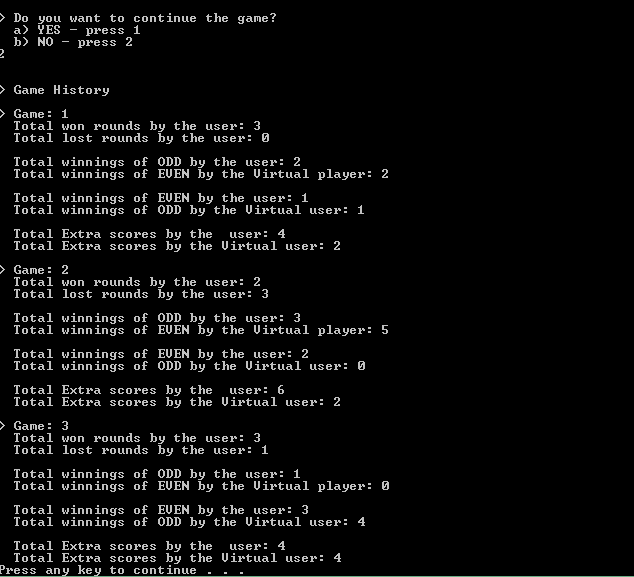
1. This time Luke enters number 457. The application asks him to enter a number between 1 and 10 and asks the same question until the valid input is provided.



1. Luke provided a correct number of fingers and played one more game. After each round, the History of the Round was displayed with the statistical data of the game.



1. Luke played few more times until he got bored. After each game the application asked him if he wants to continue to play. This time he choses NO by providing the input of a number 2. Instantly the Game History was displayed, informing Luke about the input and outcome of each game.



**Conclusion**

The process of the PBL project’s implementation was successful resulting the smooth and efficient development of the final product. The participating students were engaged in the process of working as a team and in the independent chores on their own tasks. The workload was divided during the first meeting. Several meetings took a place over the course of a month in which the team members shared the updates on their progress, consulted on the solutions of the specific tasks and overviewed the process of the project. All members engaged in the PBL project, delivered solutions of the individual parts of the project and by doing so, gained new professional skills, deepened their knowledge and increased expertise in the Java programing.