PBL Project Morra Odds and Evens Game

Software Development



***Lecturer:***

*Adriana E. Chis*

***Group F:***

[*Niall Deely*](https://app.slack.com/team/UTG4XCMK9)

*George Tudor Pascu*

*Alexandre Zurcher*

**“Morra Odds and Evens” game rules**

1. Two-player game. One player is team Odds and the other is team Evens.
2. In each round of the game, the players will simultaneously hold out between 1 and 10 ﬁngers.
3. The winner of the round is decided based on the sum of ﬁngers shown by both players. Namely, if the sum is an even number then the “Evens” player wins, otherwise if the sum is an odd number then the “Odds” player wins.
4. The winner of the round receive 3 points.
5. In addition, the player whose number of fingers is closer to the sum receives 2 bonus points.
6. The winner of the game is the ﬁrst player to accumulate 12 or more points.

**Application development of Morra Odds and Evens**

1. The application displays a welcome message in the “input screen” as the first thing the user will be able to see when running the program.
2. Under this, the user is presented with a numbered options menu with three options; option “1” will display a game description to help the user to understand the game, option “2” will start the game, and option “3” will stop the application.
3. At the beginning of each game the user will be prompted to choose whether he/she would like to be either the “Odds” or the “Even” player.
4. In each round of the game, the user must decide the number of ﬁngers to show (i.e. between 1 and 10).
5. Similarly, in each round of the game the computer will randomly pick one number between 1 and 10.
6. In each round, the game displays the numbers chosen by the player and computer.
7. After each round the game displays the number of points each player has, and whether the user or the computer won the round.
8. The game finishes when one of the players has accumulated at least 12 points.
9. At the end of a game, the game displays who the winner is, and a history of the numbers of ﬁngers shown by both the user and the computer per round.
10. Once a game has ﬁnished the application asks the player if he/she would like to play another game.
11. At the end of all games, a history of the games played is displayed. The history shows, for each game, the number of rounds won and lost by each player, how many even and odd numbers have been chosen by each player, and the extra points received by each player per game. The player is then returned to the initial menu.
12. If the user wishes to leave the game, they will need to select the option “3” (exit) in the menu and the program will be terminated.

**Project deliverables**

1. **The project submission should include a project report outlining very clearly who has coded which part of the code**

Some decisions were took working together literally, for instance in the *MorraGame class* as the application core and rules were well understood by each team member we coded, each of us, the general variables, the MorraGame() constructor, and the playRound() method as being really crucial for the game work.

We can tell that the Team work has been continuous and cordial and that sometimes even important than the main code. We had a really good time working together.

The code has been segmented in parts:

Alexandre Zurcher created the gameEnd() and playRound() methods in the *MorraGame* class. He was also responsible for the UML class diagram and parts of the Submission documentation.

Tudor George Pascu worked in the Initial Menu Loop and the runMainLoop() in *the MorraGameManager* class, and the chooseTeam() method in the *MorraGame class*, as well as realizing some of the comments and the full program methods and classes descriptions.

Niall Deely created the constructors for the classes, the playGame() and printGamesSummary() in *the MorraGameManager* class, and also the calcScore() and toString() methods in the *MorraGame* class.

Each contributor analysed, revised and tested their own code, and the final integrated program was tested by all contributors.

1. **Description of the input, main processing and output (IPO)**

**Inputs**:

Outside the game:

The user must input his option in the Menu (“1” for game description/”2” to play game/”3” to quit game).

Inside the main Morra game:

The first input is the answer if the user wants to play the game (“y” for “yes”/”n” for “no”).

The second input is the team chosen (numbers: 0 for evens, 1 for odds).

The third inputs are the player moves for the respective rounds (numbers from 1 - 10).

After the first game the software prompts the user as to whether they would like to play another game. If they do, the second and thirds inputs described above are repeated. If they don’t, the summary will be displayed and they will be returned to the menu.

**Process:**

The first batch process is to compute/check who was the winner/loser, who get the bonus and calculate the score for player and computer on each respective round.

The second batch process is to compute/check who was the winner/loser of the overall game based on the overall score.

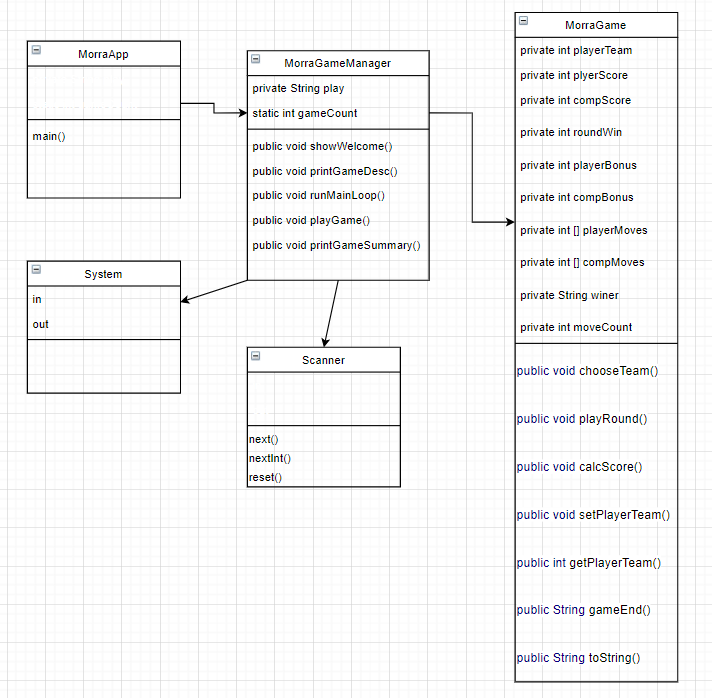
**Outputs:**

The first batch of outputs appears in the end of the first round and are the messages showing the numbers (moves) chosen by the user and the computer in the round, who was the winner/loser in the round, and the total score for the user and computer in the end of the round.

The second batch of outputs appears at the end of each game and shows who won the game and the moves chosen by the user and the computer for per round.

The third batch of outputs occurs with a game summary when the user has indicated they do not want to continue playing where, for each game played, the console displays the number of rounds won and lost by the player, the total number of even and odd numbers chosen by each player, and the total bonus points accumulated by each player during the game.

1. **Class diagram for your application**



1. **Decisions we take in designing and implementing the project**

**Three classes**

It was ultimately decided that this project would be divided into 3 classes; MorraApp, MorraGameManager, MorraGame.

**MorraGame:** Controls all functionality for a single game of Morra and stores all relevant information (teams, moves, round wins, game winner, etc).

**MorraGameManager:** Controls the functionality for interaction with the user about playing a game/games, storing the games played, and printing out the game summary of each played when the user has indicated they’re done playing. Displays also an extra user friendly Entry menu and dialogues.

**MorraApp:** Contains the main method, which simply runs an instance of MorraGameManager, and could easily be rewritten to run a single MorraGame instance if someone wants to just play one game.

**MorraGameManager**

We have tried to improve user experience by implementing a small welcome mask and a menu, with 3 options. The first one displays a small description of the Morra game. The second option leads the user to play the game, and the third option to exits the program.

As ArrayLists could not be used, an array of size 10 was chosen to store games. If the user exceeds this, they are asked if they would like to continue playing. If they do, an array with space for 10 more games is created, the previous games are copied to it, and the original array is overwritten by the new one.

**MorraGame**

Player and CPU moves were stored in an array of size 6, because there is no possible scenario where more than 6 moves can be played in a single game. A game requires a minimum of 3 moves, and a maximum of 6 for one of the players to reach or exceed 12 points.

The toString method was used in the MorraGame class to print the summary of a game in a string, allowing the MorraGameManager class to print each class to show the summary when the user is finished playing, rather than relying on printing a method for each class.

Grammar fixes were used in the toString method of MorraGame to prevent grammar mistakes such as “1 even numbers” instead of “1 even number”.

**MorraGame/MorraGameManager**

Line breaks and formatting (eg. “=== GAME 1 ===”) were added to improve console readability in certain sections.

1. **Detailed game description**

**MorraApp.java**

**(Class) public class MorraApp {}**: Contains the **main method**, that does not return any value as it has the void parameter and this one calls the collection new **MorraGameManager{}**, which has more constructors that control the game development.

**MorraGameManager.java**

**(Class) public class MorraGameManager {}:** This class as described in its name, deals with the games management. Contains **the scanner** class that takes the input from the user by creating a new object of type Scanner and the saving the input in a variable respecting the type of the variable, will present different code with the method next() or nextInt(), etc. Declares the **int** **counter variable** that counts the number of games**,** and the variable of type **String play** == ” ”; that controls the game flow saving in the empty String answers from the user and using them with meanings, by associations with terms like break, continue game etc. This last two in game variables have a **private** modifier because they are going to be accessible only inside the method. Also contains all the constructors that operate the game and the variables.

After declaring the variables, **The Morra Game [] games array** is called with an assigned space of [10].

**public MorraGameManager() {}:** Constructor declared that takes the inputs from the user and assigns them to an option selection in the game mask, or game menu. This Menu contains 3 options: 1: “Show game description”; 2:”Play” ; 3”Exit” ;

It uses two variables, one **boolean active= true;** and an **int option**: with default value = 0 and an error handler variable type **Boolean iError = true;** by default

Thanks to two while loops checks the **active Boolean** and the **Boolean iError** relating them to the Input in an **If selection statement** that encloses a switch loop.

The loop combination with the selection statement with switch cases loop works as follows: Each option in the menu is a switch case that is compared with the user input. If the input is out of the input conditions stated in the if selection statement, The jError will be launched with different messages, one for numbers out of range, and one for characters used instead of numbers. If these errors are presented, the outer while loop with parameter Boolean (active) will still be set to true.

In the case, that one of the options for the menu is correctly selected with an user input. The case that this user corresponds to will be executed running the corresponding method to it assigned and breaking the loop and running the related method. Only in the switch case number 3 the switch case will run into changing the Boolean active to false, breaking the loops and exiting the program.

**public void showWelcome() {}:** Method that prints a small header with a welcome message; Executed automatically when lunching the java program. No return as is a void type.

**public void printGameDesc() {}:** This method is launched when in the Main menu the option 1 will be selected: and it displays a bunch of prints statements that contain a small description of the game. No return as is a void type.

**public void runMainLoop() {}**: This method assures the repetition of the game as much as the user desires. The method with return type void, or no return value expected. Runs a loop type while the **String play** is different to “n” that contains a message to the user **asking him if he want to play a /another game**? Depending if it´s the first game or if he had played before. And relates this question to an if selection statement inside the while loop that takes the user input if this is among the listed, and if not printing an error message. If the answer **is yes** then we proceed in executing the **PlayGame() {} method**. If not, the Game Summary will be printed if this summary exists, and implicitly a past game was played.

**public void playGame() {}**: This method is connected directly to the **MorraGame [] array** that initially has 10 empty spaces, each one is going to be related to a Morra Game played and stored in the array. The loop contains two if selections statements, being a control loop. This control is launched when the limit of the array will be exceeded and the user has finished the tenth game in the array. Then a question will be printed asking the user if he feels ok after playing for a long while the same elementary game... If the user wants to continue, then a New array will be added to the original one **MorraGame[] games2 = new MorraGame[this.games.length + 10];** and the game will continue for another 10 games and the same sentinel process will be repeated again.

**public void printGamesSummary(){}**: This method prints the results saved in the **array MorraGame game = new MorraGame();** withhelp of the variable **gameCount** that iterated through the game and saved the games played game into the array.

**MorraGame.java**

**(Class) public class MorraGame {}:** This method as its name says represent the main functionality of the app.

At the beginning of the method we declare **The scanner** method that takes the input from the user by creating a new object of type Scanner and saving the input in a variable respecting the different type of the variable, this will present code with the different construct next(), nextInt(), etc.

Also, these variables are private and will all be used along the different code methods.

private int playerTeam; //private int playerScore = 0, compScore = 0; // private int roundWin = 0;//private int playerBonus = 0, compBonus = 0; // private int[] playerMoves = new int[6]; (no more than 6 moves logically required to reach the target number)// private int[] compMoves = new int[6]; // private String winner = ""; // private int moveCount = 0;

**public MorraGame() {}:** This constructor represents the main core of the game, and with this I mean the main rules of the game. There is a while loop that runs while the **score** (playerScore or compScore ) of each player (player & computer) is not equal to or higher than 12, calls the **PlayRound() constructor** and increments the count variable. After doing that **if** the score reached or exceeded 12, compares the computer score and the player score with the help of an **if selection statement and** depending on the result will display who is the winner or if is a draw. And then prints the results and ends the game.

**public void chooseTeam() {}:** In this constructor of type void(no value return needed) ,the player will chose the team in which he will play “odds” or “evens”. With the help of **a variable team initialized with 0 of type int.**  And **an error variable of type Boolean** starting with the value of true**.** Here is used a **while loop** that checks this Boolean as parameter and decides the user input with an **if selection statement**. Deciding whether the input was an odd number (1) or an even number (0). If these conditions are met, then the team variable will be assigned to the playerTeam and prints the corresponding team to the user. If not, The **error Boolean** will not be changed and the user will receive an error message. The sc.reset() resets the values inserted in the sc variable that intakes the values from the user with the help of the Scanner function.

**public void playRound() {}** In this constructor of type void(no value return needed) , is where all the fun takes place, or all the interaction between “users”, players occurs. Here the **int playerMove= 11** is declared with this value as is not allowed to show this number in the round and is physically impossible in the real game. Also there is a while loop with an error Boolean variable set by default to true, as only if the correct values are introduced by the user this **Boolean error** sentinel will be changed to false allowing the user value to be saved into the **playerMove** variable. At the same time to increase the moveCount counter variable. If the wrong values are presented, then again the error will be displayed, different error depending on the type of incorrect input. The compMove variable will be assigned a random value by using the int compMove = (int)(Math.random() \* 10 + 1); **the Math.Random** function that gives a random value as the value that the computer displays in each round, and of course this random value is limited by the same constrains that the player has in **the if selection statements**. At the end the **calcScore()** method is called.

**public void calcScore(int playerMove, int compMove) {}:** This method makes the mathematical part of the game calculating the moves of each player following the rules specified in different constrains along with **if/else if selection along** the code, that is why is dealing this two parameters playerMove, compMove from the constructor declaration. Here we decided to calculate the points in two different ways as per required criteria, one for round points, and one for bonus points. For both players we used **if / else if selection** statements. For the player we **compare the playerteam value and the total sum variables and we apply to both of them the modulo %2, if(playerTeam%2 == sum%2**). If the result is equal, that means the player team and the sum of the scores are both even or odd, and the player has won the round so we add +3 points and add the count to the **roundWin++ variable**. If the case is the contrary, then the computer is the one who won and the point would be assigned to him compScore += 3.

To calculate the bonus points we have **compared the difference of the sum variable (of the numbers for each team) and the player/compMove)**  to use this comparation we have used the **Math.abs** function **if(Math.abs(sum - compMove) > Math.abs(sum - playerMove**)) , and depending on which player has a bigger number that means their number was closest to the sum. If the numbers are equal, no bonus points are awarded.

Then both bonus points (if any are won in the round) and Scores are displayed for both players respectively.

**public void setPlayerTeam(int team) {}:** Sets the player team values without any return as is a void, as usual in the setter methods

**public int getPlayerTeam() {}:** Gets the value from the player team and assigns it to the team variable.

**public String gameEnd() {}:** In this method we have moved all the variables to a variable **String moveSummary** that prints the winner, and then goes through the playerMoves/compMoves arrays and summarize the data that was obtained though playing the game and saving the values in it obtained. This method then returns the **MoveSummary** string.

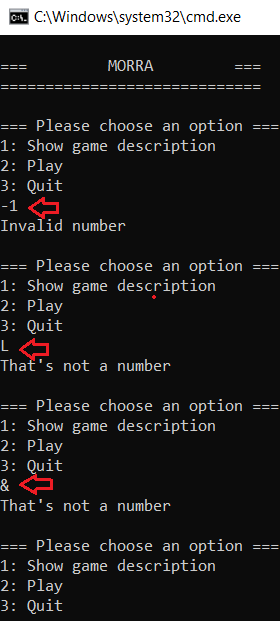
**public String toString() {}:** This is a method created to display the game in a grammatically correct String format **gameString**. It shows the number of rounds won and lost, the number of odd and even numbers chosen by the player and computer during the game, and the total bonus points received by each player. **if/else** statements are used to fix the grammar and make the word “numbers” singular when it refers to a number of “1”, and plural if it refers to any other number.

At the end of the method the **gameString String variable** that stored the game information will be returned.

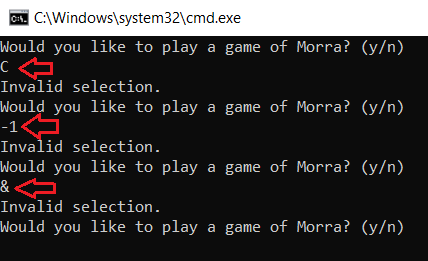
1. **Testing**

Extensive testing was done by all team members to try to find bugs in the code.

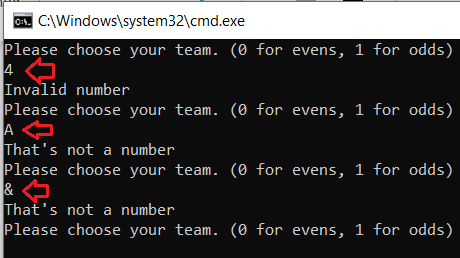
On the first prompt, if a user chooses a number different than expected or a non numeric character, the program says that it is an “Invalid Number” or “That’s not a number” and prompts the user again.



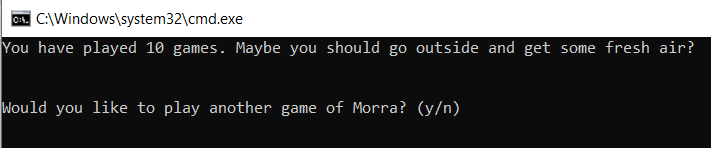
On the second prompt, if the user inputs something different from “y”/”n” (or a String starting with “y”/”n”), it will return “invalid selection” and prompt the user again.



On the third prompt, if the user inputs something different from (0/1) it returns “invalid selection” and prompts the user again.



After play 10 games the program the user if he wants to get some air and ask the user if he wants to play once more.



Bugs found and fixed:

Some bugs were found as the “Invalid Number” message wasn’t displayed because of the iError Boolean implementation in the loop, the solution that we found is to use exactly this Boolean inside the while loop and the result is positive because now the error is correctly displayed if the user input is out of the pointed range accepted in the if selection statement.

The code would crash when the scanner was expecting a nextInt(), but received a String. To fix this, we implemented hasNextInt() to check for an int in advance, and looped the code until a valid int was input.

The toString method of MorraGame would output incorrect grammar where the number was 1, such as “You chose 1 odd numbers, and …”. To fix this, the code was changed so that numbers were checked beforehand, and in cases where the number was 1, the word “number” was used instead of “numbers”.