Ferid Gökkaya - 1250370
Tho Phan Chi - 1403884
Thanh Le Hoang Kim - 1403839
Kim Tran Hoang - 1403509
Ngoc Pham Nhu - 1403648

Nanu

– Java Project Application –

February 12, 2023

Springer Nature

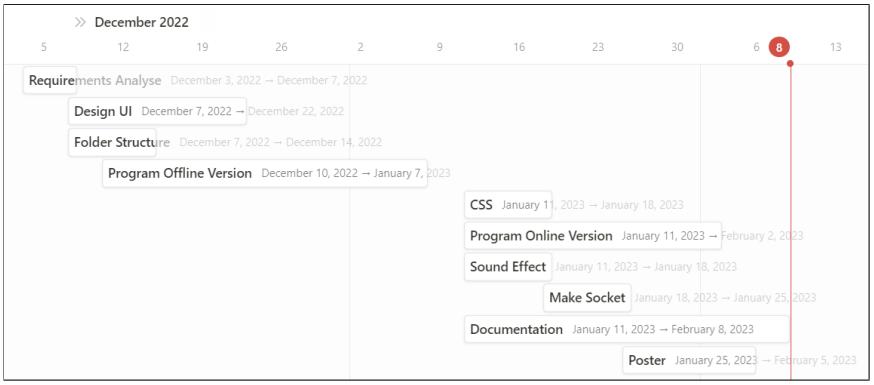
Contents

Chapter 1
Team Work

Our team members and the matriculation numbers.

- Ferid Gökkaya - 1250370
- Tho Phan Chi - 1403884
- Thanh Le Hoang Kim - 1403839
- Kim Tran Hoang - 1403509
- Ngoc Pham Nhu - 1403648.

1.1 Work structure



1.2 Who does what?

Name	Assign	Date
Requirements Analyse	Ferid, Kim Thanh, Ngoc Pham Nhu, Tho Phan, Kim Tran Hoang	December 3, 2022 → December 7, 2022
Use case diagram	Ngoc Pham Nhu	
Flow chart	Ngoc Pham Nhu, Tho Phan	
Design UI	Kim Thanh, Ferid, Kim Tran Hoang, Ngoc Pham Nhu	December 7, 2022 → December 22, 2022
Design on Basamiq	Kim Thanh, Ferid, Kim Tran Hoang	
Design on Figma	Ngoc Pham Nhu, Kim Tran Hoang	
Program Offline Version	Kim Thanh, Tho Phan, Kim Tran Hoang, Ferid, Ngoc Pham Nhu	December 10, 2022 → January 7, 2023
Sound Effect	Kim Thanh	January 11, 2023 → January 18, 2023
GUI	Tho Phan, Kim Thanh, Kim Tran Hoang, Ferid, Ngoc Pham Nhu	
Logic	Tho Phan, Kim Thanh	
Generate data	Ngoc Pham Nhu, Tho Phan, Kim Thanh	

Name	Assign	Date
Report	Ngoc Pham Nhu, Ferid, Kim Thanh, Tho Phan, Kim Tran Hoang	January 11, 2023 → February 8, 2023
Content for the report	Ngoc Pham Nhu, Tho Phan, Kim Thanh, Ferid, Kim Tran Hoang	
Team work	Ngoc Pham Nhu	
Introduction	Ferid	
Problem description	Kim Tran Hoang	
Related work	Ngoc Pham Nhu, Kim Thanh, Tho Phan	
Proposed Approach	Tho Phan, Kim Thanh	
Implementation Details	Tho Phan, Kim Thanh	
Conclusions and Future Works	Ferid	
Format latex	Ngoc Pham Nhu	
Program Online Version	Kim Thanh, Tho Phan	January 11, 2023 → February 2, 2023
GUI	Tho Phan, Kim Thanh	
Logic	Kim Thanh, Tho Phan	

Name	Assign	Date
CSS	Ngoc Pham Nhu, Kim Thanh, Kim Tran Hoang	January 11, 2023 → January 18, 2023
Homescreen	Ngoc Pham Nhu, Kim Thanh	
Enter profile	Kim Thanh, Kim Tran Hoang	
Guess picture	Ngoc Pham Nhu, Kim Thanh	
Leaderboard	Kim Thanh, Kim Tran Hoang	
Right answer	Ngoc Pham Nhu, Kim Thanh	
Wrong answer	Kim Thanh	

Name	Assign	Date
Poster	Ngoc Pham Nhu, Ferid, Kim Thanh, Tho Phan, Kim Tran Hoang	January 25, 2023 → February 5, 2023
Content of the poster	Kim Thanh, Tho Phan, Ferid, Ngoc Pham Nhu, Kim Tran Hoang	
Concept of the game	Kim Thanh	
Related work	Kim Thanh	
Application structure	Kim Thanh	
GUI design	Ferid	
UML diagram	Ngoc Pham Nhu	
Team work	Kim Tran Hoang	
What we have leaned	Kim Tran Hoang	
Idea for the future game	Ferid	
Online version	Tho Phan	
Design poster	Ngoc Pham Nhu, Kim Tran Hoang	
JavaDoc	Kim Thanh, Tho Phan, Kim Tran Hoang, Ferid	February 1, 2023 → February 8, 2023

Chapter 2
Introduction

2.1 General topic

This game was originally developed to strengthen and to train the memory of children. It is important that children and little persons who visit the nursery or the elementary school train their memory. This is necessary because at children age, it is very important to train the Long-term memory and Short-time memory.

Because children can realize, learn and practice much better with games and fun, this game-NANU is very suitable for children who are at the preschool age. Also, this game is quite suitable for normal gaming fun.

2.2 Similar problems in practice

This game-NANU, is a Memory game which is very popular at children who are at the Preschool or also for persons who like memory games. Such kind of games are played and are available at places where are children like at the nursery, preschool or at locations where children train and learn the performance of concrete activities and their effects on man and his environment is addressed, like occupational therapy.

Advantages:

- Children train their memory
- A good method to persuade children with playing games

Chapter 3
Problem Description

3.1 Set up the game

The administrator places 24 discs, with all pictures facing up, in a grid shape on the board game. Then the administrator places the five covers over the five discs. All players have to pay attention to this part and watch and listen to remember which pictures are where before starting the game.

3.2 Concept of the game

At the beginning of the game, the first player throws the dice. The color of the dice that the player gets will indicate the color of cover that the player will be lifting. If the dice is the joker, the participant can lift any cover. Let's pretend the player got an orange on the dice. Before lifting the cover, a list of names will appear for the player to select the guess image that is hidden beneath the cover. If the player is correct, the disc will disappear and the point will be counted. For each correct disc, the player will have one point. Then the player chooses any disc left in the grid to place the cover on. If he is incorrect, the player must leave the disc there and re-cover it before the next player throws the dice. A turn is over until the player chooses the wrong answer. The administrator displays what a player is covering and the color of the cover on the other player's screen. The game will end when there are only four discs left in the grid and there is nowhere to place the fifth cover. The player with the most points at this point on the leaderboard is the winner.

Chapter 4
Related Work

4.1 HashMap

A Java in-built collection called HashMap uses an implementation of hash table. The implementation provides constant-time performance for operations like get, which retrieves values, and put, which stores items.

The major advantage of HashMap is that they increase the effectiveness of search and deletion operations. Additionally, HashMap only makes sense when the data we want to store has unique keys.

And when there are a lot of collisions, hash is inefficient. For a large number of possible keys, hash collisions are virtually impossible to avoid. Also null values are not supported by hash and implementing hash tables can be challenging.

4.2 Singleton

A class is called as singleton if it only permits the creation of one instance of itself and gives access to it. It has static variables that could store distinct, private instances of itself. Programming languages like Java and.NET use the singleton pattern to define a global variable. A single object that is called through all systems remains constant and does not need to be defined multiple times.

There are some disadvantages of Singleton: First, the singleton design pattern's global nature causes dependency hiding. Then unit testing Singleton Pattern-based programs can be challenging. Additionally, it may result in tightly coupled code that is challenging to update. The final one is that because the object only has one instance, if it is corrupted, the entire system is at risk.

4.3 Model View Controller MVC

The model (in the singular) is what we'll refer to as the application's or domain's unchanging essence. In object-oriented terminology, this will be the set of classes that describe and support the underlying issue. As a result, these classes tend to be stable and as long-lived as the problem itself.

There will be one or more interfaces with the model for a certain condition in a specific version, which we'll refer to as the views (plural). These will be groups of classes that provide us with "windows" onto the model in object-oriented terminology. Additionally, the views need to know the existence and nature of the model.

An object called a controller gives you control over a view. To put it a little simply, the controller manages input, and the view manages output. The most gathering information on operating systems and platforms are controllers. Views are generally unaffected by the origin of their events, whether it be Microsoft Windows, X Windows, or another source.

In the same way that views are aware of their model but model are unaware of their views, controllers are aware of their views but views are unaware of their controllers.

4.4 The 60-30-10 Rule

"60% is your dominant hue, 30% is secondary color and 10% is for accent color."

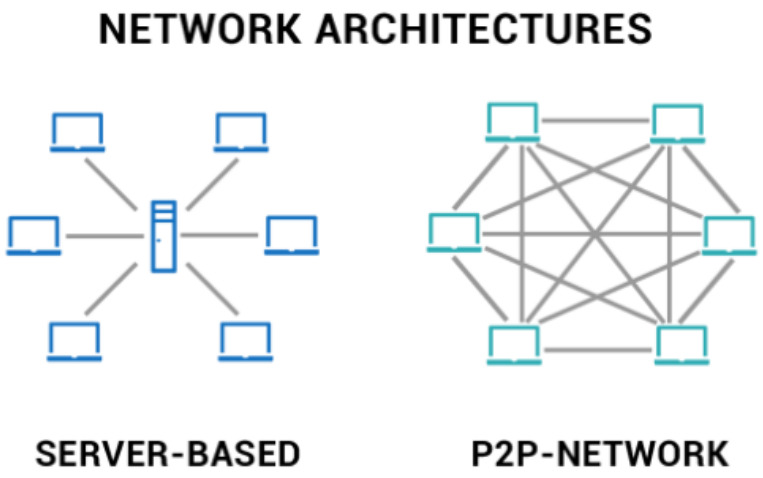
The 60-30-10 rule is defined on the line below. One of the most crucial elements in UI design is color. But how do you balance your color choices for your UI and how do you put them into practice? The answer is the 60-30-10 rule.

This rule assists with in creation of an appropriate and balanced colour combination for your design. The basic concept is to use one color solely for 60% of the palette (often a neutral color), another color for 30% of the palette (a complimentary color), and a third color (an accent color) for the remaining 10% of the design.

4.5 Client server

There are mainly two possible network model: peer-to-peer and client-server. In the peer-to-peer architecture, data is exchanged between any pair of connected players while in the client-server architecture, data is only exchanged between players and

the server.



We decided to choose the client-server architecture. The benefit of client-server structure is that it provides the data integrity between players, which means that the player can not cheat. Moreover, the client will have less workload due to the service being handled by the server.

4.6 Java socket library

Java socket library provides support for network communication using sockets. It includes classes for implementing client and server applications, handling common network protocols and working with IP addresses. The Java socket library allows developers to create network applications by providing a simple API for communicating over the network using sockets.

This library also provides transport protocols: TCP and UDP.

- TCP (Transmission Control Protocol) is a reliable connection based protocol. All data the device sends is guaranteed to arrive at the other side and in order that it sends. Because of making sure data is arrived at the receiver, TCP may cause latency compared to UDP.
- UDP (User Datagram Protocol) is connectionless protocols, instead of adding lots of features and complexity compared to TCP, it is a very thin layer over IP. This means that it is suitable for streaming, online gaming with low latency.
- However, based on the game rule and requirement, we choose TCP because this is a turn-based card game. This game requires data accuracy rather than real-

time game data between players. Moreover, the TCP is simpler to implement the multiple player game rather than UDP.

Chapter 5
Proposed Approaches

5.1 Hashmap

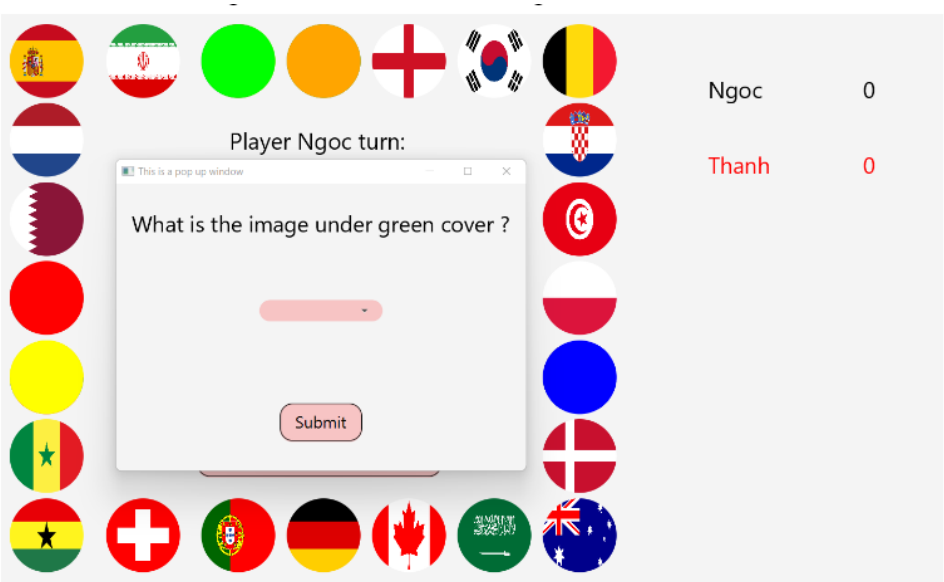
The hashmap coverHashMap is created to map each cover with the index of disc in myList. The hashmap helps us answer the question “What picture is under the [color] cover?” faster and easier. Here we find in the hashmap the value of the key color, if the key color does not have any index, we will put a new index to that key, otherwise we replace the old index with the new one.

```
public static HashMap<String, Integer> coverHashMap = new HashMap<>();
```

```
if (GameManager.coverHashMap.get(color) == null) {
    GameManager.coverHashMap.put(color, index);
} else {
    GameManager.coverHashMap.replace(color, index);
}
```

5.2 60-30-10 rule

The primary color we chose is white (#FFFFFF), the complementary color is pink (#F9A4A4), and the highlight color is black (#000000), all in accordance with the 60-30-10 rule.



Since we used several images to create the game cards, the white background will help the image on the cards appear clear and prevent viewers from being confused when looking at them.

Pink is a comforting, amusing, and sentimental color that makes people think of their childhoods. The software is made more kid-friendly by using pink as a secondary color and all of the buttons and the scroll bar will be filled with pink.

Our app's accent color is black. To make every word in the app show up against either a pink or white background, we used it.

5.3 Client-server model using Socket connection

5.3.1 Set up server:

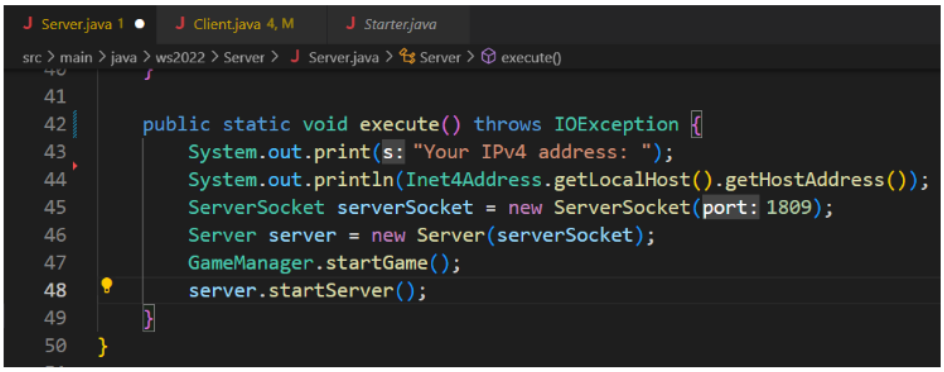
In order to create a socket connection between two machines we need to create another thread for listening and processing the message of the device..

```
18      public void startServer() {
19          try {
20              while (!serverSocket.isClosed()) {
21                  Socket socket = serverSocket.accept();
22                  ClientHandler clientHandler = new ClientHandler(socket);
23                  Thread thread = new Thread(clientHandler);
24                  thread.start();
25              }
26          } catch (IOException e) {
27              // TODO: handle exception
28          }
```

For listening thread, we can easily create by implementing the Runnable interface. In the Runnable interface we have to overwrite the run, and this method is automatic when we create a new thread. Therefore, when adding a new device i.e a new thread created, we want this thread to listen to the input of the device and process the message of that input like below.

```
19  public class ClientHandler implements Runnable {
20
21      @Override
22      public void run() {
23          String messageFromClient;
24          while (socket.isConnected()) {
25              try {
26                  messageFromClient = bufferedReader.readLine();
27                  if (messageFromClient != null) {
28                      System.out.println(messageFromClient);
29                      handleMessage(messageFromClient);
30                  }
31              } catch (Exception e) {
32                  closeEverything(socket, bufferedReader, bufferedWriter);
33                  e.printStackTrace();
34                  break;
35              }
```

Finally to create a server, we create a ServerSocket object with a port by programmers, and that port is necessary for the client to connect. Then we can use the startServer() method to listen for new devices as mentioned above. We also have to show the server IPv4 address for the client to connect the server by using the Inet4Address class.



5.3.2 Set up Client:

Same as the server we also need to create another thread for listening and processing the message from the server. Since the method to create a thread is simple, we use an anonymous class implementing Runnable interface.

To connect the server, we simply pass the IPV4 of the server as mentioned, and the port that has been decided previously in the Socket class.

5.4 Singleton

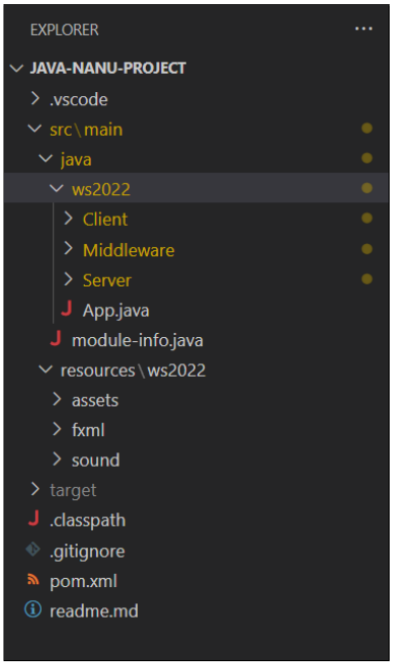
This is a simple implementation for a singleton class. This helps to provide a global access point to that instance, avoiding creating the unnecessary object multiple times in application.

In the JavaFX library, the controller will be created when its corresponding fxml file is runned. Because the implementation of the singleton pattern, we cannot create the singleton object directly using the *new method*. Therefore, we have to inject the class by using *setController()* method when we load the fxml file like the picture below.

For example, when we want to update the game by using GameManager Class when the user gets the right answer, we just simply get the object by using the static method getInstance of the class.

Chapter 6
Implementation Details

6.1 Folder structure

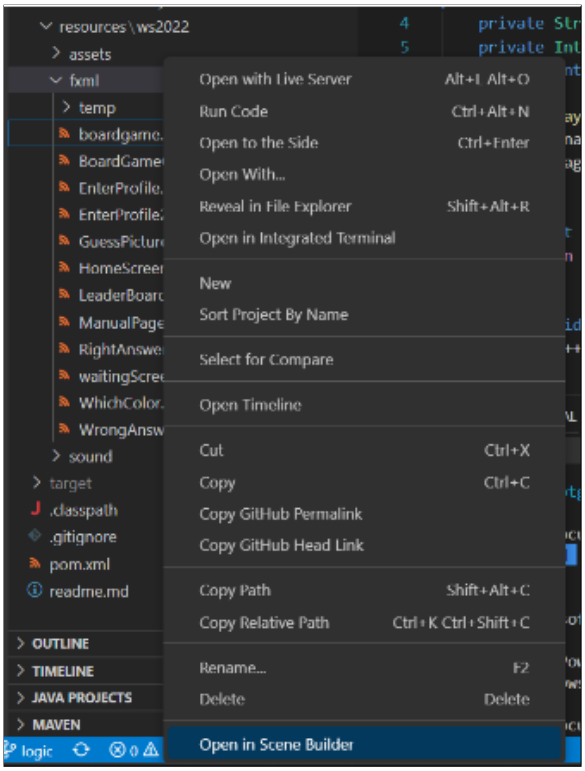


- src/main: main folder, this is the part where we put all our code.
- .vscode: Workspace settings as well as debugging and task configurations are stored at the root in a .vscode folder.
  - java/ws2022: This folder contains 3 more subfolders: client, middleware, and server and one file app.java.
    - Client: this folder contains the code for control the logic (offline game) and also the code for the client side (online game).

    - Model: contains class structure of the game, directly manages data, logic, and rules of the application.
    - View: contains the UI of the game.
    - Controller: contains of java file controlling fxml file.
  - Middleware: the API is the translation message between the client and server (online game).
  - Server: it contains the logic for set up the server to connect between multiple machines (online game).
- resources/ws2022: This folder keeps all the resources, e.g. sound, image for our games. In this folder, there are 3 subfolders: assets (keeps the image), sound (keeps sound files for sound effect), fxml (keeps all the file with the extension .fxml for the UI).
- target: The target folder is the maven default output folder. When a project is build or packaged, all the content of the sources, resources and web files will be put inside of it, it will be used for construct the artifacts and for run tests.
- .classpath: The .classpath maintains the project's source and target references for Java compilation and compressed file or project dependencies.
- .gitignore: When working with github to share our code, we don't want to push some unnecessary files to our repository, e.g. .class, .vscode so we add some patterns to this file so github will ignore these files.
- pom.xml: The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.
- readme.md: The What, Why, and How of the project are addressed in the readme.md file. Visitors to the repository will see the README automatically from GitHub.

6.2 GUI

We used the JavaFX library to create the scene. The JavaFX library also has the built-in SceneBuilder, which is very convenient in managing the skeleton, controller and stylesheets. We downloaded the Scene Builder and the extension in VSCode, then we configured the executable path so that the fxml file can be launched immediately from the VSCode. In the Scene Builder application we can drag and drop the elements (e.g. button, menu, label, text . . .) to design JavaFX application user interfaces. Then we add stylesheets to style and layout the game just like the Figma design we have done. Finally, the controller will be connected to the fxml file to make the application more interactive.



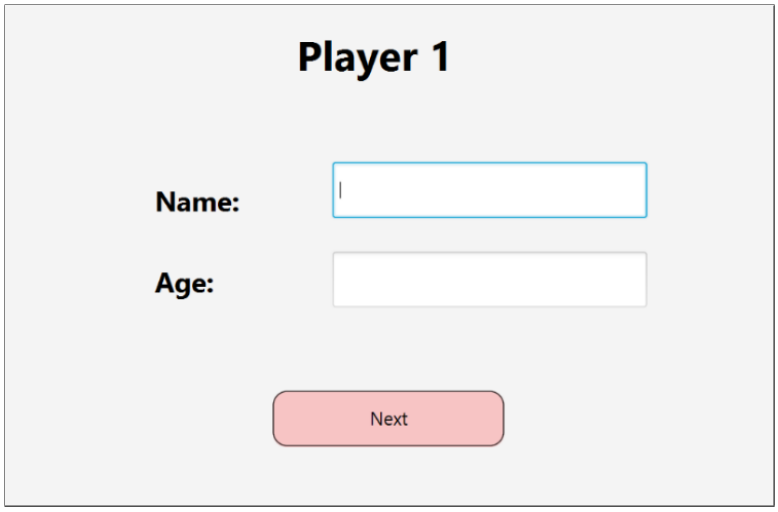
6.3 Logic for offline game

6.3.1 Start application

To start the Nanu offline game, in the home screen the player can click to the “Play offline” button. Then the user will be redirected to the “EnterProfile” scene.

6.3.1.1 Enter profile

We decided that the offline version of Nanu will have 2 players. So the players will enter their name and age respectively. The player 1 will enter their name and their age then when they click the “Next” button the player 2 also enter their name and age.



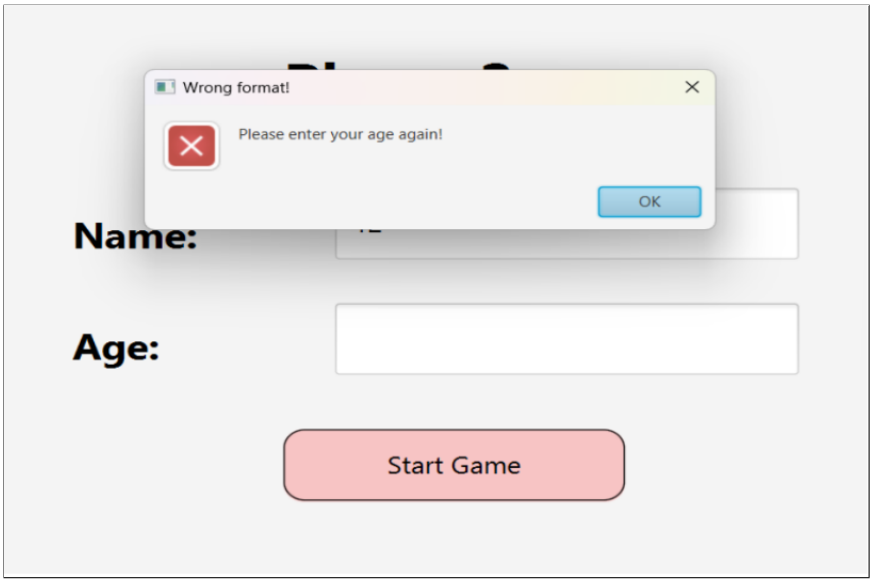
6.3.1.2 Validate input

During the input phase, there are some scenarios where the player input unexpected value. For example, the players enter their age as a string not a number. Also when the player input their age too large or a negative number, we try to detect errors and

alert it to the player and ask them to enter another value. Also, because in our games there are multiple players, in the game there will be text like “There is turn of player [name].” So the players need to have different names, so when the players have the same name we also alert the error.

```
public static void validateValue(String name, String age) {
    SceneController sceneController = SceneController.getInstance();
    if (name.isEmpty()) {
        sceneController.showAlertMessage(Alert.AlertType.ERROR, title: "Name Required!",
                message: "Please enter your name");
        return;
    }
    if (age == null) {
        sceneController.showAlertMessage(Alert.AlertType.ERROR, title: "Age Required!",
                message: "Please enter your age");
        return;
    }
    if (!age.matches(regex: "\\d+")) {
        sceneController.showAlertMessage(Alert.AlertType.ERROR, title: "Wrong format!",
                message: "Please enter your age again!");
        return;
    }
}
```

```
public void enterGame(ActionEvent event) throws IOException {
    // in second enter profile (before go to game)
    soundc.click();
    String name = nameTF.getText();
    String age = ageTF.getText();
    GameManager.validateValue(name, age);
    if (GameManager.PLAYER1.getName().equals(name)) {
        sceneController.showAlertMessage(Alert.AlertType.ERROR, title: "Same name!",
                message: "You have the same name as the player 1. Please enter another name!");
        return;
    }
    GameManager.PLAYER2 = new Player(name, Integer.parseInt(age));

    sceneController.enterGame(event);
}
```



6.3.1.3 Choose first turn

The traditional Nanu game rules that the youngest player will play the first turn. So to choose the youngest player, we take the input age and compare which is the smaller number and set the first turn as this player. If the players have similar ages, the player who enters their name and age first will play the first turn.

```
public static void getFirstTurn() {
    if (GameManager.PLAYER1.getAge() > GameManager.PLAYER2.getAge()) {
        GameManager.isPlayer1Turn = false;
        return;
    }
    GameManager.isPlayer1Turn = true;
}
```

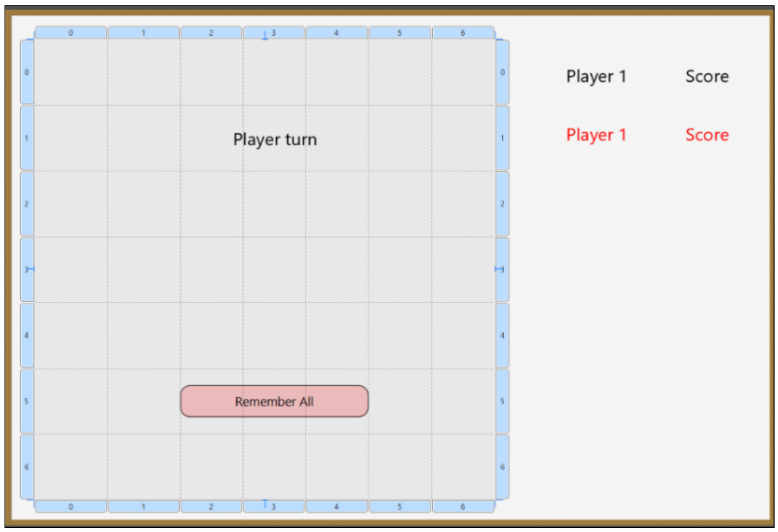
When the player 1 finishes their turn, the game will automatically change the turn to the other player.

```
public static void changeTurn() {
    if (isOnline) {
        return;
    }
    isPlayer1Turn = !isPlayer1Turn; // flip turn
}
```

6.3.2 Set up game

6.3.2.1 Set up skeleton

In the boardgame, we set up first in fxml the basic structure. Here we have one 7x7 gridpane to put the images, but we only put the images on the first row, first column, last row and last column. We intentionally leave the blank space inside the gridpane for the popup window later, so that it does not hide the images. On the right side, we also have the names of the players and their scores. On the inside of gridpane, we also have the label announcing whose turn and the button position to control the flow of the game.



6.3.2.2 Display image and players' names and scores

First we will have all the pictures' names in one List and we use Collections.Shuffle to shuffle the data so that we get a new order of pictures randomly.

```
public static void startGame() {
    if (isOnline) {
        return;
    }
    GenerateData.generateDisc(myList);
    Collections.shuffle(myList);
}
```

Here we set up the board game by getting the pictures from the theme folder and then set width, height and circle shape for the pictures. We run 2 for loops to set the images in the gridpane, if the index of the gridpane cells is not in the first and last row or first and last column, we will not set the images in that cell.

```
for (int y = 0; y < 7; y++) {
    for (int x = 0; x < 7; x++) {
        if (x != 0 && y != 0 && x != 6 && y != 6)
            continue;
        String selectedImage = "/ws2022/assets/FootballTheme/"
                + GameManager.myList.get(index).getImage();
        Image image = new Image(this.getClass()
                .getResource(selectedImage)
                .toExternalForm());
        Circle clip = new Circle(centerX: 50, centerY: 50, radius: 45);
        imageView = new ImageView(image);

        // populate matrix
        imageView.setFitWidth(value: 100);
        imageView.setFitHeight(value: 100);
        imageView.setClip(clip);
        imageView.setOnMouseClicked(event -> changeDisc(event));
        imageView.setUserData(new Coordinate(x, y)); // index data
        boardgame.add(imageView, x, y);
        index++;
    }
}
```

Also we set the names and the score of the player on the left side of the scene.

```
player1.setText(GameManager.PLAYER1.getName());
player1Score.setText("" + GameManager.PLAYER1.getScore());
player2.setText(GameManager.PLAYER2.getName());
player2Score.setText("" + GameManager.PLAYER1.getScore());
```

6.3.2.3 Set up cover, set turn and display "Roll dice" button

When the players click the "Remember all" button, the game will set covers in 5 random images, delete the "Remember all" button and replace it with the "Roll dice" button.

```
public static Coordinate[] setUpCover() {
    Coordinate[] coordinates = new Coordinate[5];
    int count = 0;
    while (count < 5) {
        Random random = new Random();
        int x = random.nextInt(bound: 7);
        int y = random.nextInt(bound: 7);
        if (x != 0 && y != 0 && x != 6 && y != 6)
            continue;
        int indexPane = y * 7 + x;
        int indexList = indexPane - (indexPane - 1) / 7 * 5;

        if (GameManager.myList.get(indexList).checkCover())
            continue;
        GameManager.myList.get(indexList).setCover();
        coordinates[count] = new Coordinate(x, y);
        count++;
    }
    return coordinates;
}
```

The function setUpCover will return the one array of Coordinates and set the cover in 5 different images. First the game will random the x index and y index of gridpane

cells and they set their covers in those cells to replace the pictures. If the cell has already been covered, this function will move on and find another cell. When the cell is covered successfully, the x, y index will be saved in the coordinates array. This array will be used later to reference from the data lists to the index of gridpane and validate the answers.
The game also announces who plays first so that the two players can know their turn.

```
public void setTurn(boolean isPlayer1Turn) {
    status.setVisible(value: true);
    if (isPlayer1Turn) {
        status.setText("Player " + GameManager.PLAYER1.getName() + " turn: ");
        return;
    }
    status.setText("Player " + GameManager.PLAYER2.getName() + " turn: ");
}
```

The createRollDiceBtn() will set the dice image in the Imageview above the roll dice button, then we set the height, width for roll dice. We also set that when we click the button, the event will trigger the function clickRollDice().

```
public void createRollDiceBtn() {
    Image diceImage = new Image(this.getClass()
            .getResource(name: "/ws2022/assets/Dice/dice.png")
            .toExternalForm());
    dice.setImage(diceImage);
    dice.setFitWidth(value: 100);
    dice.setFitHeight(value: 100);
    if (guessPicture != null) {
        boardgame.getChildren().remove(guessPicture);
    }
    rollDice.setPrefHeight(value: 50);
    rollDice.setPrefWidth(value: 297);
    rollDice.setOnAction(event -> {
        try {
            clickRollDice();

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    });
    boardgame.setColumnIndex(rollDice, value: 2);
    boardgame.setRowIndex(rollDice, value: 5);
    boardgame.setColumnSpan(rollDice, value: 3);
    boardgame.getChildren().add(rollDice);
}
```

Fig. 6.1 Create Roll Dice source code

Fig. 6.2 Roll dice GUI

6.3.3 Start game

6.3.3.1 Roll dice

When clicking on the button “Roll dice”, we will maybe get a normal answer from the 5 colors: “red”, “blue”, “yellow”, “green”, “orange” or we get the joker which we can choose any color.

```
public void clickRollDice() throws IOException {
    soundc.click();
    GameManager.COLOR = Dice.rollDice();

    if (GameManager.COLOR.equals(anObject: "joker")) {
        soundc.joker();
        getJoker();
    } else {
        getNormalColor();
    }

}
```

If we get a normal answer, on the imageView above the button will be replaced the dice image with the color cover result from roll dice. We also delete the “Roll Dice” button and add the “guess picture” button, set width, height for the button just like other buttons.

```
public void getNormalColor() {
    // display cover has been generated by roll dice
    String coverImage = "/ws2022/assets/Covers/" + GameManager.COLOR + ".png";
    Image cover = new Image(this.getClass()
            .getResource(coverImage)
            .toExternalForm());
    dice.setImage(cover);
    dice.setFitWidth(value: 100);
    dice.setFitHeight(value: 100);
    if (chooseColor != null) {
        boardgame.getChildren().remove(chooseColor);
    }
    // remove roll Dice button
    boardgame.getChildren().remove(rollDice);

    // add guess Picture button
    guessPicture.setPrefWidth(value: 297);
    guessPicture.setPrefHeight(value: 50);
```

6.3.3.2 Joker card

If we get the joker card, the Which Color popup will appear. Then the player will choose the color they want and submit it.

```
public void getJoker() {
    // display dice image
    String diceImage = "/ws2022/assets/Dice/Joker.png";
    Image cover = new Image(this.getClass()
            .getResource(diceImage)
            .toExternalForm());
    dice.setImage(cover);
    dice.setFitWidth(value: 100);
    dice.setFitHeight(value: 100);

    // remove roll Dice button
    boardgame.getChildren().remove(rollDice);

    // add which color button
    chooseColor.setPrefWidth(value: 297);
    chooseColor.setPrefHeight(value: 50);
    chooseColor.setOnAction(event -> {
```

```
    // add which color button
    chooseColor.setPrefWidth(value: 297);
    chooseColor.setPrefHeight(value: 50);
    chooseColor.setOnAction(event -> {
        try {
            Stage popupwindow = new Stage();
            popupwindow.initModality(Modality.APPLICATION_MODAL);
            popupwindow.setTitle(value: "This is a pop up window");
            FXMLLoader loader = new FXMLLoader(
                    this.getClass().getResource(name: "/ws2022/fxml/WhichColor.fxml"));
            Parent popUp = loader.load();
            popupwindow.setY(GameManager.stage.getY() + GameManager.stage.getHeight() / 3.5);
            popupwindow.setX(GameManager.stage.getX() + GameManager.stage.getWidth() / 7.75);
            WhichColorController gpc = loader.getController();
            gpc.display();
            Scene scene = new Scene(popUp);
            popupwindow.setScene(scene);
            popupwindow.showAndWait();

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    });
    boardgame.setColumnIndex(chooseColor, value: 2);
    boardgame.setRowIndex(chooseColor, value: 5);
    boardgame.setColumnSpan(chooseColor, value: 3);
    boardgame.getChildren().add(chooseColor);
}
```

6.3.3.3 Guess picture

When we click the “Guess Picture” button, one popup window will appear and ask us to submit the picture under the corresponding cover.

```
guessPicture.setOnAction(event -> {
    try {
        soundc.click();
        Stage popupwindow = new Stage();
        popupwindow.initModality(Modality.APPLICATION_MODAL);
        popupwindow.setTitle(value: "This is a pop up window");

        popupwindow.setY(GameManager.stage.getY() + GameManager.stage.getHeight() / 3.5);
        popupwindow.setX(GameManager.stage.getX() + GameManager.stage.getWidth() / 7.75);
        FXMLLoader loader = new FXMLLoader(
                this.getClass().getResource(name: "/ws2022/fxml/guessPicture.fxml"));
        Parent popUp = loader.load();
        GuessPictureController gpc = loader.getController();
        gpc.display();
        Scene scene = new Scene(popUp);
        popupwindow.setScene(scene);
        popupwindow.showAndWait();

    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
});
boardgame.setColumnIndex(guessPicture, value: 2);
boardgame.setRowIndex(guessPicture, value: 5);
boardgame.setColumnSpan(guessPicture, value: 3);
boardgame.getChildren().add(guessPicture);
```

In the popup for Guess Picture, the question “What is the image under [color] cover?” will be set and the color will be changed to the corresponding cover which is the result from rolling dice. The comboBox also sets the items, which picture that has not been guessed.

```
public static ArrayList<String> getArrayValue() {
    ArrayList<String> result = Disc.convertToValue(myList);
    Collections.sort(result);
    return result;
}
```

```java
public boolean checkIsGuess() {
    return isGuess;
}

public static ArrayList<String> convertToValue(ArrayList<Disc> discArrayList) {
    ArrayList<String> result = new ArrayList<>();
    for (Disc disc : discArrayList) {
        if (!disc.checkIsGuess()) {
            result.add(disc.value);
        }
    }
    return result;
}
```

```java
public class GuessPictureController {

    @FXML
    private Text cover;

    @FXML
    private ComboBox<String> comboBox = new ComboBox<>();

    @FXML
    private Button submit;
    SoundController soundc = new SoundController();

    public void display() throws IOException {
        cover.setText("What is the image under " + GameManager.COLOR + " cover ?");

        comboBox.getItems().addAll(GameManager.getArrayValue());
        comboBox.setVisibleRowCount(value: 5);
    }
```

6.3.3.4 Validate answer

The game will compare between the answer of the player and the picture under cover to decide whether the player is right or wrong. To get the correct answer, we use the function getAnswer() to use the coverHashMap to get the index of the data list of pictures and then we get the value of the picture and compare it with the value from the comboBox.

```java
public static String getAnswer() {
    return GameManager.myList.get(GameManager.coverHashMap.get(GameManager.COLOR)).getValue();
}
```

If the player got the right answer, the Right Answer popup appears, then the player clicks next to choose a picture to cover. Then the player will get one score. Otherwise, the wrong answer will appear, in this popup window, the correct answer under the cover will display and when they click next, the turn will change to another player.

```java
public void clickSubmit(ActionEvent event) throws IOException {
    soundc.click();
    String myChoice = comboBox.getValue();
    String answer = GameManager.getAnswer();
    SceneController sc = SceneController.getInstance();
    if (myChoice.equals(answer)) {
        soundc.correctAnswer();
        sc.createScene(event, name: "rightAnswer");
    } else {
        soundc.wrongAnswer();
        sc.createScene(event, name: "wrongAnswer");
    }

}
```

Fig. 6.3 Validate Player answer

```java
public class RightAnswerController {
    @FXML
    private Button nextBtn;
    @FXML
    private Pane pane;
    SoundController soundc = new SoundController();

    public void closePopUp(ActionEvent event) throws IOException {
        soundc.click();
        Stage mystage = (Stage) pane.getScene().getWindow();
        mystage.close();
        GameManager.addScore();
        GameManager.updateGame(GameManager.stage);

    }
}
```

Fig. 6.4 Right Answer Controller

Fig. 6.5 GUI

```java
@FXML
public void initialize() throws FileNotFoundException {

    coverText.setText("The image under " + GameManager.COLOR + " cover is:");
    String selectedImage = "/ws2022/assets/FootballTheme/"
            +
            GameManager.getCardImage();
    Image image = new Image(this.getClass()
            .getResource(selectedImage)
            .toExternalForm());
    coverImage.setImage(image);
    coverImage.setFitWidth(value: 200);
    coverImage.setFitHeight(value: 200);
    valueText.setText(GameManager.getAnswer());
}
```

Fig. 6.6 Wrong Answer Controller

Fig. 6.7 GUI

6.3.3.5 Choose picture to cover

After selecting the correct answer, the game manager will decide whether we continue the game or not. If the number of total pictures is more than 4, we will continue the game, otherwise the player will be redirected to the leaderboard.

```
public static void updateGame(Stage stage) throws IOException {
    if (isOnline) {
        return;
    }
    totalDisc--;
    if (totalDisc > 4) {
        BoardGameController bgc = BoardGameController.getInstance();
        bgc.removeGuessPictureBtn();
        bgc.update();
        myList.get(coverHashMap.get(COLOR)).setGuess();
        return;
    } else {
        SceneController sc = SceneController.getInstance();
        // create leaderboard here
        sc.leaderboard(stage);
    }
}
```

First, the Guess Picture button will be removed and the score will be updated on screen. Then the label to announce who is playing the next turn will be replaced by the sentence: “Please choose picture to place [color] cover.”

```
public void update() {
    if (GameManager.isPlayer1Turn) {
        player1Score.setText("" + GameManager.PLAYER1.getScore());
    } else {
        player2Score.setText("" + GameManager.PLAYER2.getScore());
    }
    status.setText("Please choose picture to place " + GameManager.COLOR + " cover");
    GameManager.isChangeDisc = true;

}

public void removeGuessPictureBtn() {
    boardgame.getChildren().remove(guessPicture);
}
```

The changeDisc() function is responsible for changing the picture to the corresponding color. Also, the picture which the player answered correctly will be removed from the board game.

```
// show disc value when click on disc

public void changeDisc(MouseEvent event) {
    if (!GameManager.isChangeDisc) {
        return;
    }
    soundc.click();
    GameManager.isChangeDisc = false;
    deleteCover();
    Node sourceComponent = (Node) event.getSource();
    Coordinate coord = (Coordinate) sourceComponent.getUserData();
    String coverImage = "/ws2022/assets/Covers/" + GameManager.COLOR + ".png";
    putCover(coverImage, coord, GameManager.COLOR);
    setTurn(GameManager.isPlayer1Turn);
    bgc.createRollDiceBtn();
}

public void alertCover(MouseEvent event) {
    if (!GameManager.isChangeDisc) {
        return;
    }
    sc.showAlertMessage(AlertType.ERROR, title: "Error",
            message: "You can not choose this picture. Choose another one without cover");
}
```

```
public void deleteCover() {
    // deleteCover by coordinate
    boardgame.getChildren().remove(hashMapImageView.get(GameManager.COLOR));
    boardgame.getChildren().remove(hashMapPicture.get(GameManager.getAnswer()));
}
```

6.3.4 Endgame

6.3.4.1 Leaderboard

When the number of pictures is less than 4, the leaderboard will appear. In the leaderboard, we compare the score of 2 players and display it in order in the leaderboard table.

```
public void initialize() throws FileNotFoundException {
    soundc.victory();
    String winnerName = "";
    String loserName = "";
    int winnerScore = 0;
    int loserScore = 0;
    if (GameManager.PLAYER1.getScore() >= GameManager.PLAYER2.getScore()) {
        winnerName = GameManager.PLAYER1.getName();
        winnerScore = GameManager.PLAYER1.getScore();
        loserName = GameManager.PLAYER2.getName();
        loserScore = GameManager.PLAYER2.getScore();
    } else {
        winnerName = GameManager.PLAYER2.getName();
        winnerScore = GameManager.PLAYER2.getScore();
        loserName = GameManager.PLAYER1.getName();
        loserScore = GameManager.PLAYER1.getScore();
    }
    name_1.setText(winnerName);
    point_1.setText("" + winnerScore);
    name_2.setText(loserName);
    point_2.setText("" + loserScore);
}
```

Fig. 6.8 Source code Leaderboard

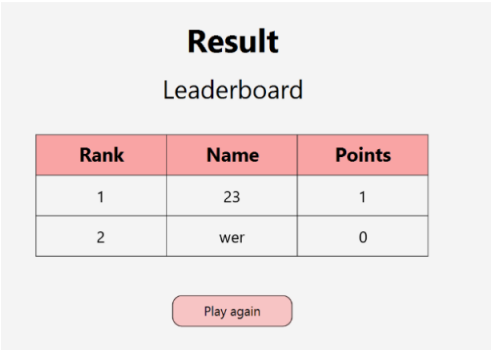


Fig. 6.9 Leaderboard GUI

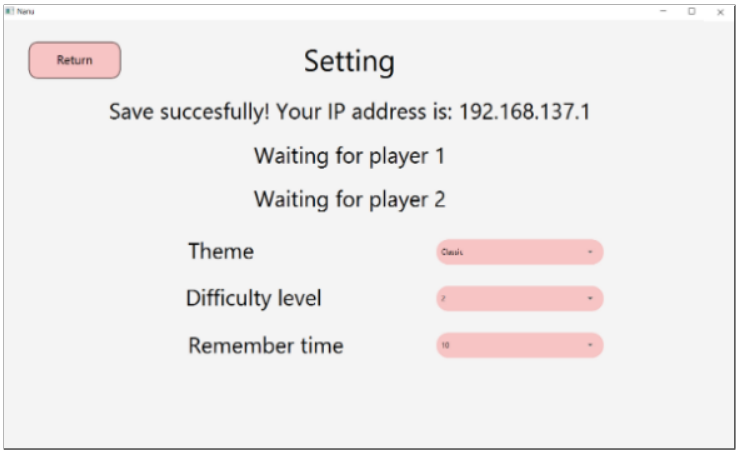
6.4 Logic for online game

6.4.1 GUI

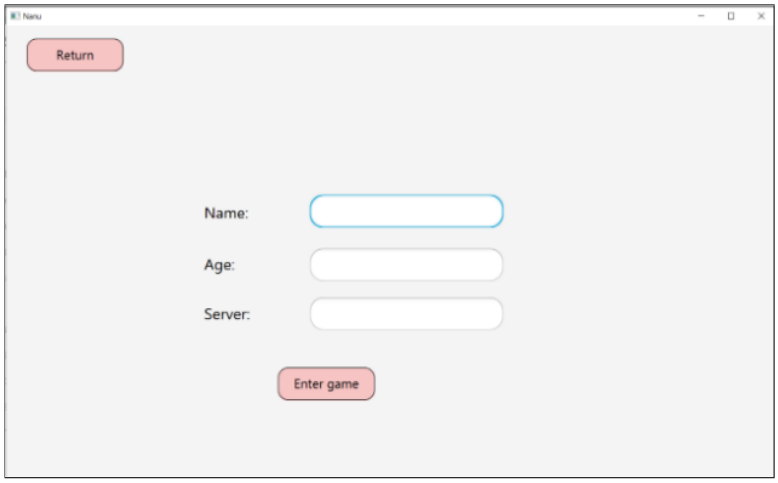
The server GUI contain: Theme, Difficulty Level, and Remember time

- Theme: Classic theme consists of normal pictures, and Football theme consists of countries' flags of the 2022 World cup.
- Difficulty Level: the number of dice to cover. It ranges from 1 to 5 dice, which is easy to hard level.
- Remember time: the amount of time for both players to remember all the pictures' position.

After filling all options, click save setting, it will appear the IPv4 address of the server for both clients connecting it.



6.4.2 Client



When choosing a client option. We get an information page containing name, age and ipv4 server address to connect.

6.4.3 Reserve keyword for transmitting

Since the message between client and server are just regular string data. We have to set some rules for client and server, to know which service server should handle as well as the client. To separate the data and the reserved keyword, we use “;” (semicolon). The rule and example message will be shown in picture below:

```
String rule = "reserved_keyword;data1;data2;...";

String message = "ENTER_PROFILE;name;tho;age;18";
```

Fig. 6.10 Sample message

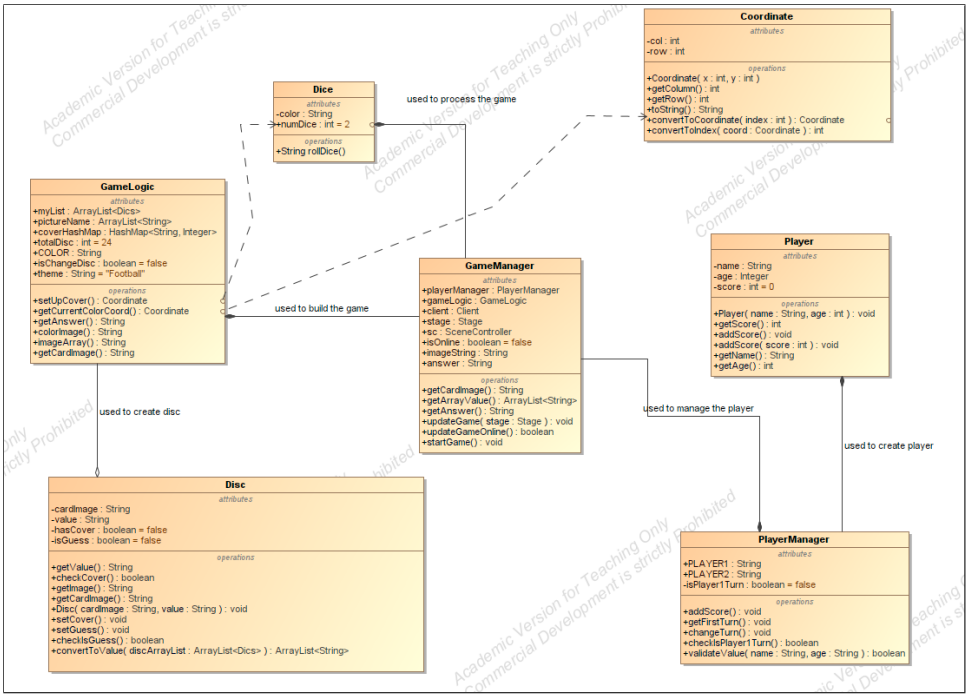
```
4   public enum Type {
5       ENTER_PROFILE,
6       DATA,
7       TURN,
8       ROLL_DICE,
9       ANSWER,
10      END_GAME,
11      UNKNOWN_TYPE,
12      POP_UP,
13      CHOOSE_COVER
14  }
```

Fig. 6.11 Reserved keyword

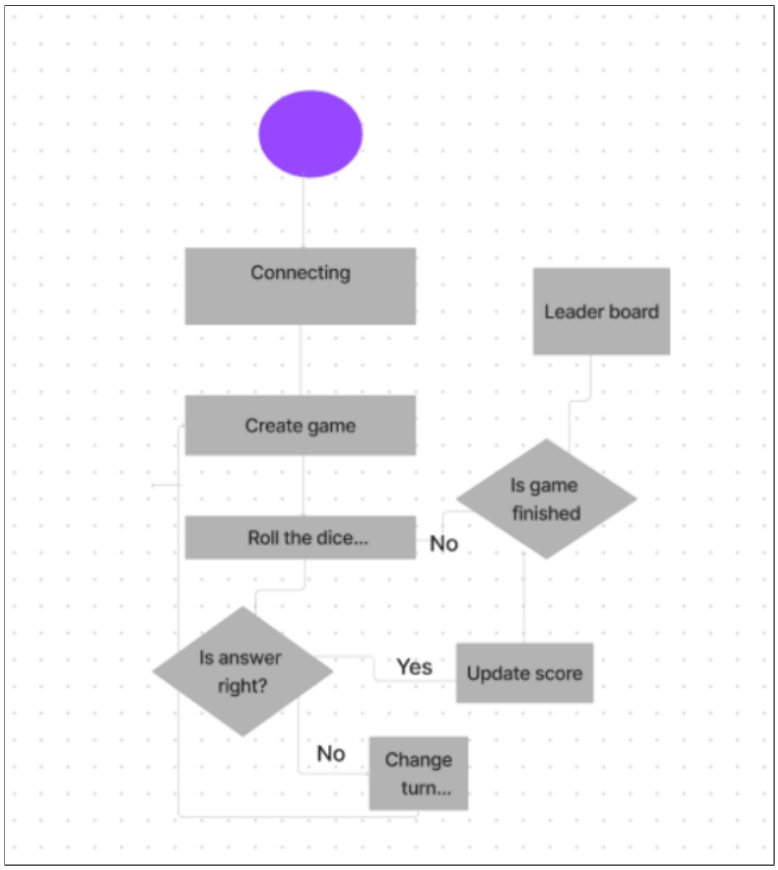
This message can be interpreted as “ENTER_PROFILE” type, have “name” = “tho”, and “age” = “18”. The server/client when received this message with the reserved keyword will put it to the corresponding method.

The reserved keyword will be saved as Enum Type in Middleware.API.java file as mentioned in Folder Structure, and this class also have method to return the reserved keyword for client and server.

6.4.4 Overall class diagram

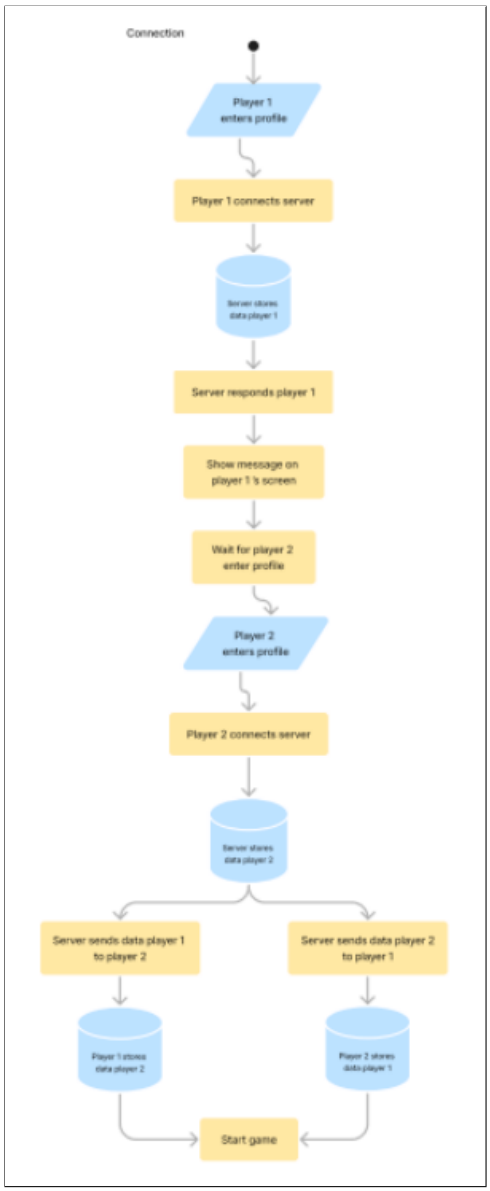


6.4.4.1 Overall flow diagram



This is our activity diagram for the online version. After the player has entered their data (Name, Age) and after connecting to the server, a new game will be created. The younger player starts. When the dice is rolled and the player has entered his answer the program checks whether the answer was correct. If the answer is wrong, the next player is the next. Otherwise if the answer was true, the player will get one point and the score will be updated. If less than five cards are left the game finishes and a leaderboard will be shown.

- Connect to server



(Flow chart show the connection between client and server)

After first player successfully connect to the server, it will show the message received from server.

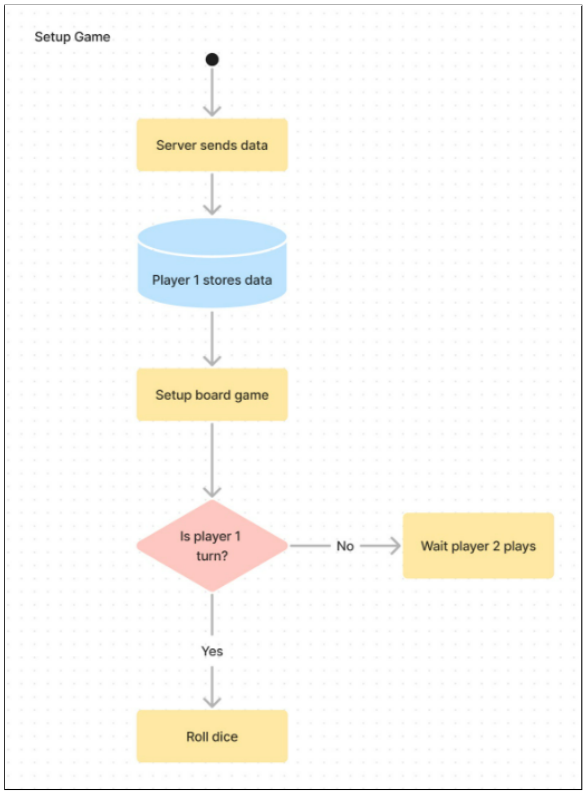
```
39    private void onReceiveEnterProfile(String s) throws IOException {
40        String[] data = s.split(regex: ";");
41        if (data.length == 1) {
42            EnterProfileOnlController epoc = EnterProfileOnlController.getInstance();
43            epoc.setStatus(); // show status connecting succesful
44            return;
45        }
46        // else this will sent the name and age of second player
47        GameManager.PLAYER2 = new Player(data[1], Integer.parseInt(data[2]));
48    }
```

(code in client when server response s = “ENTER_PROFILE;” i.e. data length array = 1)

Result when the first player connects successfully.

The game will start as soon as the player 2 connect to the server.

- Set up game



(Flow chart when set up game)

When two clients connect successfully, the server will send the card data that it generated in order, as well as the turn (who plays first) to both clients. The clients both received it and store in the GameManager for initialize the boardgame.

```
264    public void generateBoardGame() {
265        String msgToClient = API.Type.DATA.toString() + ";" + "boardgame;";
266        for (Disc disc : GameManager.myList) {
267            msgToClient = msgToClient + disc.getValue() + ";";
268            msgToClient = msgToClient + disc.getCardImage() + ";";
269        }
270        broadcastMessage(msgToClient);
271    }
```

Server sends image data to both players separately by semicolon “;” with the API “DATA”, with type “boardgame”.

```
285     public void onReceiveData(String s) {
286         String type = s.split(regex: ";")[1];
287         System.out.println(s);
288         if (type.equals(anObject: "boardgame")) {
289             setUpGame(s);
```

```
262
263     public void setUpGame(String s) {
264         String splString[] = s.split(regex: ";");
265         System.out.println(splString.length);
266         for (int i = 2; i < splString.length - 1; i = i + 2) {
267             GameManager.myList.add(new Disc(splString[i + 1], splString[i]));
268         }
269         GameManager.pictureName = GameManager.getArrayValue();
270         Platform.runLater(new Runnable() {
```

The client receive data and separate image value by using split function and add to Arraylist GameManager.myList.

Scene when two players successfully set up the board game with countdown time to remember all positions of the picture.

- Player roll dice
  When server send the color to both players, client immediately store the color value and show on screen.

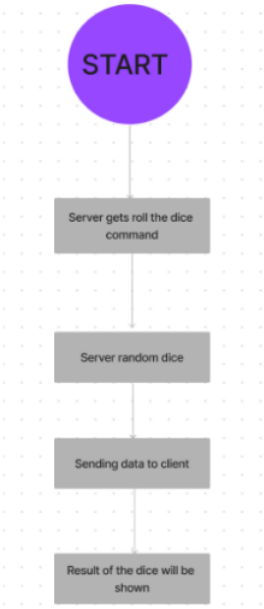


Fig. 6.13 Code implementation

Fig. 6.12 Flow chart for roll dice

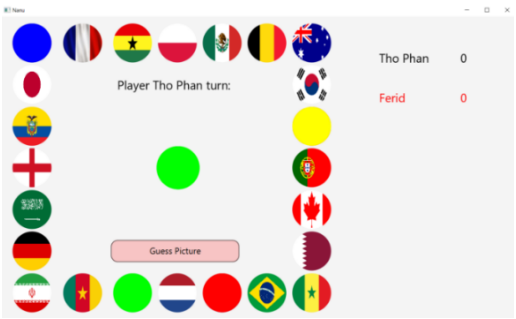
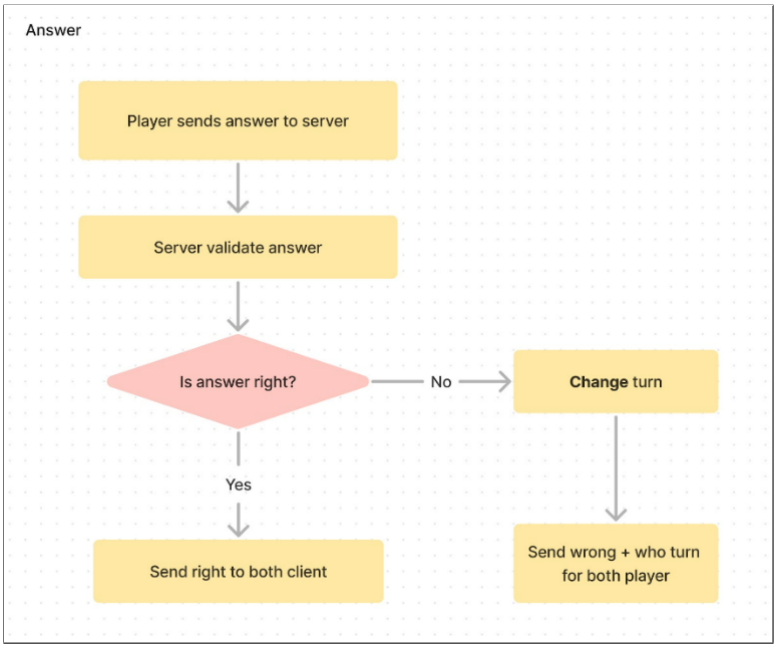


Fig. 6.14 Player perspective

Fig. 6.15 Opponent perspective

- Handle Answer



(Flow chart diagram when user choose answer)

6.5 Additional feature

6.5.1 Sound effect

We use the MediaPlayer from the JavaFX library. Here we get the path of the sound file from the folder assets and create one object MediaPlayer and pass the file path to it, then we use the method play() to play sound. Here we use 5 different sound files: click, correctAnswer, wrongAnswer, joker, dice. We call the function corresponding to the sound name when the event button click occurs or when we roll dice and the result is joker. So that the games is more enjoyable and funny.

```java
import java.net.URL;
import java.util.ResourceBundle;

import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;

public class SoundController implements Initializable {

    @Override
    public void initialize(URL url, ResourceBundle rb) {

    }

    @FXML
    public void playSound(String path) {
        Media media = new Media(this.getClass()
                .getResource(path)
                .toExternalForm());

        MediaPlayer mediaPlayer = new MediaPlayer(media);
        mediaPlayer.play();
    }

    public void click() {
        String path = "/ws2022/sound/click.mp3";
        playSound(path);
    }
```

```java
public void closePopUp(ActionEvent event) throws IOException {
    soundc.click();
    Stage mystage = (Stage) pane.getScene().getWindow();
    mystage.close();
    GameManager.addScore();
    GameManager.updateGame(GameManager.stage);

}
```

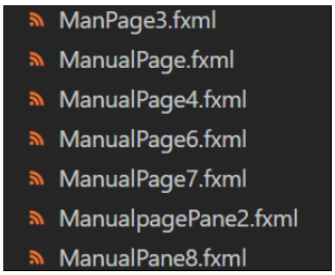
6.5.2 Generate data theme

We have two sets of pictures, Football theme and Classic Theme. The players can choose the theme in the setting. We use the combo box to take the input of the players. Instead of hard code the link to the images, we use relative links to make sure the application can be executed properly on different operating systems. For the selection in the combo box, we use the list() method from Object File to generate all the names of files, and modify it to be readable.

```
public class GenerateData {
    public static String[] Images;
    public static String[] Values = new String[24];

    public static void generateDataForFolder() {

        File directory = new File("target/classes/ws2022/assets/Theme/" + GameManager.gameLogic.theme);
        Images = directory.list();

        for (int i = 0; i < Values.length; i++) {
            String temp = Images[i].substring(beginIndex: 0, Images[i].length() - 4); // remove .jpg, .png
            String[] temp1 = temp.split(regex: "(?=\\p{Lu})"); // split when uppercase to array
            temp = Arrays.toString(temp1); // array concat to string
            Values[i] = temp.replaceAll(regex: "[^a-zA-Z0-9\\s+]", replacement: ""); // remove all special character
except blank space
        }
    }

    public static void generateDisc(ArrayList<Disc> discArray) {
        generateDataForFolder();
        for (int i = 0; i < Values.length; i++) {
            discArray.add(new Disc(Images[i], Values[i]));
        }
    }

}
```

6.5.3 Manual page

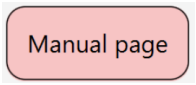
We used Scenebuilder for creating the windows of the Manualpage FXML.



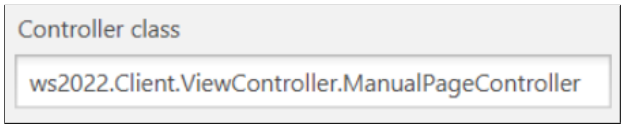
On resources we have created a separate folder to add our FXML.Files including the Manualpage windows. The windows of the manualpage have only photos to illustrate to the user how to play the game. The manualpage part includes every single photo

of the window of this Nanu-Program.

To create a full working Manualpage part:
On the homescreen:



We have created on the top left side of the homescreen a button which leads the player to the manualpage area.



We used the class “ManualPageController” to manage and to let the manualpage windows work.

On Action: The methods of the class “ManualPageController” will be put here to message the program what to do when the backbutton or the nextbutton will be clicked.

Chapter 7
Challenge

7.1 Design pattern

7.1.1 God object

In object-oriented programming, a god object (sometimes also called an omniscient or all-knowing object) is an object that references a large number of distinct types, has too many unrelated or uncategorized methods, or some combination of both. The god object is an example of an anti-pattern and a code smell. In our application, we have one god object, the Game Manager, it contains so many different objects and so many methods to control the game. We are still on the way to refactor it, in the future, we will separate it into different objects and design some better patterns.

7.1.2 Not optimal code

We realize we can not predict the errors in our design or our code in the future. So sometimes we have to redo the whole thing to fix one problem. For example, we have to rearrange the boardgame to make sure that the pop up window is not covered with pictures. We have to change it so many times. These problems cost us a lot of time, but we also learned a lot during the process.

7.2 Online version

Because this is the first time I am learning socket programming without the library supporting the API message between client and server. The most challenging part will be designing the flowchart for communication between connecting devices.

Secondly, since socket programming also requires more than one thread, it is harder to debug it. If one thread (not the main thread) got a bug and escaped, instead of terminating the program, it still continued the game without showing any bug (run-time error). After lots of time researching, I realized that by using try catch block with printStackTrace() method of Error Class to show the error.

```
79      public void onReceiveEndGame(String s) {
80          updateScore(s);
81          Platform.runLater(new Runnable() {
82              @Override
83              public void run() {
84                  SceneController sc = SceneController.getInstance();
85                  try {
86                      sc.loadSceneByStage(GameManager.stage, name: "Leaderboard");
87                  } catch (Exception e) {
88                      e.printStackTrace();
89                      // TODO: handle exception
90                  }
91              }
92          });
93      }
```

Thirdly, the server is not deployed online. To run this game, we have to run it in the same private network with the firewall turned off in order to be discoverable by another machine.

7.3 Convention of naming

First we have a problem in naming convention, because each of us named the file in different ways so the directory is very messy. After that, we decide on some conventions:

- _variable_ start with low character. Ex: ‘player1‘, ‘playerController‘
- _function_ start with Capital letter. Ex: ‘DrawCard()‘, ‘StartGame()‘
- _class_ always starts with Capital letter. Ex: ‘Controller‘, ‘Card‘

7.4 Run-time Error

During development, creating bugs is inevitable.Thanks to debugging, we learn so much about coding. Below will be an example for one of those bug, this bug is made by misunderstood about the fundamental of switch-case java:

This method is in the ClientHandler file, which handles a specific type of message that we defined in the implementation for logic of online games. The SET_COLOR

is run when a client rolls the dice and gets a joker. They can choose the color and send it back to the server to store the color data.

```
ClientHandler.java M
src > main > java > ws2022 > Server > ClientHandler.java > ClientHandler > handleMessage(String)
80      public void handleMessage(String s) throws IOException {
81          API.Type type = API.getTypeFromClient(s);
82          switch (type) {
83              case SET_COLOR:
84                  handleSetColor(s);
85              case CLOSE_CONNECTION:
86                  handleCloseConnection();
87                  break;
88              default:
89                  System.out.println(x: "Unspecify type");
```

Initially, I thought that switch case just like an if-else-then statement, which is when type is "SET_COLOR" it will execute the method and then jump out the switch-case, which is not. As soon as they click next, the server will handle the message and also

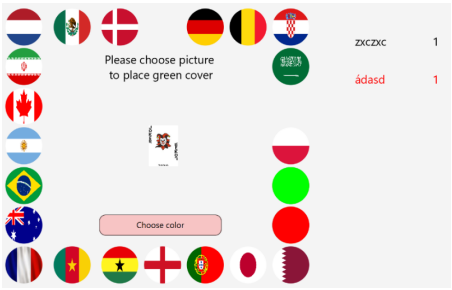


Fig. 7.1 Player get Joker

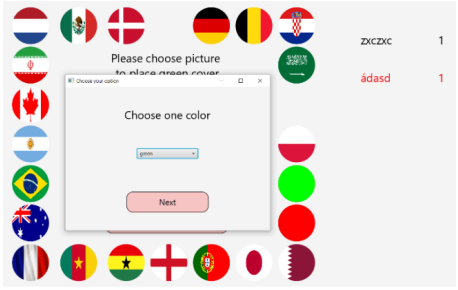


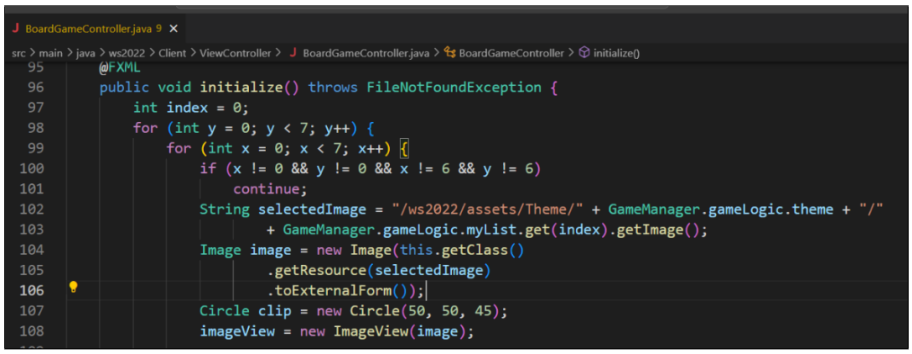
Fig. 7.2 Player choose color

close the connection with that client. When I was debugging I realized that after the handleSetColor() call, it also called handleCloseConnection() (which is close to the connection of a specific client). I research online and find that the switch will find the suitable case and then after that it will execute all other cases below it until it meets a break.

7.5 Operating System Limitation

- We have tried the application on different machines and realized that JavaFX can only run on specific OS environments that we are working on.

- For example: In the initialize() method of BoardGameController, where it will be set up the board game by filling the selectedImage into the gridPane. In windows, it works perfectly normal. However, in Linux, it got an error due to an import image problem.



- Therefore, this application now can only run in the Windows version. In the future, we will try to make the application OS independent.

Chapter 8
Conclusions and Future Work

8.1 How was the team work

In the beginning, we always had regular meetings for each member to catch up on the work progress. In each meeting, we summarized the completed workload and divided tasks equally and according to each member's capacity. The problems were explored and supported together. Moreover, instead of finishing the application first and then going to complete the document, our team chose to handle both at the same time. It was a suitable approach for our team. We were separated into two groups: coding and report writing. Each member was responsible for at least one part of both two groups, allowing all members to experience tasks in all fields.

However, we also ran into several working-style and communication problems. When we first started working together, we had trouble coordinating our methods of working, coding, and data storage. The operating procedure is therefore inconsistent. After a few revisions of adjusting and unifying the rules and the way of working, the team's productivity increased dramatically. Since English is the main language used for communication, we occasionally experienced misunderstandings when texting or discussing.

8.2 What you have learned

8.2.1 New board game

This is our first time playing the Nanu game. So it is difficult for us at first since learning a new game takes time, and almost all instructional videos on the internet are in German. Fortunately, we found an English video and discussed it together to learn more about the rules. This was an opportunity for us to play and explore

German games. It is a fantastic memory game that will assist children in improving their memory abilities.

8.2.2 New tools

During the project, we had to self-learn a ton of tools. We learned how to draw UML diagrams and practiced on the Magic Systems of Systems Architect tool. These diagrams help us to see the project's flow and provide us a chance to improve our diagramming abilities. We utilized all three of these programs to create rough sketches for GUI designs: Balsamiq, Scene Builder, and Figma. Then, depending on the drawing, we learned and added more colors using CSS and HTML. For storing data, we used GitHub and Google Drive. We could easily exchange codes together via Github and files via Google Drive.

JavaFX is the project's primary tool. We need additional time to study and become accustomed to writing JavaFX code because we lack experience with it. We acquired knowledge from a variety of sources, including teacher-provided materials and YouTube online courses, among others. We could put the fundamental information to use after we had learned it.

8.2.3 Teamwork

During this project, we always adjusted and unified our working method to suit the work progress. We gained knowledge and experience on how to control the workflow and improve problem-solving skills. Moreover, English is the primary language in which we communicate and complete all tasks. Therefore, our English skills have improved.

8.3 Ideas for the future development of your application, new algorithms

During our team work we also noticed some things which can be taken into account at projects in the future. We have the opinion that the game could be more more better, if more than two player can play this game. Because if more than two players can play this game, there will be more fun and more interaction between the players. Creating an account is also a good aim for the future because the user or players can manage their high score or performance. Creating the GUI of the game in 3D-Design can make the game more efficient and serious. In addition adding a background music is also a good aim.

References

1. How the 60-30-10 rule saved the day. (2020). Retrieved 2 February 2023, from https://uxdesign.cc/how-the-60-30-10-rule-saved-the-day-934e1ee3fdd8
2. Hernandez, F. (2016). Maven Project Structure Example. Retrieved 2 February 2023, from https://examples.javacodegeeks.com/enterprise-java/maven/maven-project-structure-example#: :text=The%20target%20folder%20is%20the,content%20with%20mvn%20clean%20command
3. IBM Documentation. (2023). Retrieved 2 February 2023, from https://www.ibm.com/docs/en/spm/7.0.4?topic=files-classpath-file
4. God object - Wikipedia. (2012). Retrieved 2 February 2023, from https://en.wikipedia.org/wiki/God_object
5. Ravensburger Nanu - by Vakko Korea (Jolie). (2023). Retrieved 2 February 2023, from https://www.youtube.com/watch?v=dkwNihodVnwab_channel=%EB%9D%BC%EB%B2%A4%EC%8A%A4%EB%B6%80%EB%A5%B4%EA%B1%B0
6. A REVIEW ON JAVA HASHMAP AND TREEMAP. (2023). Retrieved 2 February 2023, from http://ijeast.com/papers/134-138,Tesma501,IJEAST.pdf
7. Applications, Advantages and Disadvantages of Hash Data Structure - GeeksforGeeks. (2022). Retrieved 2 February 2023, from https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/
8. What is Singleton? - Definition from Techopedia. (2023). Retrieved 2 February 2023, from https://www.techopedia.com/definition/15830/singleton
9. 4 Disadvantages of Singleton Pattern and How to Fix Them. (2022). Retrieved 2 February 2023, from https://methodpoet.com/disadvantages-of-singleton-pattern/
10. What is MVC Framework? (Components, Use And Benefits) (2022). Retrieved 2 February 2023, from https://in.indeed.com/career-advice/career-development/mvc-framework
11. What were the major problems with MVC Framework ? - GeeksforGeeks. (2022). Retrieved 2 February 2023, from https://www.geeksforgeeks.org/what-were-the-major-problems-with-mvc-