FRANKFURT UNIVERSITY OF APPLIED SCIENCES

JAVA ADVANCED OOP

---

# Nanu? Game Report

---

*Author:*
*Nguyen Minh Nguyen*
*Yen Ngoc Nguyen*
*Tan Khai Huynh*

*Supervisor:*
Prof. Dr. Doina Logofatu
Mr. Julia Garbaruk

*Documentation*

Group NANU?
Computer Science and Engineering

February 9, 2024


# Declaration of Authorship

We hereby certify that the project report we are submitting is entirely our own original work except where otherwise indicated. We did not submit this work anywhere else before. We are aware of the University's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.
Signed:

Nguyen Minh Nguyen - 1520123

Yen Ngoc Nguyen - 1518788

Tan Khai Huynh - 1515592

Date: February 9, 2024

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

# *Abstract*

Faculty Name
Computer Science and Engineering

Bachelor of Computer Science

**Nanu? Game Report**

by Nguyen Minh Nguyen
Yen Ngoc Nguyen
Tan Khai Huynh

Our project is centered around the development of the board game "Nanu?" using Java and JavaFX, emphasizing object-oriented programming (OOP) concepts. The project's primary objective is to create a sophisticated digital version of the "Nanu?" board game, incorporating OOP principles. The game development process involves leveraging Java's OOP features to model game elements, such as players, cards, color lids, and the game board, using classes and inheritance. JavaFX is employed for creating an intuitive and visually appealing user interface, allowing players to interact seamlessly with the digital adaptation of "Nanu?"

# *Acknowledgements*

We express our sincere gratitude to Prof. Dr. Doina Logofatu and Mr. Julia Garbaruk for their unwavering support throughout the development of our "Nanu?" board game project. Their guidance and expertise were instrumental in achieving this significant milestone. Their commitment to excellence in crafting the game's user interface, implementing Java OOP concepts, encouraging self study and research have been invaluable. Their dedicated contributions have played a crucial role in the success of our game.

# Contents

# Chapter 1

# Introduction

In this report, we present a detailed account of our project focused on the digital adaptation of the classic memory board game "Nanu?" 1.2 Utilizing Java programming and JavaFX, our team has diligently worked to transform the traditional board game into an interactive digital experience. The report will delve into the technical aspects, design considerations, and overall methodology employed during the development process, highlighting the use of object-oriented programming (OOP) concepts and leveraging JavaFX's graphical capabilities to create an immersive gaming environment. This documentation serves to provide a comprehensive overview of our project's development journey.


FIGURE 1.1: Graphical User Interface

## 1.1 Project Description

This project involves the digital transformation of the board game "Nanu?" 1.2 through the use of Java programming and JavaFX. The objective is to create an engaging and interactive digital version by applying object-oriented programming (OOP) concepts and leveraging JavaFX's graphical capabilities. The project encompasses the development of game logic, user interface, and overall functionality, with the aim of simulating the original board game as similarly as possible.

## NANU? Board Game

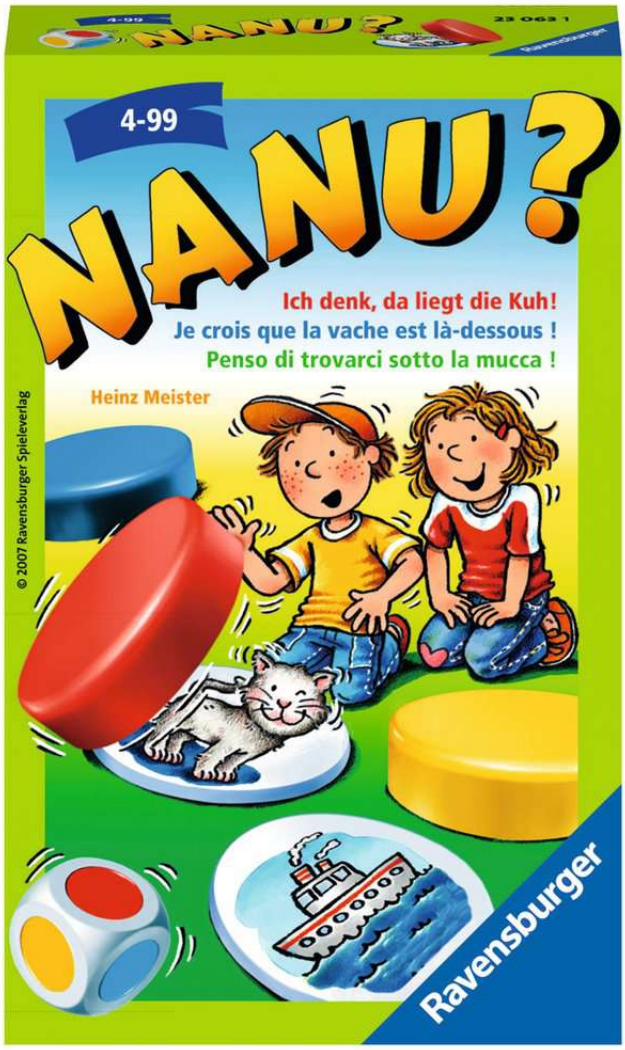


FIGURE 1.2: Game Box

### Contents

- 24 picture cards
- 5 lids

– 1 color cube

### Goal of the Game

Who remembers best which motif is hidden under the colorful lids? With the correct predictions, you will win the most picture cards!

### Preparation

Before the first game, carefully remove the round picture cards from the punch board. If younger children are playing along, it is best to look at and name the motifs depicted on them together.

### Let's Start

Distribute all the picture cards face up on the table.

The oldest player then takes the five lids and covers five picture cards one after the other. He always announces which lid he will place over which picture card. An example: "I put the red lid over the cow, the green lid over the cherry....

The youngest player now rolls the color die. The color dot rolled indicates which lid he can look under - but first, he names the motif he suspects is under the lid. Only then does he reveal it!

**Example:** You roll the color blue. Do you remember what's hidden under the blue lid? You're pretty sure it's the frog and you say it out loud. Then you lift the lid. Great, it's actually true! Take the picture card and place it face down in front of you.

Then name another motif and place the lid over it. Now five picture cards are hidden again. It is now the next player's turn and can roll the dice and guess. If your prediction is not correct, put the lid back over the picture card after everyone has seen it. Your turn is over and it continues with the next player. If you roll the joker, you can choose which lid you want to lift. But be careful: the lid that was taken last is not allowed! Here, too, you first name the motif before you can look under the lid and see whether you guessed correctly.

### End of the Game

The round ends when there are only four picture cards left in play and the fifth, freed lid can no longer be placed over any motif.

Whoever now has the most picture cards wins the game.

## 1.2 Related Work

### 1.2.1 Related Algorithms

1. **Memory Matching Algorithm:** Essential for recalling motif positions and making successful matches.
2. **Pattern Recognition Algorithm:** Identifies visual similarities between motifs to aid in matching.
3. **AI for Player Guidance:** Provides recommendations or plays optimally to assist players.
4. **Difficulty Scaling Algorithm:** Adjusts game complexity based on player performance for balanced challenge.
5. **Modulo Algorithm for Turn Management:** Ensures fair distribution of turns among players by cycling through player indices using modulo arithmetic.
6. **Optimization Algorithms:** Enhances lid placement strategies and reduces time complexity for smoother gameplay.

### 1.2.2 Similar Application


FIGURE 1.3: Matching Pairs Game

The Matching Pairs memory game 1.3 is similar to the NANU? memory game in several ways:

1. **Concepts:** Memory and pattern recognition are key components of both games. Players must match pairs with identical symbols or images in the matching pairs memory game by keeping track of the locations of different cards. In the NANU memory game, players must also recollect the locations of various motifs concealed beneath lids and match them with matching symbols or pictures.

2. **Mechanisms:** Both games revolve around the idea of uncovering and matching hidden elements. Players flip cards to disclose their concealed symbols in the matching pairs memory game, then use their recollection of the positions of previously revealed cards to try and locate matching pairs. Players open lids in the NANU memory game to reveal motifs, then try to match them by remembering their locations and connections.

3. **Engagement and Pleasure:** Players of all ages can have fun with these games. Finding hidden parts and properly matching them is one task that makes you feel accomplished and satisfied. The competitive aspect of the games—whether it be against other players or oneself—increases their replay value and fun factor.

4. **Educational Value:** Improving memory and cognitive skills is one of the games' main educational benefits. Regularly engaging in these activities enhances one's visual-spatial, attention span, and memory recall. As they devise strategies for quickly finding and matching pieces, players foster strategic thinking and problem-solving abilities.

## 1.3 Ideas

1. **Start Scene:**
    - Welcome message and game title.
    - Brief instructions on how to play.
    - Start button to proceed to the next scene.
2. **Instructions Scene:**
    - Detailed game rules and instructions.
    - Visual aids or examples to explain gameplay.
    - Back button to return to the start scene.
3. **Player Selection Scene:**
    - Options to choose the number of players (2-4). Default 3 players (as there are 3 people in our team).
    - Start button to initiate the main game scene.
4. **Main Game Scene:**
    - Display 5 color lids, 24 images, and a color dice (5 color sides, 1 joker side) .
    - Display each player's label with their score.
    - Submit field for players to enter answers.
    - Submit button to check answers and update scores.
    - Game logic and algorithms to manage turns, scoring, and validate answers.
5. **End Game Scene:**
    - Display the winner and congratulate them.
    - Option to start another game or return to the player selection scene.

## 1.4 Development Potential


FIGURE 1.4: Potential

Our "Nanu?" game project, built using JavaFX and SceneBuilder, holds exciting potential for future advancements in gaming technology:

- Integration of virtual reality (VR) elements could offer players an immersive gaming experience, enhancing strategic decision-making and interactions.
- Advancements in mobile and online gaming platforms could expand the game's reach to a global audience, fostering a vibrant online community.
- The project's adaptability allows for updates with fresh challenges, characters, or themes, ensuring dynamic and engaging gameplay.

# Chapter 2

# Team Work

## 2.1 General Assignment

Below is the work assignment of our team:

| Members | Work |
|---|---|
| **Yen Ngoc Nguyen** | • **Code**<br>• **Main UI Designer**<br>• **Design poster**<br>• **Write poster** |
| **Nguyen Minh Nguyen** | • **Main Coder**<br>• Design UI<br>• **Design poster**<br>• **Write poster** |
| **Tan Khai Huynh** | • Code<br>• Design UI<br>• **Draw UML Diagrams**<br>• **Present and write report** |

## 2.2 Individual Summary

### 2.2.1 Member 1

| | |
|---|---|
| **Name:** | TAN KHAI, HUYNH |
| **ID:** | 1515592 |

**Exercise Group: Wednesday**

**Part 1: Basics Summary**

- Basic problems: 100% solved
- Kattis problems: 90% solved
- Kattis score: 238.5
- 5 times presented Kattis Problems in front of the class:
  - 25/10/2023: Harshad Numbers
  - 01/11/2023: Election Paradox
  - 08/11/2023: Pseudoprimes
  - 15/11/2023: Good Morning!
  - 22/11/2023: Polygon Area

**Part 2: Application Summary - NANU?**

- I have contributed to the coding and UI development of our NANU? Projekt. Moreover, I presented the product of our team to class, did UML modeling, and wrote the report.
- I did 1/3 whole work
- 3 feedback talks:
  - 25/01/2024: Presentation to Prof. Logofatu
  - 31/01/2024: Presentation in front of the class
  - 01/02/2024: Presentation in front of the class

### 2.2.2 Member 2

| | |
|---|---|
| **Name:** | NGUYEN MINH, NGUYEN |
| **ID:** | 1520123 |

**Exercise Group: Wednesday**

**Part 1: Basics Summary**

- Basic Problem: 100% solved
- Kattis Problem: 80% solved (of total problems, more than requirement)
- Kattis score: 168.2
- 2 times presented Kattis Problems in front of the class:
  - 15/11/2023: Mravi
  - 22/11/2023: Triangle Ornaments

**Part 2: Application Summary - NANU?**

- I am the main developer of our project "Nanu?". I also took part in the UI development, poster design, and wrote the report.
- I did 1/3 whole work
- 3 feedback talks:
  - 25/01/2024: Presentation to Prof. Logofatu
  - 31/01/2024: Presentation in front of the class
  - 01/02/2024: Presentation in front of the class

### 2.2.3 Member 3

| **Name:** | YEN NGOC, NGUYEN |
|---|---|
| **ID:** | 1518788 |

**Exercise Group: Wednesday**

**Part 1: Basics Summary**

- Basic Problem: 100% solved
- Kattis Problem: 80% solved (of total Kattis problems, more than requirement)
- Kattis score: 135.8
- 2 times presented Kattis Problems in front of the class:
  - 15/11/2023: Who wins?
  - 22/11/2023: Bits

**Part 2: Application Summary - NANU?**

- I am the main UI developer of our NANU project and poster designer. Additionally, I also contributed to the coding and report.
- I did 1/3 whole work
- 3 feedback talks:
  - 25/01/2024: Presentation to Prof. Logofatu
  - 31/01/2024: Presentation in front of the class
  - 01/02/2024: Presentation in front of the class

# Chapter 3

# UML Modeling

In our "Nanu?" game project, Object-Oriented Programming (OOP) concepts and Unified Modeling Language (UML) can be applied to design a well-structured and modular system.

## 3.1 General Diagram

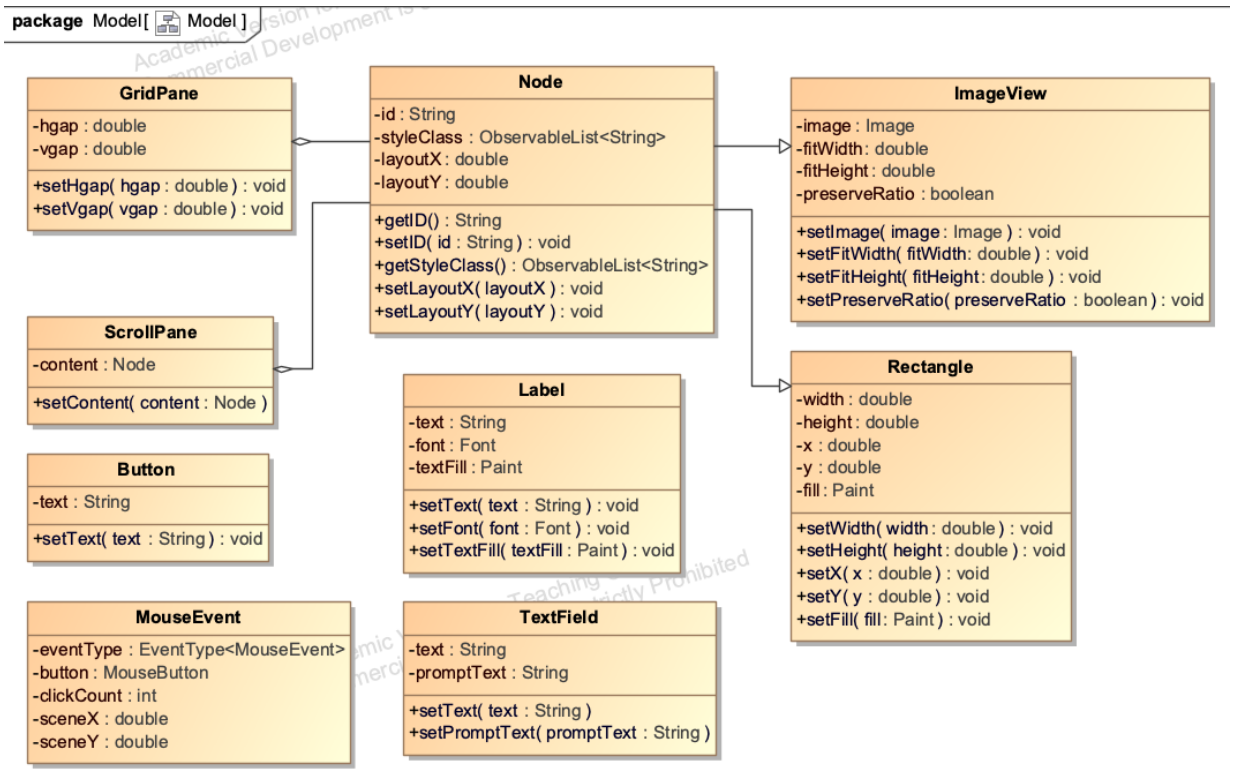


FIGURE 3.1: Basic Illustration

These classes and their attributes/operations in Figure 3.1 are fundamental in JavaFX for building graphical user interfaces and handling user interactions.

1. **Node**
   **Attributes:**

   – `id:` `String` - Identifies the node with a string.

- `styleClass: ObservableList<String>` - List of style classes for applying CSS styles.
- `layoutX: double` - X-coordinate for layout positioning.
- `layoutY: double` - Y-coordinate for layout positioning.

**Operations:**

- `getId()` - Gets the ID of the node.
- `setId(String id)` - Sets the ID of the node.
- `getStyleClass(): ObservableList<String>` - Retrieves the style classes.
- `setLayoutX(double layoutX)` - Sets the X-coordinate for layout.
- `setLayoutY(double layoutY)` - Sets the Y-coordinate for layout.

2. **Rectangle**
**Attributes:**

- `width: double` - Width of the rectangle.
- `height: double` - Height of the rectangle.
- `x: double` - X-coordinate of the rectangle's position.
- `y: double` - Y-coordinate of the rectangle's position.
- `fill: Paint` - Paint object representing the fill color.

**Operations:**

- `setWidth(double width)` - Sets the width of the rectangle.
- `setHeight(double height)` - Sets the height of the rectangle.
- `setX(double x)` - Sets the X-coordinate of the rectangle.
- `setY(double y)` - Sets the Y-coordinate of the rectangle.
- `setFill(Paint fill)` - Sets the fill color of the rectangle.

3. **ImageView**
**Attributes:**

- `image: Image` - Image object to be displayed.
- `fitWidth: double` - Width of the image view.
- `fitHeight: double` - Height of the image view.
- `preserveRatio: boolean` - Boolean indicating whether to preserve the image's aspect ratio.

**Operations:**

- `setImage(Image image)` - Sets the image to be displayed.
- `setFitWidth(double fitWidth)` - Sets the width of the image view.
- `setFitHeight(double fitHeight)` - Sets the height of the image view.

- `setPreserveRatio(boolean preserveRatio)` - Sets whether to preserve the image's aspect ratio.

4. **Label**
   **Attributes:**
   - `text: String` - Text content of the label.
   - `font: Font` - Font for the label text.
   - `textFill: Paint` - Paint object representing the text color.

   **Operations:**
   - `setText(String text)` - Sets the text content of the label.
   - `setFont(Font font)` - Sets the font for the label text.
   - `setTextFill(Paint textFill)` - Sets the text color of the label.

5. **GridPane**
   **Attributes:**
   - `hgap: double` - Horizontal gap between grid elements.
   - `vgap: double` - Vertical gap between grid elements.

   **Operations:**
   - `setHgap(double hgap)` - Sets the horizontal gap.
   - `setVgap(double vgap)` - Sets the vertical gap.

6. **Button**
   **Attributes:**
   - `text: String` - Text content displayed on the button.

   **Operations:**
   - `setText(String text)` - Sets the text content of the button.

7. **TextField**
   **Attributes:**
   - `text: String` - Text content of the text field.
   - `promptText: String` - Text to display as a prompt when the field is empty.

   **Operations:**
   - `setText(String text)` - Sets the text content of the text field.
   - `setPromptText(String promptText)` - Sets the prompt text.

8. **ScrollPane**
   **Attributes:**

- `content: Node` - Node to be displayed within the scroll pane.

**Operations:**

- `setContent(Node content)` - Sets the content to be displayed within the scroll pane.

9. **MouseEvent**
   **Attributes:**

   - `eventType: EventType<MouseEvent>`
   - `button: MouseButton`
   - `clickCount: int`
   - `sceneX: double`
   - `sceneY: double`

   **Operations:** Various methods for retrieving event-related information.

## 3.2 Main Diagram

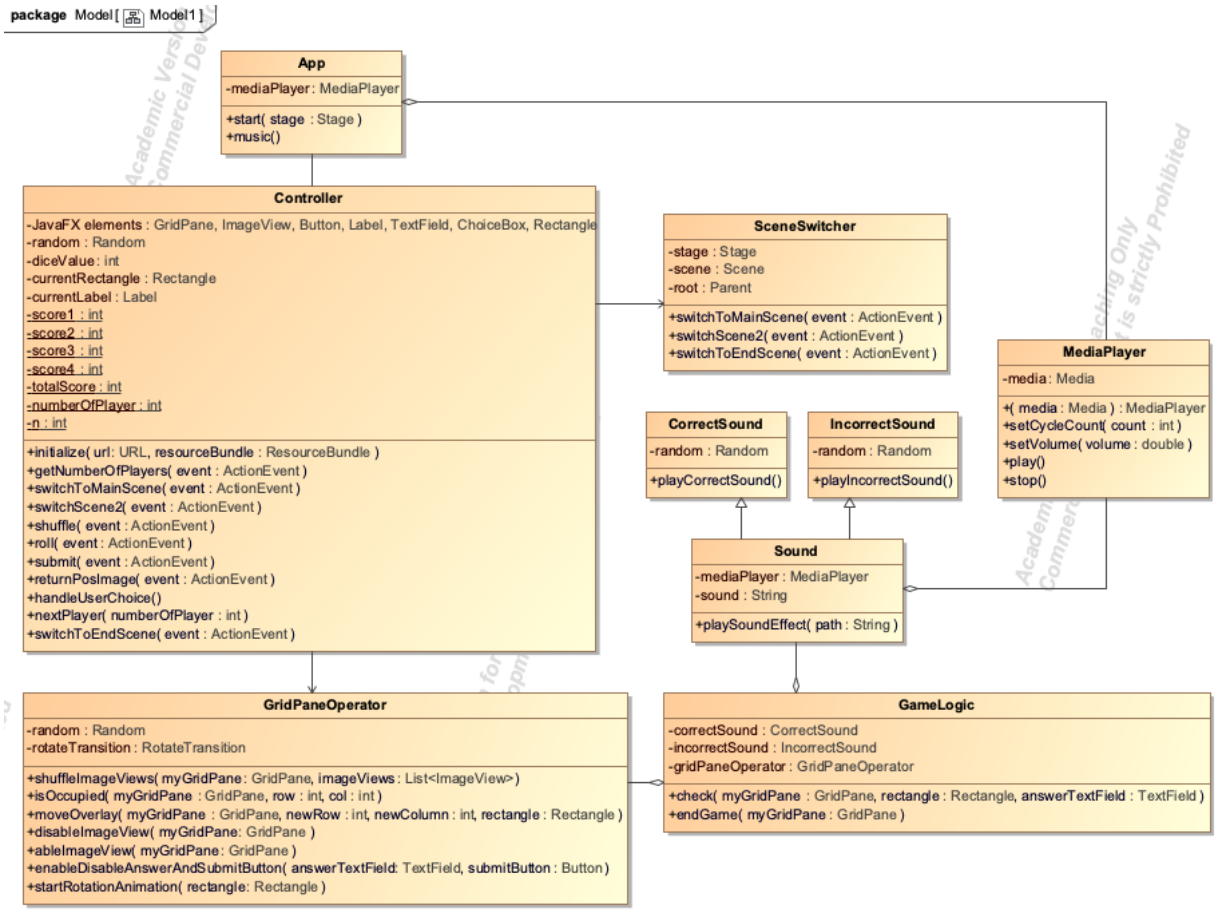


FIGURE 3.2: Main Illustration

Figure 4.4 represents the main implementation structure of our NANU? game.

1. **Sound**
   **Attributes:**

   - `mediaPlayer: MediaPlayer`
   - `sound: String`

   **Operations:**

   - `playSoundEffect(String path)` - Plays a sound effect specified by the file path.
   - `pause()` - Pauses the current sound playback.
   - `resume()` - Resumes the paused sound playback.
   - `stop()` - Stops the sound playback.

2. **SceneSwitcher**
   **Attributes:**

   - `stage: Stage`

- `scene: Scene`
- `root: Parent`
- `mainScenePath: String`
- `scene2Path: String`
- `endScenePath: String`

**Operations:**

- `loadFXML(String fxmlPath)` - Loads an FXML file.
- `switchToMainScene(ActionEvent event)` - Switches to the main game scene.
- `switchScene2(ActionEvent event)` - Switches to another scene.
- `switchToEndScene(ActionEvent event)` - Switches to the end game scene.

3. **GridPaneOperator**
**Attributes:**

- `random: Random`
- `rotateTransition: RotateTransition`
- `rotationAngle: double`
- `overlayMaxRow: int`
- `overlayMaxCol: int`
- `rotationSpeed: int`

**Operations:**

- `shuffleImageViews(GridPane myGridPane, List<ImageView> imageViews)` - Shuffles image views within a GridPane.
- `isOccupied(GridPane myGridPane, int row, int col)` - Checks if a grid cell is occupied.
- `moveOverlay(GridPane myGridPane, int newColumn, int newRow, Rectangle rectangle)` - Moves an overlay within a GridPane.
- `disableImageView(GridPane myGridPane)` - Disables image views within a GridPane.
- `ableImageView(GridPane myGridPane)` - Enables image views within a GridPane.
- `enableDisableAnswerAndSubmitButton(TextField answerTextField, Button submitButton)` - Enables or disables answer text field and submit button.
- `startRotationAnimation(Rectangle rectangle)` - Starts rotation animation on a rectangle.

4. **GameLogic**
**Attributes:**

- `correctSound: CorrectSound`

- `incorrectSound: IncorrectSound`
- `gridPaneOperator: GridPaneOperator`
- `attempts: int`
- `maxAttempts: int`

**Operations:**

- `check(GridPane myGridPane, Rectangle rectangle, TextField answerTextField)` - Checks the correctness of an answer.
- `endGame(GridPane myGridPane)` - Ends the game and performs necessary actions.
- `resetAttempts()` - Resets the number of attempts.
- `incrementAttempts()` - Increments the number of attempts.

5. **Controller**
   **Attributes:**
   - `numberOfPlayers: int`
   - `gameLogic: GameLogic`
   - `...` - (List of various attributes)

   **Operations:**
   - `initialize(URL url, ResourceBundle resourceBundle)` - Initializes the controller.
   - `getNumberOfPlayers(ActionEvent event)` - Retrieves the number of players.
   - `startGame(ActionEvent event)` - Starts the game.
   - `handleAnswerSubmit(ActionEvent event)` - Handles the submission of an answer.
   - `...` - (List of various operations)

6. **App**
   **Attributes:**
   - `mediaPlayer: MediaPlayer`
   - `backgroundMusicPath: String`

   **Operations:**
   - `main(String[] args)` - Main method to launch the application.
   - `start(Stage stage)` - Starts the application.
   - `music()` - Plays background music.
   - `stopMusic()` - Stops background music.

7. **CorrectSound extends Sound**
   **Attributes:**

- `random: Random`
- `...` - (List of various attributes)

**Operations:**

- `playCorrectSound()` - Plays a correct sound effect randomly selected.

8. **IncorrectSound extends Sound**
   **Attributes:**

   - `random: Random`
   - `...` - (List of various attributes)

   **Operations:**

   - `playIncorrectSound()` - Plays an incorrect sound effect randomly selected.

9. **MediaPlayer**
   **Attributes:**

   - `media: Media`
   - `volume: DoubleProperty`
   - `...` - (List of various attributes)

   **Operations:**

   - `MediaPlayer(Media media)` - Constructor to create a media player.
   - `setMedia(String path)` - Sets the media file path.
   - `setVolume(double volume)` - Sets the volume of the media player.
   - `play()` - Plays the media.
   - `pause()` - Pauses the media.
   - `stop()` - Stops the media.

# Chapter 4

# Implementation

Our university game project, Nanu, was successfully implemented using Java, JavaFX, SceneBuilder, and Object-Oriented Programming (OOP) principles. Java served as the foundation of our development, providing a flexible and dependable platform for creating the game's core functionalities. JavaFX, combined with SceneBuilder, provided an accessible framework for creating intuitive graphical user interfaces, which are critical for user engagement and interaction. Following OOP principles, we organized our codebase into modular and reusable components to facilitate team collaboration and ensure maintainability.

## 4.1 GUI

### 4.1.1 Start Scene


FIGURE 4.1: Start Scene

The provided FXML file describes a JavaFX user interface layout organized within an AnchorPane container, with a GridPane nestled inside to organize its components. The AnchorPane

acts as the root container, allowing for precise positioning of its child components. Various UI elements are defined in the GridPane, which is positioned with absolute coordinates within the AnchorPane. These include two Button components with distinct background and border colors, both with a SepiaTone effect. The first Button, labeled "Start," triggers an action when clicked, executing the 'switchToMainScene' method, directs to main scene. A ChoiceBox control is also included, which allows the user to select from a list of options. This ChoiceBox is styled similarly to the buttons and includes a Cursor effect, which improves user interaction. The second button calls the 'switchScene2' method in the controller class, which directs the user to instructions or additional information about how to interact with the application or game. Both buttons are designed with similar styling and effects, ensuring visual coherence and a consistent user experience throughout the interface.

### 4.1.2 Rule Scene

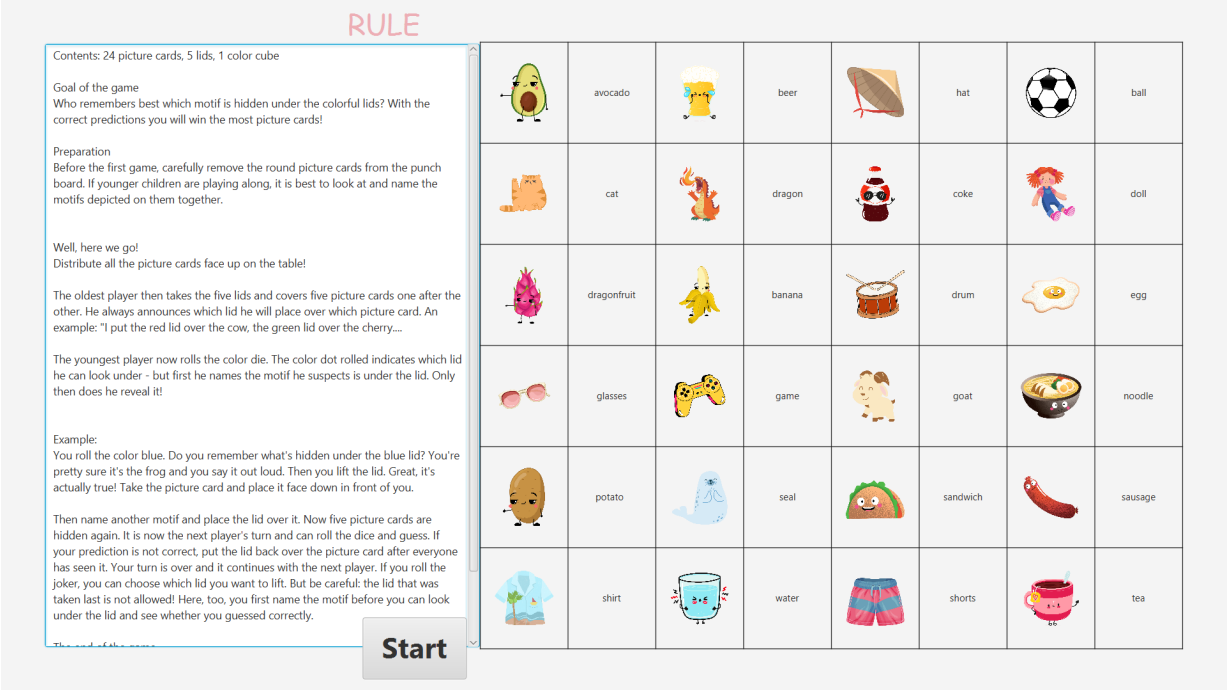


FIGURE 4.2: Rule Scene

In this scene, several key elements make gameplay and user interaction easier. The TextArea component functions as an instructional hub, providing detailed rules and guidelines for playing the game. It includes important information such as game content, objectives, preparation steps, gameplay mechanics, and end conditions, ensuring that players know how to proceed. The GridPane layout arranges a group of ImageView elements, each representing a different picture card in the game. These ImageView components show images of various game objects, allowing players to visually recognize and interact with them while playing. Labels on each ImageView provide textual descriptions or names for the depicted objects, assisting players in recognizing and associating images with corresponding game elements. Finally, the "Start" button initiates the game, letting players know that they can start playing whenever they are ready.

Together, these elements form a cohesive and informative game interface that guides players through the game while increasing engagement and enjoyment.

### 4.1.3 Main Scene

### 4.1.4 Rule Scene

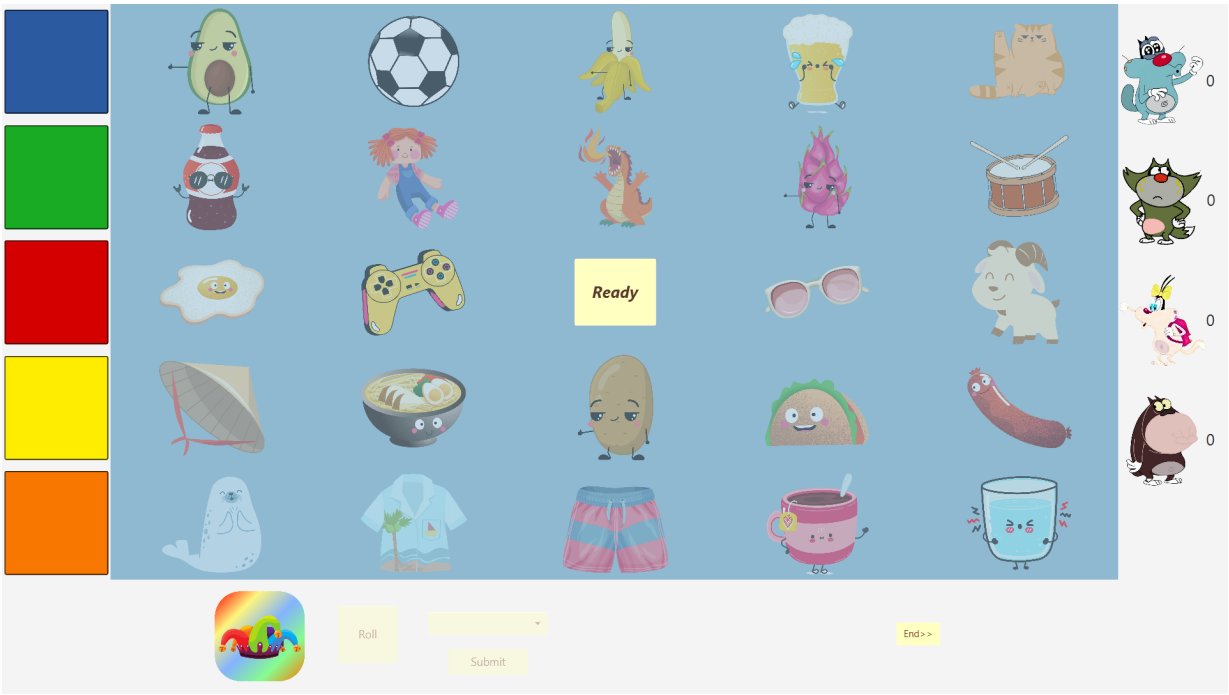


FIGURE 4.3: Main Scene

The SceneBuilder supports to represent a comprehensive JavaFX application layout designed for a game scene. At its core, the central GridPane (‘myGridPane‘) holds ImageView elements representing a wide array of game objects, each equipped with event handlers for mouse clicks, thereby enabling interactive functionality. These interactions are managed and processed by the associated controller class (‘Controller‘), facilitating dynamic responses to user inputs. Additionally, a "Ready" Button housed at the bottom StackPane initiates a shuffle action for the game elements, enhancing gameplay variability. The bottom StackPane further accommodates essential UI controls such as the roll dice Button (‘rollButton‘), choice submission Button (‘submitButton‘), a ChoiceBox (‘choiceBox2‘), and an ImageView (‘diceImage‘) for displaying rolled dice images, all tightly integrated with event handlers to capture user interactions. The left GridPane (‘myGridPane2‘) incorporates colored Rectangle elements, likely representing distinct game zones, each equipped with event handlers for user engagement. Conversely, the right Pane features Label elements (‘scoreLabel1‘, ‘scoreLabel2‘, ‘scoreLabel3‘, ‘scoreLabel4‘) for displaying player scores and ImageView elements portraying character avatars (Oggy, Jack, Olivia, Bob), effectively communicating game progress and player identities. This layout harmoniously merges visual representation with functional interactivity, providing a rich and immersive gaming experience.

### 4.1.5 Final Scene

### 4.1.6 Rule Scene


FIGURE 4.4: Final Scene

The Labels represent the names or identifiers of players in a game, with their positions dynamically determined by the scores they achieve in the main game scene. The first Label (‘fx:id="firstPlace"‘) corresponds to the player currently holding the top position in terms of score, followed by the second Label (‘fx:id="secondPlace"‘) representing the player in the second position, and the third Label (‘fx:id="thirdPlace"‘) indicating the player in the third position. These Labels likely receive their values dynamically from the game logic based on the scores obtained by each player during gameplay, ensuring real-time updates on player standings. Accompanying these Labels are trophy ImageView elements within a GridPane, creating a visually appealing representation of player rankings. Additionally, the layout features an "Exit" Button with a SepiaTone effect and an associated action handler, providing users with an option to exit the game. Overall, the XML defines an interactive and visually engaging interface for displaying game results and facilitating user interaction.

## 4.2 Application class

The provided Java code is the entry point for a JavaFX application named "Nanu?". It extends the Application class and overrides its start method to initialize the application’s main scene. The start method loads the StartGame.fxml file using FXMLLoader to set up the initial scene layout. It configures the stage with a title, disables resizing, sets it to full screen, and displays it. Additionally, it sets up an event handler to intercept the close request event and prompts the user with a confirmation dialog before closing the application. The exitGame method handles

the exit confirmation dialog and closes the stage if the user confirms. Furthermore, there's a music method responsible for playing background music using the MediaPlayer class with a specified media file. Overall, the code initializes the application, sets up the main scene, handles user exit confirmation, and plays background music during gameplay.

```
1  public class App extends Application {
2
3      public static void main(String[] args) {
4          launch(args);
5      }
6
7      @Override
8      public void start(Stage stage) {
9          try {
10             music();
11             Parent root = FXMLLoader.load(getClass().getResource("↩
                   StartGame.fxml"));
12             Scene scene = new Scene(root);
13             stage.setScene(scene);
14             stage.setTitle("Nanu?");
15             stage.setResizable(false);
16             stage.setFullScreen(true);
17             stage.show();
18             stage.setOnCloseRequest(e -> {
19                 e.consume();
20                 exitGame(stage);
21             });
22
23         } catch (Exception e) {
24             e.printStackTrace();
25         }
26     }
27
28     public void exitGame(Stage stage) {
29
30         Alert alertBox = new Alert(AlertType.CONFIRMATION);
31         alertBox.setTitle("Log out");
32         alertBox.setHeaderText("You are about to exit Nanu?");
33         alertBox.setContentText("Thanks for playing our (not so great) ↩
               game :')");
34
35         if (alertBox.showAndWait().get() == ButtonType.OK) {
36             stage.close();
37         }
38     }
39
40     MediaPlayer mediaPlayer;
```

```
41
42      public void music() {
43          String musicFile = "/Sound Effects/background.mp3";
44          Media media = new Media(getClass().getResource(musicFile).↩
                toExternalForm());
45          MediaPlayer mediaPlayer = new MediaPlayer(media);
46          mediaPlayer.setCycleCount(MediaPlayer.INDEFINITE);
47          mediaPlayer.setVolume(0.15);
48          mediaPlayer.play();
49      }
50  }
```

## 4.3 Scene Switching class

The provided Java class 'SceneSwitcher' serves as a utility for switching between different scenes in a JavaFX application. It contains methods for transitioning to the main game scene ('switchToMainScene'), a scene explaining how to play the game ('switchScene2'), and the final scene ('switchToEndScene'). Each method loads the corresponding FXML file using 'FXMLLoader' and sets up the stage with the appropriate scene. Additionally, there's an 'exitGame' method that displays a confirmation dialog when the user attempts to exit the game, and if confirmed, closes the application window. Each method takes an 'ActionEvent' parameter, typically generated by user interaction, and retrieves the source node (usually a button) to access its associated scene and stage. Overall, this class facilitates smooth scene transitions and handles the exit confirmation process within the application.

```
1   public class SceneSwitcher {
2
3       private Stage stage;
4       private Scene scene;
5       private Parent root;
6
7       public void switchToMainScene(ActionEvent event) throws IOException↩
            {
8           root = FXMLLoader.load(getClass().getResource("MainScene.fxml")↩
                );
9           stage = (Stage) ((Node) event.getSource()).getScene().getWindow↩
                ();
10          scene = new Scene(root);
11          stage.setScene(scene);
12          stage.setFullScreen(true);
13          stage.show();
14      }
15
16      public void switchScene2(ActionEvent event) throws IOException {
```

```
17          root = FXMLLoader.load(getClass().getResource("HowToPlay.fxml")↩
                ;
18          stage = (Stage) ((Node) event.getSource()).getScene().getWindow↩
                ();
19          scene = new Scene(root);
20          stage.setScene(scene);
21          stage.show();
22      }
23
24      public void switchToEndScene(ActionEvent event) throws IOException ↩
            {
25          root = FXMLLoader.load(getClass().getResource("FinalScee.fxml")↩
                ;
26          stage = (Stage) ((Node) event.getSource()).getScene().getWindow↩
                ();
27          scene = new Scene(root);
28          stage.setScene(scene);
29          stage.setFullScreen(true);
30          stage.show();
31      }
32
33      public void exitGame(ActionEvent event) throws IOException {
34          Alert alertBox = new Alert(AlertType.CONFIRMATION);
35          alertBox.setTitle("Log out");
36          alertBox.setHeaderText("You are about to exit Nanu?");
37          alertBox.setContentText("Thanks for playing our (not so great) ↩
                game :')");
38
39          if (alertBox.showAndWait().get() == ButtonType.OK) {
40              stage = (Stage) ((Node) event.getSource()).getScene().↩
                    getWindow();
41              stage.close();
42          }
43      }
44
45  }
```

## 4.4 GridPane Operator class

The ‘GridPaneOperator‘ class encapsulates a set of functionalities for managing elements within a JavaFX ‘GridPane‘. Primarily, it facilitates the shuffling of ImageView elements within the GridPane through the ‘shuffleImageViews‘ method, ensuring their random distribution without overlap. Methods like ‘isImgViewOccupied‘ and ‘isRecOccupied‘ enable the checking of cell occupancy by ImageViews or Rectangles, respectively, aiding in determining valid placement locations. Moreover, the class provides mechanisms for moving Rectangles as overlays within the

GridPane via 'moveOverlay', along with validating the success of such moves using 'moveUnSuccessfull'. Additionally, it allows for the dynamic enabling or disabling of ImageView elements with 'disableImageView' and 'ableImageView', providing control over user interaction. Lastly, the class facilitates the management of ChoiceBox and Button states through 'enableDisableAnswerAndSubmitButton', toggling their usability as required. These functionalities collectively contribute to the effective manipulation and interaction management of elements within a JavaFX GridPane, enhancing the flexibility and usability of graphical user interfaces developed with JavaFX.

```
1  public class GridPaneOperator {
2
3      private final Random random = new Random();
4      RotateTransition rotateTransition;
5
6      public void shuffleImageViews(GridPane myGridPane, List<ImageView> ↩
           imageViews) {
7          Collections.shuffle(imageViews);
8          myGridPane.getChildren().clear();
9
10         for (ImageView imageView : imageViews) {
11             int randomRow, randomCol;
12             do {
13                 randomRow = random.nextInt(5);
14                 randomCol = random.nextInt(5);
15             } while (isImgViewOccupied(myGridPane, randomRow, randomCol↩
                   ) || (randomCol == 2 && randomRow == 2));
16
17             myGridPane.add(imageView, randomCol, randomRow);
18         }
19     }
20
21     public boolean isImgViewOccupied(GridPane myGridPane, int row, int ↩
           col) {
22         for (Node node : myGridPane.getChildren()) {
23             if (GridPane.getRowIndex(node) == row && GridPane.↩
                   getColumnIndex(node) == col) {
24                 return true;
25             }
26         }
27         return false;
28     }
29
30     public boolean isRecOccupied(GridPane myGridPane, int row, int col)↩
           {
31         for (Node node : myGridPane.getChildren()) {
32             if (node instanceof Rectangle) {
```

```
33                  if (GridPane.getRowIndex(node) == row && GridPane.↩
                        getColumnIndex(node) == col) {
34                      return true;
35                  }
36              }
37          }
38          return false;
39      }
40
41      public void moveOverlay(GridPane myGridPane, int newColumn, int ↩
            newRow, Rectangle rectangle) {
42          if (isRecOccupied(myGridPane, newRow, newColumn) != true) {
43              myGridPane.getChildren().remove(rectangle);
44              GridPane.setColumnIndex(rectangle, newColumn);
45              GridPane.setRowIndex(rectangle, newRow);
46              myGridPane.getChildren().add(rectangle);
47          }
48      }
49
50      public boolean moveUnSuccessfull(GridPane myGridPane, int newColumn↩
            , int newRow, Rectangle rectangle) {
51          boolean result = true;
52          if (isRecOccupied(myGridPane, newRow, newColumn) != true) {
53              result = false;
54          }
55          return result;
56      }
57
58      public void disableImageView(GridPane myGridPane) {
59          for (Node node : myGridPane.getChildren()) {
60              if (node instanceof ImageView) {
61                  node.setDisable(true);
62              }
63          }
64      }
65
66      public void ableImageView(GridPane myGridPane) {
67          for (Node node : myGridPane.getChildren()) {
68              if (node instanceof ImageView) {
69                  node.setDisable(false);
70              }
71          }
72      }
73
74      public void enableDisableAnswerAndSubmitButton(ChoiceBox<String> ↩
            choiceBox, Button submitButton) {
75          Platform.runLater(() -> {
76              if (choiceBox.isDisable()) {
```

```
77                  choiceBox.setDisable(false);
78                  submitButton.setDisable(false);
79              } else {
80                  choiceBox.setDisable(true);
81                  submitButton.setDisable(true);
82              }
83          });
84
85      }
86
87      public void startRotationAnimation(Rectangle rectangle) {
88          if (rectangle != null) {
89              rotateTransition = new RotateTransition(Duration.seconds↩
                    (2.5), rectangle);
90              rotateTransition.setByAngle(360);
91              rotateTransition.setCycleCount(2);
92              rotateTransition.play();
93          }
94      }
95  }
```

## 4.5 Game Logic class

The GameLogic class, the check function is responsible for verifying the correctness of a user's selection during gameplay. This method systematically evaluates the user's choice by comparing it with the correct answer associated with the game element represented by the Rectangle. Upon a successful match, signified by the user selecting the correct answer from the provided options, the corresponding game element is visually hidden and disabled to prevent further interaction. The endGame function orchestrates the conclusion of the game by initiating a smooth fade-out effect on all game elements within the grid. This transition enhances the visual experience as the game concludes, gradually removing the elements from view. Additionally, the winner function determines the game's winners based on their respective scores and updates the relevant labels accordingly. By leveraging a HashMap to map player scores to their corresponding names, this function ensures accurate recognition of the top performers in the game, fostering a sense of achievement and competition among players. Together, these functions contribute to the seamless execution and immersive experience of the game logic.

```
1  public class GameLogic {
2
3      CorrectSound correctSound = new CorrectSound();
4      IncorrectSound incorrectSound = new IncorrectSound();
5      GridPaneOperator gridPaneOperator = new GridPaneOperator();
6
```

```
7       public boolean check(GridPane myGridPane, Rectangle rectangle, ↩
            ChoiceBox<String> choiceBox2) {
8
9           String answer = choiceBox2.getValue();
10          int targetRow = GridPane.getRowIndex(rectangle);
11          int targetColumn = GridPane.getColumnIndex(rectangle);
12          boolean result = false;
13
14          System.out.println(answer);
15
16          for (Node node : myGridPane.getChildren()) {
17              if (node instanceof ImageView) {
18                  if (GridPane.getColumnIndex(node) == targetColumn
19                          && GridPane.getRowIndex(node) == targetRow
20                          && answer.equals(node.getId())) {
21
22                      result = true;
23                      node.setVisible(false);
24                      node.setDisable(true);
25                  }
26              }
27          }
28          return result;
29      }
30
31      public void endGame(GridPane myGridPane) {
32          for (Node node : myGridPane.getChildren()) {
33              if (node instanceof Rectangle) {
34                  FadeTransition fadeOut = new FadeTransition(Duration.↩
                        seconds(3), node);
35                  fadeOut.setFromValue(1.0);
36                  fadeOut.setToValue(0);
37                  fadeOut.play();
38              }
39          }
40      }
41
42      public void winner(int score1, int score2, int score3, int score4, ↩
            Label first, Label second, Label third) {
43          HashMap<Integer, String> winner = new HashMap<>();
44          winner.put(score1, "Oggy");
45          winner.put(score2, "Jack");
46          winner.put(score3, "Olivia");
47          winner.put(score4, "Bob");
48
49          // Create a TreeSet with a custom comparator to sort by keys
50          TreeSet<Map.Entry<Integer, String>> sortedEntries = new TreeSet↩
                <>(
```

```
51                  Comparator.<Map.Entry<Integer, String>>comparingInt(Map↩
                        .Entry::getKey).reversed()
52          );
53
54          // Add all entries from the HashMap to the TreeSet
55          sortedEntries.addAll(winner.entrySet());
56
57          // Retrieve the entries based on their position in the set
58          Iterator<Map.Entry<Integer, String>> iterator = sortedEntries.↩
                iterator();
59
60          if (iterator.hasNext()) {
61              Map.Entry<Integer, String> firstPlace = iterator.next();
62              first.setText(firstPlace.getValue());
63          }
64
65          if (iterator.hasNext()) {
66              Map.Entry<Integer, String> secondPlace = iterator.next();
67              second.setText(secondPlace.getValue());
68          }
69
70          if (iterator.hasNext()) {
71              Map.Entry<Integer, String> thirdPlace = iterator.next();
72
73              if (thirdPlace.getKey() != 0) {
74                  third.setText(thirdPlace.getValue());
75              } else {
76                  third.setText(null);
77              }
78          } else {
79              third.setText(null);
80          }
81      }
82  }
```

## 4.6 Sound effects class

### 4.6.1 Sound class

The ‘Sound‘ class provides a generic framework for playing sound effects within the game. It utilizes JavaFX media components to handle the playback of audio files. The class contains a ‘MediaPlayer‘ object and a ‘String‘ variable to store the path of the sound file to be played. The ‘playSoundEffect‘ method takes the path of the sound file as input and initializes a new ‘Media‘ object with the specified path. It then creates a new ‘MediaPlayer‘ instance using the ‘Media‘ object and starts playback of the audio. The ‘setCycleCount‘ method configures the

number of times the sound should repeat (in this case, it's set to play once). This class serves as a foundation for playing various sound effects throughout the game, allowing for a more immersive and interactive gaming experience.

```
1  public class Sound {
2
3      MediaPlayer mediaPlayer;
4      String sound;
5
6      public void playSoundEffect(String path) {
7          Media media = new Media(getClass().getResource(sound).↩
               toExternalForm());
8          MediaPlayer mediaPlayer = new MediaPlayer(media);
9          mediaPlayer.setCycleCount(1);
10         mediaPlayer.play();
11     }
12 }
```

### 4.6.2 Dice' sound effect class

The 'DiceSound' class encapsulates functionality related to playing a sound effect associated with rolling a dice in the game. This class extends the 'Sound' class, inheriting its methods and properties. The 'playDiceSound' method is specifically tailored to play the sound effect for the dice roll. It sets the path to the sound file and invokes the 'playSoundEffect' method inherited from the 'Sound' class to initiate the playback of the sound effect. Additionally, a message is printed to the console indicating the sound file being played for debugging and verification purposes. This class serves to enhance the gaming experience by providing auditory feedback to the player when performing essential actions such as rolling the dice, contributing to the overall immersion and engagement in the game.

```
1  public class DiceSound extends Sound {
2
3      public void playDiceSound() {
4          sound = "/Sound Effects/dice.mp3";
5          playSoundEffect(sound);
6          System.out.println(sound);
7      }
8  }
```

### 4.6.3 Correct Sound Effect class

The 'CorrectSound' class extends the 'Sound' class and specializes in playing various correct sound effects during gameplay. It utilizes a 'Random' object to generate a random number between 1 and 8. Based on the generated random number, it selects one of several correct sound effects to play. The 'playCorrectSound' method switches on the random result and assigns a specific sound file path corresponding to the generated number. It then calls the 'playSoundEffect' method inherited from the 'Sound' class to play the selected sound effect. After playing the sound effect, the method prints the path of the sound file to the console. This class enhances the gaming experience by providing auditory feedback to the player upon successfully completing certain actions or achieving specific milestones within the game.

```
1  public class CorrectSound extends Sound {
2      private Random random = new Random();
3
4      public void playCorrectSound() {
5          int result = random.nextInt(8) + 1;
6          switch (result) {
7              case 1:
8                  sound = "/Sound Effects/correct1.mp3";
9                  playSoundEffect(sound);
10                 System.out.println(sound);
11                 break;
12             case 2:
13                 sound = "/Sound Effects/correct2.mp3";
14                 playSoundEffect(sound);
15                 System.out.println(sound);
16                 break;
17             case 3:
18                 sound = "/Sound Effects/correct3.mp3";
19                 playSoundEffect(sound);
20                 System.out.println(sound);
21                 break;
22             case 4:
23                 sound = "/Sound Effects/correct4.mp3";
24                 playSoundEffect(sound);
25                 System.out.println(sound);
26                 break;
27             case 5:
28                 sound = "/Sound Effects/correct5.mp3";
29                 playSoundEffect(sound);
30                 System.out.println(sound);
31                 break;
32             default:
33                 System.out.println("null dung nha");
34                 break;
35         }
```

```
36      }
37  }
```

### 4.6.4 Incorrect Sound Effect class

The 'IncorrectSound' class, also extending the 'Sound' class, is responsible for playing various incorrect sound effects during gameplay. It employs a 'Random' object to generate a random number between 1 and 8. Based on the generated random number, it selects one of several incorrect sound effects to play. The 'playInCorrectSound' method switches on the random result and assigns a specific sound file path corresponding to the generated number. It then calls the 'playSoundEffect' method inherited from the 'Sound' class to play the selected sound effect. After playing the sound effect, the method prints the path of the sound file to the console. This class contributes to the gaming experience by providing auditory feedback to the player upon making mistakes or encountering errors during gameplay.

```
1  public class IncorrectSound extends Sound {
2      private Random random = new Random();
3
4      public void playInCorrectSound() {
5          int result = random.nextInt(8) + 1;
6          switch (result) {
7              case 1:
8                  sound = "/Sound Effects/wrong1.mp3";
9                  playSoundEffect(sound);
10                 System.out.println(sound);
11                 break;
12             case 2:
13                 sound = "/Sound Effects/wrong2.mp3";
14                 playSoundEffect(sound);
15                 System.out.println(sound);
16                 break;
17             case 3:
18                 sound = "/Sound Effects/wrong3.mp3";
19                 playSoundEffect(sound);
20                 System.out.println(sound);
21                 break;
22             case 4:
23                 sound = "/Sound Effects/wrong4.mp3";
24                 playSoundEffect(sound);
25                 System.out.println(sound);
26                 break;
27             case 5:
28                 sound = "/Sound Effects/wrong5.mp3";
29                 playSoundEffect(sound);
30                 System.out.println(sound);
```

```
31                  break;
32              default:
33                  System.out.println("null sai nha");
34                  break;
35          }
36      }
37  }
```

## 4.7 Controller class

The 'Controller' class within the Nanu? game application serves as the central hub for managing user interactions and coordinating game logic. Upon initialization, the 'initialize' method is responsible for setting up the initial state of the game interface. This includes tasks such as populating choice boxes with player options and configuring UI elements to prepare the game environment. Additionally, it invokes the 'winner' method from the 'GameLogic' class to initialize the leaderboard with default scores, ensuring that the game starts with the necessary setup for player engagement.

One of the core functionalities of the 'Controller' class is handling the rolling of the dice. The 'roll' method orchestrates this process by initiating a threaded animation sequence to simulate the rolling of a dice. This animation involves generating random values and updating the dice image accordingly, providing players with a visual representation of the dice roll outcome. Furthermore, the method manages player interaction during the rolling process, ensuring smooth gameplay and accurate representation of game events.

Player interaction within the game is facilitated by various methods in the 'Controller' class. For instance, the 'returnPosImage' method is responsible for managing the selection of items from the grid. It ensures proper movement and interaction with the game environment, allowing players to make choices and progress through the game. Similarly, the 'submit' method processes player choices, updates scores based on the selections made, and manages the overall progression of the game based on player actions.

The 'nextPlayer' method plays a crucial role in maintaining the game state by managing player turns. It cycles through players, ensuring fair participation and equal opportunities for all players to make their choices. By updating player scores and managing turn-based gameplay, this method maintains the integrity of the game and ensures a balanced gaming experience for all participants.

Sound effects are integrated into the gameplay experience to provide auditory feedback to players. Methods such as 'playCorrectSound' and 'playInCorrectSound' trigger appropriate sound effects based on player actions, enhancing the overall gaming experience. These sound effects provide additional feedback to players, reinforcing correct and incorrect choices made during gameplay.

In summary, the 'Controller' class serves as the backbone of the Nanu? game application, orchestrating various aspects of gameplay, including dice rolling, player interaction, game state

management, and sound effects integration. Through effective coordination of these elements, the 'Controller' class ensures an engaging and immersive gaming experience for players.

```
1  public class Controller implements Initializable {
2      Random random = new Random();
3      static int totalScore = 0;
4
5      @FXML
6      private GridPane myGridPane;
7
8      @FXML
9      private GridPane myGridPane2;
10
11     @FXML
12     private ImageView diceImage;
13
14     @FXML
15     private ImageView avocado, ball, banana, beer, cat, coke, doll, ↩
           dragon, dragonfruit, drum, egg, games, glasses,
16             goat, hat, noodles, potato, sandwich, sausage, seal, shirt,↩
                   shorts, tea, water;
17
18     @FXML
19     private Button rollButton;
20
21     @FXML
22     private Label scoreLabel1, scoreLabel2, scoreLabel3, scoreLabel4, ↩
           firstPlace, secondPlace, thirdPlace;
23
24     @FXML
25     private Button submitButton;
26
27     @FXML
28     private ChoiceBox<String> choiceBox1;
29
30     @FXML
31     private ChoiceBox<String> choiceBox2;
32
33     @FXML
34     private Rectangle redRectangle, blueRectangle, greenRectangle, ↩
           yellowRectangle, purpleRectangle;
35     private Rectangle currentRectangle = null;
36
37     private int diceValue;
38
39     private Label currentLabel;
40
```

```
41      private static int score1 = 0, score2 = 0, score3 = 0, score4 = 0;
42
43      private String[] choiceBox1Strings = { "2 Players", "3 Players", "4↩
            Players" };
44      private String[] choiceBox2Strings = { "avocado", "ball", "banana",↩
            "beer", "cat", "coke", "doll", "dragon",
45              "dragonfruit", "drum", "egg", "game", "glasses",
46              "goat", "hat", "noodles", "potato", "sandwich", "sausage", ↩
                   "seal", "shirt", "shorts", "tea", "water" };
47
48      int n = 0;
49      private static int numberOfPlayer = 3;
50
51      GridPaneOperator gridPaneOperator = new GridPaneOperator();
52      SceneSwitcher sceneSwitcher = new SceneSwitcher();
53      GameLogic gameLogic = new GameLogic();
54      CorrectSound correctSoundEffect = new CorrectSound();
55      IncorrectSound incorrectSoundEffect = new IncorrectSound();
56      DiceSound diceSound = new DiceSound();
57
58      @Override
59      public void initialize(URL url, ResourceBundle resourceBundle) {
60          if (choiceBox1 != null) {
61              choiceBox1.getItems().addAll(choiceBox1Strings);
62              choiceBox1.setOnAction(this::getNumberOfPlayers);
63          }
64
65          if (choiceBox2 != null)
66              choiceBox2.getItems().addAll(choiceBox2Strings);
67
68          if (firstPlace != null)
69              gameLogic.winner(score1, score2, score3, score4, firstPlace↩
                       , secondPlace, thirdPlace);
70          }
71
72      public void getNumberOfPlayers(ActionEvent event) {
73          String result = choiceBox1.getValue();
74          if (result.equals("2 Players")) {
75              numberOfPlayer = 2;
76          } else if (result.equals("3 Players")) {
77              numberOfPlayer = 3;
78          } else
79              numberOfPlayer = 4;
80      }
81
82      @FXML
83      public void switchToMainScene(ActionEvent event) throws IOException↩
            {
```

```
84          sceneSwitcher.switchToMainScene(event);
85      }
86
87      @FXML
88      public void switchScene2(ActionEvent event) throws IOException {
89          sceneSwitcher.switchScene2(event);
90      }
91
92      @FXML
93      public void switchToEndScene(ActionEvent event) throws IOException ↩
            {
94          sceneSwitcher.switchToEndScene(event);
95      }
96
97      @FXML
98      public void exitGame(ActionEvent event) throws IOException {
99          sceneSwitcher.exitGame(event);
100     }
101
102     public void roll(ActionEvent e) {
103         currentRectangle = null;
104         diceSound.playDiceSound();
105         rollButton.setDisable(true);
106
107         Thread animationThread = new Thread(() -> {
108             try {
109                 for (int i = 0; i < 24; i++) {
110                     diceValue = random.nextInt(6) + 1;
111                     Image image = new Image(getClass().↩
                            getResourceAsStream("/roll/" + diceValue + ".png↩
                            "));
112
113                     Platform.runLater(() -> diceImage.setImage(image));
114                     Thread.sleep(50);
115                 }
116
117                 switch (diceValue) {
118                     case 1:
119                         gridPaneOperator.startRotationAnimation(↩
                                blueRectangle);
120                         break;
121                     case 2:
122                         gridPaneOperator.startRotationAnimation(↩
                                yellowRectangle);
123                         break;
124                     case 3:
125                         gridPaneOperator.startRotationAnimation(↩
                                greenRectangle);
```

```
126                         break;
127                     case 4:
128                         gridPaneOperator.startRotationAnimation(↩
                               purpleRectangle);
129                         break;
130                     case 5:
131                         gridPaneOperator.startRotationAnimation(↩
                               redRectangle);
132                         break;
133                     case 6:
134                         currentRectangle = null; // Reset ↩
                               currentRectangle
135 
136                         // Set the event handler for moving the ↩
                               rectangle
137                         myGridPane.getChildren().forEach(node -> {
138                             if (node instanceof Rectangle) {
139                                 Rectangle rectangle = (Rectangle) node;
140                                 rectangle.setOnMouseClicked(mouseEvent ↩
                                       -> {
141                                     currentRectangle = rectangle;
142                                     gridPaneOperator.↩
                                           startRotationAnimation(↩
                                           currentRectangle);
143                                     System.out.println(currentRectangle↩
                                           );
144                                 });
145                             }
146                         });
147                         break;
148                 }
149 
150                 gridPaneOperator.enableDisableAnswerAndSubmitButton(↩
                       choiceBox2, submitButton);
151             } catch (Exception ex) {
152                 ex.printStackTrace();
153             }
154         });
155 
156         animationThread.start();
157     }
158 
159     public void returnPosImage(MouseEvent event) {
160         // Get the source node of the MouseEvent
161         Node source = (Node) event.getSource();
162 
163         // Fetch the row and column indices of the ImageView in the ↩
               GridPane
```

```
164         Integer row = GridPane.getRowIndex(source);
165         Integer col = GridPane.getColumnIndex(source);
166 
167         // Check if a rectangle in myGridPane2 is clicked
168         if (source instanceof Rectangle && ((Rectangle) source).↩
                getParent() == myGridPane2) {
169             currentRectangle = (Rectangle) source;
170         } else if (currentRectangle != null) {
171             // If a rectangle is selected and a cell in myGridPane is ↩
                clicked
172             gridPaneOperator.moveOverlay(myGridPane, col, row, ↩
                currentRectangle);
173             if (!gridPaneOperator.moveUnSuccessfull(myGridPane, col, ↩
                row, currentRectangle)) {
174                 currentRectangle = null; // Reset selected rectangle ↩
                    after moving
175             }
176         }
177     }
178 
179     public void submit(ActionEvent event) {
180         try {
181             switch (diceValue) {
182                 case 1:
183                     currentRectangle = blueRectangle;
184                     break;
185                 case 2:
186                     currentRectangle = yellowRectangle;
187                     break;
188                 case 3:
189                     currentRectangle = greenRectangle;
190                     break;
191                 case 4:
192                     currentRectangle = purpleRectangle;
193                     break;
194                 case 5:
195                     currentRectangle = redRectangle;
196                     break;
197             }
198             handleUserChoice();
199 
200             if (totalScore == 20) {
201 
202                 PauseTransition pause = new PauseTransition(Duration.↩
                    seconds(3));
203                 pause.setOnFinished(e -> {
204                     gameLogic.endGame(myGridPane);
205                 });
```

```
206                 pause.play();
207             } else {
208                 rollButton.setDisable(false);
209 
210             }
211 
212         } finally {
213             gridPaneOperator.enableDisableAnswerAndSubmitButton(↩
                    choiceBox2, submitButton);
214         }
215     }
216 
217     private void handleUserChoice() {
218         nextPlayer(numberOfPlayer);
219 
220         if ((currentRectangle != null && gameLogic.check(myGridPane, ↩
                currentRectangle, choiceBox2))) {
221 
222             totalScore++;
223             System.out.println(totalScore);
224             if (currentLabel.getId().equals("scoreLabel1")) {
225                 score1++;
226                 currentLabel.setText(": " + score1);
227 
228             } else if (currentLabel.getId().equals("scoreLabel2")) {
229                 score2++;
230                 currentLabel.setText(": " + score2);
231             } else if (currentLabel.getId().equals("scoreLabel3")) {
232                 score3++;
233                 currentLabel.setText(": " + score3);
234             } else if (currentLabel.getId().equals("scoreLabel4")) {
235                 score4++;
236                 currentLabel.setText(": " + score4);
237             }
238 
239             correctSoundEffect.playCorrectSound();
240             gridPaneOperator.ableImageView(myGridPane);
241             myGridPane2.getChildren().remove(currentRectangle);
242 
243         } else {
244 
245             incorrectSoundEffect.playInCorrectSound();
246             gridPaneOperator.disableImageView(myGridPane);
247 
248             FadeTransition fadeOut = new FadeTransition(Duration.↩
                    seconds(2), currentRectangle);
249             fadeOut.setFromValue(1.0); // Full opacity
250             fadeOut.setToValue(0.5);
```

```
251
252             FadeTransition fadeIn = new FadeTransition(Duration.seconds↩
                    (2), currentRectangle);
253             fadeIn.setFromValue(0.5); // Half opacity
254             fadeIn.setToValue(1.0);
255
256             // Set the onFinished event for fadeOut to start fadeIn ↩
                    after fadeOut finishes
257             fadeOut.setOnFinished(e -> {
258                 fadeIn.play();
259                 currentRectangle = null; // Set to null after the fade-↩
                        in animation
260             });
261
262             fadeOut.play();
263         }
264     }
265
266     @FXML
267     private void shuffle(ActionEvent event) {
268         rollButton.setDisable(false);
269         redRectangle.setDisable(false);
270         blueRectangle.setDisable(false);
271         yellowRectangle.setDisable(false);
272         purpleRectangle.setDisable(false);
273         greenRectangle.setDisable(false);
274
275         List<ImageView> imageViews = new ArrayList<>();
276
277         // Collect ImageView elements from the GridPane
278         for (Node node : myGridPane.getChildren()) {
279             if (node instanceof ImageView) {
280                 node.setOpacity(1);
281                 imageViews.add((ImageView) node);
282             }
283         }
284
285         // Shuffle the positions of ImageView elements
286         gridPaneOperator.shuffleImageViews(myGridPane, imageViews);
287     }
288
289     public void nextPlayer(int numberOfPlayer) {
290         int turn = n % numberOfPlayer;
291
292         switch (turn) {
293             case 0:
294                 currentLabel = scoreLabel1;
295                 break;
```

```
296                 case 1:
297                     currentLabel = scoreLabel2;
298                     break;
299                 case 2:
300                     currentLabel = scoreLabel3;
301                     break;
302                 case 3:
303                     currentLabel = scoreLabel4;
304                     break;
305             }
306             n++;
307         }
308 }
```

# Chapter 5

# Running & Debugging

Efficiently running and debugging a game is a critical aspect of the development process, ensuring that players can enjoy a seamless and immersive gaming experience. In this section, we delve into the intricacies of launching and troubleshooting "Nanu". From understanding the system requirements to navigating the user interface, mastering the art of running the game sets the foundation for an enjoyable gaming journey. Additionally, robust debugging practices play a pivotal role in identifying and resolving issues that may hinder gameplay, ultimately elevating the overall quality and performance of the game.

## 5.1 Platform.runLater()

When implementing the roll button, our team encountered unforeseen issues by not utilizing Platform.runLater() to update the user interface on the JavaFX application thread. The absence of this crucial mechanism led to undesirable consequences, including occasional UI freezes and unresponsiveness during the dice rolling process.

Platform.runLater() would have ensured that UI updates occurred seamlessly on the JavaFX application thread, mitigating these issues and providing a more stable and responsive user experience.

```
1     Platform.runLater(() -> diceImage.setImage(image));
```

## 5.2 Initializable

In JavaFX, implementing the Initializable interface in a controller class allows for the automatic invocation of the initialize method after the associated FXML file is loaded. This method serves as a standardized entry point for developers to execute initialization code specific to the controller, streamlining the setup of the initial user interface state.

In JavaFX, when working with FXML, the initialize method is automatically called after the associated FXML file is loaded, and @FXML-annotated fields are injected. The conditions in your code serve as a defensive mechanism, ensuring that the referenced UI components (choiceBox1, choiceBox2, and firstPlace) are not null before attempting to perform operations on them.

```
1      @Override
2      public void initialize(URL url, ResourceBundle resourceBundle) {
3          if (choiceBox1 != null) {
4              choiceBox1.getItems().addAll(choiceBox1Strings);
5              choiceBox1.setOnAction(this::getNumberOfPlayers);
6          }
7          // other code
```

Not using these if conditions when accessing UI components leads to a NullPointerException, particularly if the corresponding FXML elements are not present or have failed to be injected during the initialization process.

## 5.3 Preventing Misspelling in Player Input

An early issue in our game involved players typing answers character by character, leading to potential misspelling and scoring challenges. To mitigate this, we implemented a two-fold solution. Firstly, we introduced a list of all possible items for players to choose from, preventing misspelling. Secondly, we transformed the answer textfield into a choicebox, allowing players to select answers from predefined options, completely eliminating the risk of misspelling.

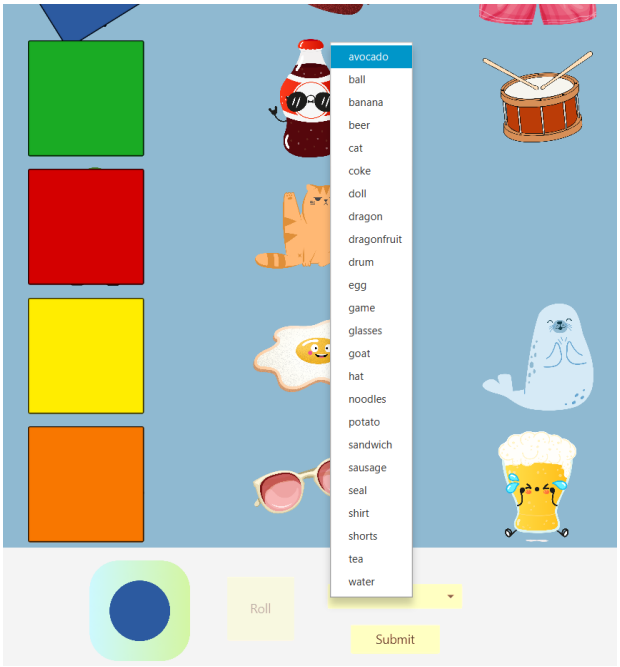


FIGURE 5.1: Choice Box

## 5.4 Gridlines

The setGridLinesVisible method in JavaFX's GridPane class, marked "For debug purposes only," offers developers a valuable tool during software development. Enabling this feature renders

visible lines on the user interface, outlining the grid's rows and columns, aiding in the swift identification and resolution of layout issues.

However, it's important to note that this functionality is meant exclusively for debugging and should be disabled in production. The annotation serves as a reminder to developers to use it judiciously, ensuring a clean and polished user interface for the end-users.

## 5.5 Lids (Cover)

In our game, a "lid" is implemented as a rectangle. When moving a rectangle to another position, users can inadvertently place it in a position where there is already another rectangle. To address this issue, we have implemented a method to handle this bug.

```
1     public boolean isRecOccupied(GridPane myGridPane, int row, int col)↩
          {
2         for (Node node : myGridPane.getChildren()) {
3             if (node instanceof Rectangle) {
4                 if (GridPane.getRowIndex(node) == row && GridPane.↩
                      getColumnIndex(node) == col) {
5                     return true;
6                 }
7             }
8         }
9         return false;
10    }
```

## 5.6 Static Variables

When a variable is declared as static in Java, a single instance of the variable is created and shared among all objects of that class at the class level. This makes static variables essentially global variables, as they are not tied to a specific instance but rather belong to the class itself.

In the context of JavaFX development, consider the scenario where a static variable like numberOfPlayer is used to keep track of a certain value, such as the number of players in a game.

```
1     private static int numberOfPlayer = 3;
2     private static int score1 = 0, score2 = 0, score3 = 0, score4 = 0;
```

Without the static keyword, each scene or instance would have its own copy of numberOfPlayer, and switching scenes leads to the variable getting reset, causing undesired behavior or bugs in the application.

Moreover, numberOfPlayer is set to 3, which is the default value in our game, as there are 3 members in our team.

## 5.7 Missing Row, Column Indices

This XML snippet represents a section of a JavaFX FXML file, defining an ImageView element.

```
1     <ImageView id="seal" fx:id="seal" fitHeight="126.0" fitWidth="131.0↩
         " onMouseClicked="#returnPosImage" opacity="0.5" pickOnBounds="↩
         true" preserveRatio="true" GridPane.halignment="CENTER" GridPane↩
         .rowIndex="4" GridPane.valignment="CENTER">
2     </ImageView>
```

However, when using Scene Builder, our team encounters a problem with JavaFX FXML files. The row indices and column indices of ImageView are sometimes not specified, even though in Scene Builder, our team has already dragged and dropped the ImageView into a cell of a GridPane. The ImageView should have row and column indices, but in our case, they are missing.

If the GridPane.rowIndex attribute is not explicitly set in the FXML code or in the associated controller, calling GridPane.getRowIndex will return null. This situation can lead to unexpected behavior or potential NullPointerExceptions.

## 5.8 Scalability

When using Scene Builder for GUI application development in JavaFX, adjusting the scene size may not automatically scale its components. Therefore, our team opts to have our game automatically run in fullscreen mode.

# Chapter 6

# Improvement

Throughout this project, we have done a lot of improvements to our NANU? game. Some initial versions of it could not run and contained many flaws and errors. Thanks to Prof. Logofatu pointing out some improvement ideas, we have managed to make our game even better. More fun to play as well!

## 6.1 Addition of Music and Sound Effects

To create a more immersive gaming experience, we added music, correct answer, and incorrect answer sound effects. These audio elements contribute to a more engaging and enjoyable gameplay environment. After all, that is how games are supposed to be!

## 6.2 Inclusion of "How to Play" Scene

Recognizing the importance of clear instructions, we added a "How to Play" scene where players can learn the game's rules and mechanics. This addition facilitates a smoother onboarding experience for players.

## 6.3 Flexible Player Count with "2-4 Players" ChoiceBox

Understanding the varying preferences for the number of players, we introduced a "2-4 Players" choiceBox in the main game scene. This feature provides players with the flexibility to choose the number of participants, enhancing user experience and accommodating different gaming scenarios.

# Chapter 7

# Conclusion

## 7.1 Team Work

Our team demonstrated an excellent level of collaboration throughout the project. 3 members supported each other effectively even when things did not go well. We held one another accountable and shared a strong commitment to achieving the project goals. In every phase of this NANU? project (coding, UI and poster design, project presentation and report), we showed up, worked together to solve problems, and gave constructive feedbacks to each other. Emotional support was most important to us, fostering a sense of unity and motivation.

## 7.2 Learning Experience

This group project offered us worthwhile educational opportunities in a number of fields. With all of our knowledge about object-oriented programming, or OOP, at our disposal, we can use JavaFX to create graphical user interfaces. We are also investing a great deal of work into creating the game logic, creating a captivating gameplay environment, and including extras like music and sound effects in order to improve the overall user experience. We also got to create a unique game commercial in the vein of a newspaper!


FIGURE 7.1: Breaking News!!

## 7.3 Future Prospects

### 7.3.1 Enhanced Interactivity

- Increase user engagement with additional interactive features.
- Incorporate complex game mechanics for deeper gameplay.

### 7.3.2 Advanced Algorithms

- Implement sophisticated algorithms for pattern recognition and difficulty scaling.
- Personalize the gaming experience with adaptive learning algorithms.

### 7.3.3 Multiplayer Functionality

- Introduce real-time multiplayer capabilities for competition or collaboration.
- Implement a leaderboard system to track high scores.

### 7.3.4 Expanding Content

- Add more levels, challenges, and themes to keep the game fresh.
- Allow for user-generated content and community-driven expansions.

### 7.3.5 Accessibility Features

- Ensure inclusivity with accessibility features.
- Provide customization options for difficulty levels and interface settings.

### 7.3.6 Integration with External Services

- Enhance user connectivity by integrating with social media and gaming communities.

By implementing these enhancements, our NANU? digital board game can offer an enriched gaming experience and potentially expand its user base.

# Chapter 8

# Reference

## 8.1 Learning Resources

1. **Ravensburger Mitbringspiel Nanu?**:
   https://www.digitalo.de/products/843174
2. **NANU? Children Board Game Simulation 1**:
   https://youtu.be/A_bEx2lpkmo
3. **NANU? Children Board Game Simulation 2**:
   https://youtu.be/IaH90iDNWi4
4. **NANU? Children Board Game Simulation 3**:
   https://youtu.be/dkwNihodVnw
5. **JavaFX Oracle Documentation**:
   https://docs.oracle.com/javase/8/javase-clienttechnologies.htm
6. **BroCode JavaFX Tutorials**:
   https://www.youtube.com/playlist?list=PLZPZq0r_RZOM-8vJA3NQFZB7JroDcMwev

## 8.2 Project Tools

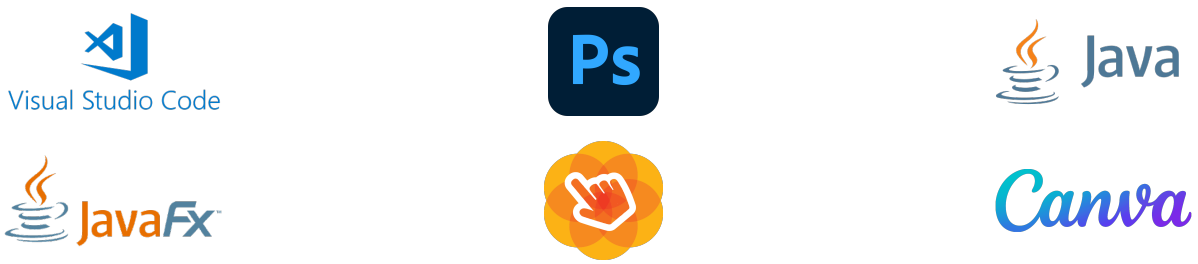


FIGURE 8.1: TOOLS