Thai Duy Dinh - 1519235
Quang Tien Doan - 1522862
Man Dat Nguyen - 1517103

Zicke Zacke - Hühnerkacke

(Winter Semester 2023-2024)

Course Name: OOP/Java
Instructor's: Professor Dr.Doina Logofatu

February 9, 2024

Springer Nature

Contents

Chapter 1
Project at a Glance

- **Project Title:** Design and Implementation a Board Game by using Object - Oriented Programming in Java
- **Platform:** Java Development Kit
- **Back-end:**In the back-end development of the game, the team utilizes Java's object-oriented programming capabilities to implement the game logic, handle player actions, and manage the game state. This involves designing classes, defining data structures, and implementing algorithms to ensure smooth gameplay.
- **Front-end:** For the front-end development, the team focuses on creating an engaging user interface (UI) and visually appealing graphics. The game board, player avatars, buttons, and other interactive elements are designed and implemented using Java's graphical user interface (GUI) libraries.
- **Team Member:**
  - Thai Duy Dinh
  - Quang Tien Doan
  - Man Dat Nguyen
- **Development Tools:** Visual Code Studio, Adobe Photoshop, Adobe Illustrator, Github Desktop, ...
- **Knowledge used:** Java, Java Swing, Java AWT, Object-Oriented Concept, Project Management,...
- **Project Duration:** 2 months
- **Guided By:** Dr. Doina Logofatu
- **Submitted To:** Fauculty of Engineering, Frankfurt University of Applied Science

Chapter 2
Introduction

In the world of gaming, board games continue to captivate players of all ages with their strategic game play and social interactions. These games often provide a fun and engaging way to challenge our minds and bond with friends and family. Among the countless board games available, 2D board games hold a special place, offering a diverse range of themes and mechanics that appeal to a wide audience.

In this project, we aim to develop a 2D board game, which is Zicke Zacke Huehnerkacke, using Java Object-Oriented Programming (OOP) principles. By leveraging the power of Java's OOP paradigm, we can create a modular and extensible game structure that promotes code reusability and maintainability. The project will focus on implementing the core mechanics of the game, designing an intuitive user interface, and providing an immersive gaming experience.

Throughout the development process, we will encounter various challenges and considerations that are common when building board games. These include designing the game board, defining the rules and game play mechanics, implementing turn-based interactions, managing player interactions, and incorporating graphical elements to enhance the visual appeal of the game.

By the end of this project, we aim to have a fully functional 2D board game implemented in Java, showcasing our understanding of OOP principles and game development concepts. Through this endeavor, we hope to not only enhance our programming skills but also provide players with an enjoyable and immersive gaming experience.

Chapter 3
Project Management

The project goes through various phases, including reviewing OOP concepts, identifying objects, creating classes, defining class relationships, implementing game logic, and finalizing the user interface (UI) and game state. We have also conducted tests, debugging, evaluations, and is currently working on finalizing the project report and presentation materials. The detailed projects management you can see details in the figure.

In order to work efficiently in a group, we use some tools and applications:

- Google Workspace: Google Drive is useful for documents management.
- Github: We use Github in order to code and manage code for this project.
- Overleaf: This website is used for final report, writing by Latex.
- Canva: This website is useful for making presentation.

Number	Name	Contribution
1	Thai Duy Dinh	Organize my team as a leader to ensure the deadlines, brainstorm idea. Coordinate with my friend to create the UI and Game Logic. Test game, debug, making the poster and finalize the documents.
2	Man Dat Nguyen	I implement the game and test it in the console. Furthermore, I cooperate with my teammates to create the UI in code and the Game Logic. Take part in debuging and testing the final game.
3	Quang Tien Doan	Almost the UI(Graphics) of the game by using Adobe PhotoShop and Illustrator. I also cooperate with my team to implement game logic, which is the overtaken part and its graphics. I also spent lots of time test game and debug it so that the game can run smoothly.

Table 3.1 The contribution of Grieffingdoor's members

ZICKE ZACKE

Content all-in-one link: Content all-in-one

Report link: Report

Github Link: https://github.com/dqtienxdd/JAVA_PROJECT_1

Evaluation Form: https://forms.gle/YP4tQ2hw2WbxrA269

Timeline

Phase	Content	Deadline	Status	Contribution		
				Duy	Tien	Dat
1	Review OOP and related stuffs	5.12	**Done**	x	x	x
	Define the Game Rules		**Done**	x	x	x
	Timeline of work		**Done**	x		
2	Identify the Objects	12.12	**Done**			x
	Create Classes		**Done**		x	
	Define Class Relationships		**Done**	x	x	x
	Create Project using Git		**Done**	x		
3	**Implementation Console and UI Preparation**					
	Related Image and Objects	19.12	**Done**		x	
	Game Implementation on Console	19.12	**Done**			x
	Implementation animation for related objects					
	Chickens	19.12	**Done**	x		
	Octagons	19.12	**Done**		x	

Fig. 3.1 Project Management Timeline

4	**Synchronize the game logic into the code with UI (according the console)**					
	Movement of chicken when it reach the exact octagon image with the next tile	7.1	**Done**	x		x
	Take over function	7.1	**Done**		x	x
	Winning Condition	7.1	**Done**			
5	**Finalize UI and Game State**					
	Add choose how many players UI	20.1	**Done**	x		x
	Add winning state UI	20.1	**Done**		x	x
	Add Instruction	20.1	**Pending**			x
6	**Test**					
	Test 0: UI appearance and game state	27.1	**Done**	x	x	x
	Test 1: Movement of chickens	27.1	**Done**	x	x	x
	Test 2: Take over function	27.1	On-going	x	x	x
	Debug					
	Debug 0: Mitigate the useless function	4.2	**Done**			x
	Debug 1: Take over function	4.2	On-going		x	
7	**Evaluation and Trials**					
	Inviting friends to play and evaluate	4.2	On-going	x	x	x
	Finalize evaluation and improve for the final	4.2	On-going	x	x	x
8	Finalize Report	6.2	On-going	x	x	x
	Slide for Presentation/Poster	6.2	On-going	x	x	x

Fig. 3.2 Project Management Timeline (cont.

Chapter 4
Problem Description

Zicke Zacke Huehnerkacke is a popular German children's board game that involves memory, strategy, and a little bit of luck. The game is designed for 2-4 players and is suitable for children aged 4 and above.

1. **Game Components:**
   - 24 Egg shaped track tiles.
   - 12 Octagonal Chicken yard tiles
   - 4 Chickens, 2 hens and 2 cocks.
   - 4 Tailfeathers

Fig. 4.1 12 Egg Shaped Track Tiles (Each will be double to make a complete map.)

Fig. 4.2 12 Octagonal Chicken Yard Tiles

2. **Game Objective:**
   The objective of Zicke Zacke Huehnerkacke is to be the first player to collect four chicken tokens of different color.
3. **Rules:**
   - **Setting Up:**
     Shuffle the 12 octagonal tiles and place them face down in the middle of the table. The 24 egg-shaped tiles are positioned around them to form a circular track. Each player chooses a chicken, puts a tail feather into one of the holes in the chicken' rump, and places it on one of the egg shaped tiles. The chickens

should be arranged so that they are spaced as evenly as possible, i.e. with four hens, there should be 5 free tiles between each chicken.

Fig. 4.3 Game Set Up

- **Movement:**
  The chickens move on the egg shaped tiles in a clockwise direction around the circular track. The youngest chick starts. They first look at the next space in front of their chicken. Then they take one of the octagonal tiles from the centre of the table and show it to all the players. If the picture on the tile they choose is the same as the picture on the next space, they may move forward onto that space. The octagonal tile is replaced face down in it's original position. The player continues to move their chicken, as long as they keep on choosing the right tile to match the next space on the track. If they make a mistake, their chicken stays on the tile that it has reached, and the next player has a go. In this way the chickens hop from tile to tile, clockwise around the track.
- **Overtaking:**
  If a chicken catches up with another chicken, they may attempt to overtake them. This time, instead of finding the matching tile for the next space, they must try to find the matching tile for the space in front of the chicken they want to overtake. If they succeed, they jump in front of the other chicken, and

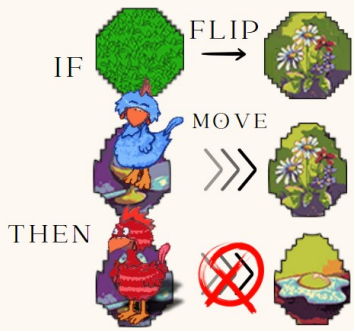

Fig. 4.4 Movement

steal all that chicken's tail feathers as they do so. The player's turn does not end at this point. They may attempt to move further by finding the matching tile for the next space, in the example in the rules the fried egg tile, and if they succeed in finding it, they move on and continue as normal.
A chicken may overtake more than one other chicken in one go if they are all standing in an unbroken row as in the example on page 4 of the rules. In this example, the rearmost chicken could jump over both the chickens in front of them, if they can find the snail tile.

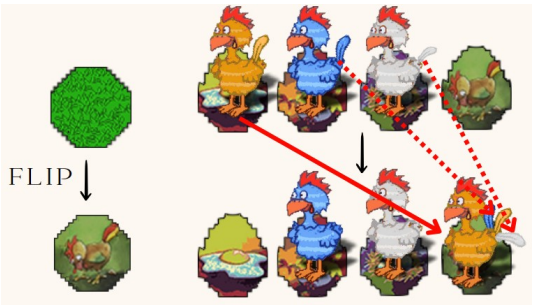

Fig. 4.5 Overtaking

- **Winning Condition:**
  The first chicken to collect all the tail-feathers is the winner.

4. **Examples:**
   Let's say there are three players: Alice, Bob, and Carol. They have set up the game board with the circular track and placed their chickens on the egg-shaped tiles, spaced evenly.

   a. Alice is the youngest, so she starts the game. She looks at the next space in front of her chicken and selects an octagonal tile from the center of the table.

Let's say the tile shows a picture of a carrot. Alice checks the next space on the track, and if it also has a carrot picture, she can move her chicken forward onto that space. She successfully finds the matching tile, places it face down in its original position, and moves her chicken to the next space.

b. Bob is next. He looks at the next space in front of his chicken and selects an octagonal tile. Suppose the tile shows a picture of a mushroom. However, the next space on the track does not have a mushroom picture, so Bob's chicken stays on the current tile, and his turn ends.
c. Now it's Carol's turn. She looks at the next space in front of her chicken and chooses an octagonal tile. Let's say the tile shows a picture of a snail. Carol checks the next space on the track and finds that it has a snail picture. She successfully matches the tile, moves her chicken forward, replaces the tile face down, and continues her turn.
d. It's Alice's turn again. She looks at the next space in front of her chicken and selects an octagonal tile. Suppose the tile shows a picture of a carrot, matching the next space. Alice moves her chicken forward, replaces the tile face down, and continues playing.
e. Bob takes his turn. He looks at the next space and chooses an octagonal tile with a mushroom picture. Unfortunately, the next space on the track does not have a mushroom picture, so Bob's chicken remains on the current tile.
f. Carol continues her turn. She looks at the next space, picks an octagonal tile with a snail picture, successfully matches it, moves her chicken forward, and replaces the tile face down.
g. The game proceeds with each player taking turns, following the same rules of finding matching tiles and moving their chickens along the track.
   At some point, one of the players, let's say Alice, successfully overtakes another player's chicken. Instead of finding the matching tile for the next space in front of her chicken, she finds the matching tile for the space in front of the chicken she wants to overtake. If she succeeds, she jumps in front of the other chicken, steals all its tail feathers, and continues her turn.
h. The players continue playing, hopping from tile to tile, matching the tiles to the next spaces on the track, and attempting to overtake other chickens if possible.
i. The game continues until one player collects all the tail feathers from the other chickens. The first player to collect all the tail feathers is declared the winner of Zicke Zacke Huehnerkacke.

Zicke Zacke Huehnerkacke is a fun and engaging game that combines memory, strategy, and a touch of luck. It's an enjoyable option for children and families to play together.

Chapter 5
Related Work

5.1 ”Zicke Zacke Hühnerkacke” Board Game

In this section, provide an overview of the original physical board game, ”Zicke Zacke Hühnerkacke.” Discuss the game’s rules, mechanics, and components. Analyze how the game encourages player interaction, strategizing, and the element of chance. Highlight the unique aspects of the game that make it enjoyable and engaging for players.

5.2 Digital Board Game Implementations

Explore existing digital implementations of board games that share similarities with your ”Zicke Zacke Hühnerkacke” game. Discuss examples of successful digital adaptations, such as ”Catan Universe,” ”Ticket to Ride Online,” or ”Tabletop Simulator.” Analyze how these digital implementations capture the essence of the original board games, provide an immersive experience, and offer multiplayer functionalities. Examine how these games handle user interface design, game flow, and player engagement.

5.3 Game Development Libraries and Frameworks

Discuss game development libraries or frameworks that can assist in the implementation of your ”Zicke Zacke Hühnerkacke” game. Explore popular libraries like Unity or frameworks like LibGDX, Java Swing. Describe the resources, tutorials, and examples available in these libraries or frameworks that can be leveraged to enhance your implementation. Discuss the benefits and challenges of using these

tools in the context of your project. For this project, we decided to work with Java Swing Library.

5.4 Graphic Design

Highlight the importance of graphic design in board game implementations. Discuss the role of visual elements, such as game board design, card illustrations, and game piece aesthetics, in enhancing the overall player experience. Explore graphic design principles and techniques that can be applied to the graphical elements of your ”Zicke Zacke Hühnerkacke” game. Discuss how effective graphic design can contribute to the game’s visual appeal and the ease of understanding for players.
In this project we use Adobe Photoshop and Adobe Illustrator to make our game components.

5.5 Team Work

Reflect on the collaborative efforts and teamwork involved in the development of your ”Zicke Zacke Hühnerkacke” game. Discuss the roles and responsibilities of team members, the communication and coordination techniques employed, and the challenges faced during the project. Highlight the lessons learned in project management, effective teamwork, and overcoming obstacles as a cohesive team.

Chapter 6
Proposed Approach

6.1 Work Structure Prosposed Aproach

We create the game the Zicke Zacke Hühnerkacke game using Object-Oriented Programming (OOP) in Java involves breaking down the problem into several classes and defining their properties and behaviors. Here is a step-by-step approach:

1. **Create timeline and environment for team work**
2. **Identify the key components of the game:** Analyze the game's rules and mechanics to identify the main components involved, such as the game board, chicken, feather, players, octagonal yard tiles, etc.
3. **Design class structures:** Create a class for each identified component, defining their properties (attributes) and behaviors (methods). Consider the relationships between these classes and how they interact with each other.
4. **Implement each class and game logic using console:** Write the code for each class and implement the game logic using the console as the user interface. Test the functionality of each class and ensure they interact correctly.
5. **Implement each class and the game logic interacting with GUI:** Modify the code to incorporate a graphical user interface (GUI) for a more user-friendly experience. Utilize frameworks like Java AWT and Java Swing to create the GUI elements and handle user interactions.
6. **Test the game and Debug:** Test the game thoroughly to identify any bugs or issues. Debug the code to fix any errors or unexpected behavior. Verify that the game functions as intended and provides a smooth user experience.
7. **Refine and enhance:** Review your implementation, test for any potential issues or bugs, and refine the code as necessary. Consider adding additional features or improvements to enhance the gaming experience.

6.2 Game Entity and Algorithm Proposed Approach

For the game logic and interacting with the game entity, we propose this algorithm. However, it can use for the console effectively. When showing the game Entity by component in the GUI, we need to change and update the garphics time by time.

```
class Chicken:
    private number
    private name
    private tailFeathers: List of strings

    constructor(number, name, feather):
        set number = number
        set name = name
        initialize tailFeathers as an empty list
        call addTailFeather(feather)

    method getNumber():
        return number

    method getName():
        return name

    method getTailFeathers():
        return tailFeathers

    method addTailFeather(feather):
        add feather to tailFeathers list

    method removeTailFeathers():
        clear tailFeathers list

class Game:
    private octagonalTiles: List of strings
    private eggShapedTiles: List of strings
    private chickens: List of Chicken objects
    private tailFeathers: List of strings
    private currentPlayer: integer
    private board: Map of Chicken to integer

    constructor():
        initialize octagonalTiles as an empty list
        add octagonal tiles to octagonalTiles list
        initialize eggShapedTiles as an empty list
        for each tile in octagonalTiles:
            add tile to eggShapedTiles list twice
        initialize chickens as an empty list
        initialize tailFeathers as an empty list
        add tail feathers to tailFeathers list
        set currentPlayer = 0
        initialize board as an empty map

    method setup():
```

```
48          shuffle octagonalTiles
49          shuffle eggShapedTiles
50          for i from 0 to 3:
51              create a new Chicken object with number i + 1, name "
                    Chicken " + tailFeathers[i], and tail feather
                    tailFeathers[i]
52              add the Chicken object to the chickens list
53              place the Chicken object on the board with index i *
                    6
54
55      method printBoard():
56          print "Board:"
57          for each chicken in chickens:
58              print chicken.getName() + " - Space " + (board.get(
                    chicken) + 1) + " - Feather: " + chicken.
                    getTailFeathers()
59
60      method play():
61          create a new Scanner object
62          loop while true:
63              set currentChicken = chickens[currentPlayer]
64              print "PLAYER " + (currentPlayer + 1) + ", it's your
                    turn."
65              print "CURRENT CHICKEN: " + currentChicken.getName()
66              set correctTile = true
67              loop while correctTile is true:
68                  call printBoard()
69                  set currentSpace = board.get(currentChicken)
70                  set nextTile = eggShapedTiles[currentSpace]
71                  print "Next tile: " + nextTile
72                  print "Guess the number of the octagonal tile (1
                        to 12): "
73                  set guessedNumber = scanner.nextInt()
74                  call scanner.nextLine() to consume the newline
                        character
75                  if guessedNumber is between 1 and the size of
                        octagonalTiles:
76                      set chosenTile = octagonalTiles[guessedNumber
                             - 1]
77                      if chosenTile is equal to nextTile:
78                          set nextSpace = (currentSpace + 1) % size
                                 of eggShapedTiles
79                          set nextChicken = getChickenAtSpace(
                                nextSpace)
80                          if nextChicken is not null:
81                              create an overtakenChickens list with
                                     nextChicken as the only element
82                              set overtakenSpace = (nextSpace + 1)
                                    % size of eggShapedTiles
83                              set nextAdjacentChicken =
                                    getChickenAtSpace(overtakenSpace)
84                              loop while nextAdjacentChicken is not
                                     null:
```

```
                                add nextAdjacentChicken to
                                    overtakenChickens list
                                set overtakenSpace = (
                                    overtakenSpace + 1) % size of
                                     eggShapedTiles
                                set nextAdjacentChicken =
                                    getChickenAtSpace(
                                    overtakenSpace)
                            print "Overtaking " + size of
                                overtakenChickens + " chicken(s)!
                                 Moving chicken to the next space
                                (s)."
                            loop for each overtakenChicken in
                                overtakenChickens:
                                remove tail feathers from
                                    overtakenChicken
                                add tailFeathers[overtakenChicken
                                    .getNumber() - 1] to
                                    currentChicken tailFeathers
                            set board[currentChicken] =
                                overtakenSpace
                        else:
                            print "CORRECT GUESS! Moving chicken
                                to the next space."
                            set board[currentChicken] = nextSpace
                            print "CURRENT CHICKEN: " +
                                currentChicken.getName()
                            if size of currentChicken
                                tailFeathers is 4:
                                call printBoard()
                                print "Congratulations! Player "
                                    + (currentPlayer + 1) + "
                                    wins!"
                                return
                    else:
                        print "WRONG GUESS! Next player's turn."
                        set correctTile = false
                else:
                    print "INVALID GUESS! Next player's turn."
                    set correctTile = false
            set currentPlayer = (currentPlayer + 1) % size of
                chickens

    method getChickenAtSpace(space):
        loop for each chicken in chickens:
            if board.get(chicken) is equal to space:
                return chicken
        return null

main():
        create a new Game object
        call setup()
        call play()
```

Chapter 7
Application Preview

In this preview, we will take you through the five captivating user interfaces (UIs) that make up our game: Menu, Gameplay, Instruction, Winning UI, and Choose How Many Player UI.
The Menu UI serves as the gateway to our game world. Once players dive into the Gameplay UI, they will be transported into a dynamic and thrilling game environment. For those seeking guidance, the Instruction UI provides a comprehensive overview of the game mechanics, controls, and objectives.
When players emerge victorious, they will be greeted by the Winning UI, celebrating their triumph in style. And let's not forget the Choose How Many Player UI! This UI allows players to customize their gaming experience by selecting the number of players in multiplayer or party modes.

Fig. 7.1 Application Preview

Chapter 8
Implemetation Detail

8.1 Game Analysis - UML Diagram

8.1.1 Class Diagram

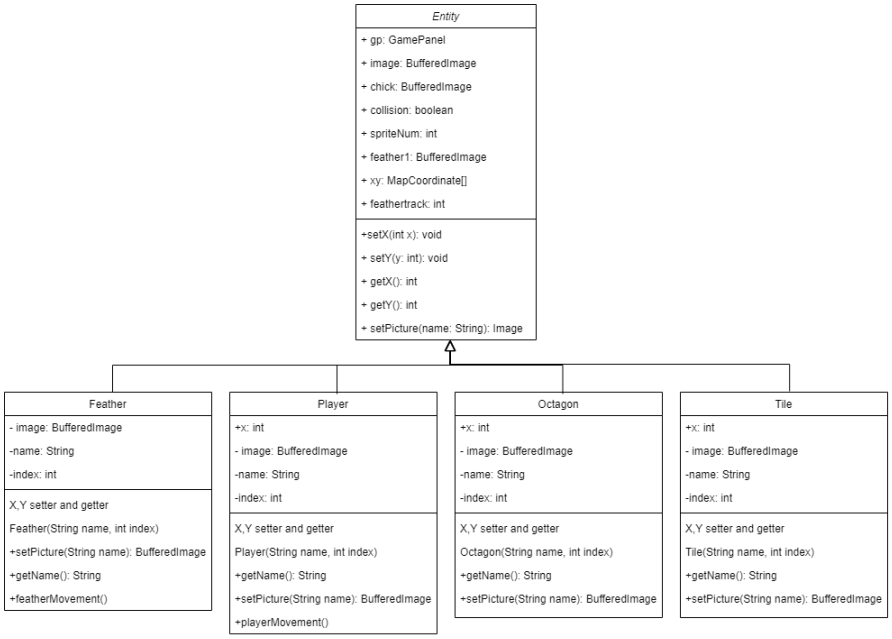

Fig. 8.1 Entity - Class Diagram

This diagram (Figure 8.1) shows the relationship between game entities. In the diagram we can see that Class Feather, Class Player, Class Octagon, Class Tile are

inherited from Class Entities. Each Class has appropriate methods in order to interact and do their own function.

8.1.2 Activity Diagram

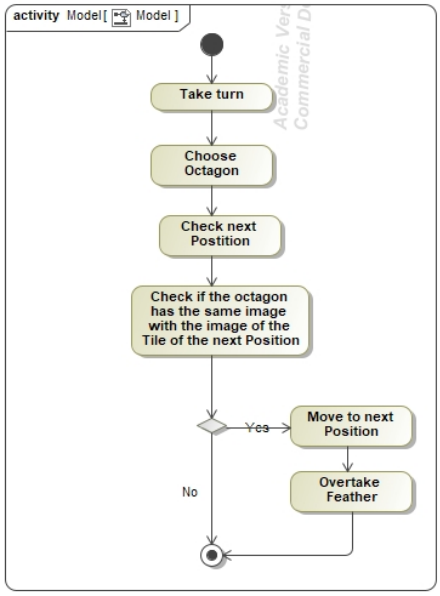

Fig. 8.2 Take turn - Activity Diagram

The figure shows the take turn function and how it work. Basically, the player will take their turn by choosing an octagon tile. Then, the system will check if the player can move to the next position or not.

8.2 Used Libraries

8.2.1 Java Swing

Java Swing is a powerful graphical user interface (GUI) toolkit that is part of the Java Foundation Classes (JFC). It provides developers with a comprehensive set of components and tools to create visually appealing and interactive desktop

applications. Swing is platform-independent, meaning that applications developed using Swing can run on any platform that supports Java.

Work Flow and Application of Java Swing:

1. Importing the Required Packages: To start using Java Swing, we need to import the necessary packages. We need to import javax.swing, which contains the core Swing components.

   ```
   import javax.swing;
   ```

2. Creating a JFrame: The JFrame class is the main window container in Swing applications. It provides the basic framework for building the application's user interface. You can create an instance of JFrame and set its properties such as size, title, and layout.
3. Adding Components: Swing provides a wide range of components, such as buttons, labels, text fields, checkboxes, and more. You can add these components to the JFrame using layout managers, which help in arranging the components within the window.
4. Handling Events: Swing components can generate events in response to user actions, such as button clicks or menu selections. You can register event listeners to handle these events and perform appropriate actions. Event listeners are interfaces that define methods to respond to specific events.
5. Customizing the Look and Feel: Swing allows you to customize the appearance of your application using different look and feel (L&F) options. You can choose from various predefined L&F themes or create your own. L&F themes control the visual style of components, including colors, fonts, and icons.
6. Building Menus and Toolbars: Swing provides classes for creating menus and toolbars to enhance the functionality of your application. You can add menu items, submenus, separators, and assign action listeners to perform operations when menu items are selected.
7. Managing Layouts: Swing offers different layout managers to define how components are positioned and sized within a container. Layout managers handle the automatic resizing and rearranging of components when the window size changes. Some commonly used layout managers include FlowLayout, BorderLayout, GridBagLayout, and BoxLayout.
8. Packaging and Deployment: Once you have developed your Swing application, you can package it into a distributable format, such as a JAR (Java Archive) file. This allows users to run your application on any system with Java support. You can also create installers for specific platforms using tools like Java Web Start or third-party packaging software

Java Swing is a versatile and feature-rich library for creating desktop applications with a graphical user interface. Its extensive set of components, event handling mechanisms, and layout managers make it a good choice for us to develop in this project.

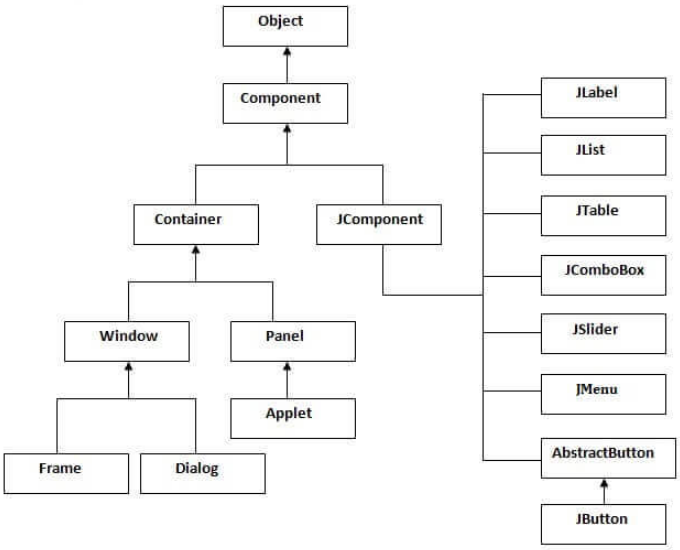

Fig. 8.3 Java Swing Hierachy

8.2.2 Java Abstract Window Toolkit

Java AWT (Abstract Window Toolkit) is a package in the Java programming language that provides a set of classes and methods for creating graphical user interfaces (GUIs). AWT was the original GUI toolkit for Java and is part of the core Java SE (Standard Edition) platform.

Key features of Java AWT include:

1. Components: AWT provides a set of pre-built components such as buttons, labels, text fields, checkboxes, radio buttons, menus, and more. These components are subclasses of the java.awt.Component class and can be customized and arranged to create GUI applications.
2. Containers: AWT includes container classes like java.awt.Frame, java.awt.Panel, and java.awt.Window that can hold and organize multiple components. Containers are used to create the structure and layout of the GUI.
3. Layout Managers: AWT provides layout managers that assist in arranging components within containers. Layout managers, such as FlowLayout, BorderLayout, GridLayout, and GridBagLayout, handle the positioning and resizing of components based on specified rules.
4. Event Handling: AWT supports event-driven programming, where components generate events (such as button clicks or mouse movements) and event listeners respond to these events. Event handling in AWT involves implementing listener interfaces like ActionListener, MouseListener, KeyListener, etc.

5. Graphics and Drawing: AWT provides classes like java.awt.Graphics and java.awt .Graphics2D that allow drawing and rendering on the screen. You can use these classes to create custom graphics, shapes, images, and perform operations like drawing lines, text, and filling shapes.
6. Fonts and Colors: AWT provides classes for working with fonts (java.awt.Font) and colors (java.awt.Color). You can specify font styles, sizes, and colors to customize the appearance of components and text within the GUI.
7. Event Dispatch Thread (EDT): AWT applications follow a single-threaded event model, where all GUI-related events are processed on the Event Dispatch Thread. This ensures thread safety and helps prevent GUI freezing or unresponsiveness.
8. While AWT is still available in Java, it has been largely superseded by Swing (Java's second GUI toolkit) and JavaFX, which provide more advanced features and improved performance. Nonetheless, AWT can still be used for basic GUI applications or when compatibility with older systems is required.
9. To use AWT in your Java application, you need to import the necessary AWT classes and create instances of the relevant components, containers, layout managers, and event listeners. By leveraging the AWT framework, you can build interactive and visually appealing GUI applications in Java.

8.3 Application Console Base

1. **Overview**
   The "Zicke Zacke Huhnerkacke" game is implemented in Java code which is outputted in the console of the compiler. This game is about moving chickens along a board with various tiles. Each tile has a corresponding octagonal tile associated with it. Players guess the octagonal tile corresponding to the next tile their chicken will land on. If guess correctly, the chicken moves forward. If guessed incorrectly, the turn passes to the next player.

2. **Classes**
   - Chicken:

```
class Chicken {
    private int number;
    private String name;
    private List<String> tailFeathers;

    public Chicken(int number, String name, String feather)
        {
        this.number = number;
        this.name = name;
        this.tailFeathers = new ArrayList<>();
        addTailFeather(feather);
    }

```

```
    public int getNumber() {
        return number;
    }

    public String getName() {
        return name;
    }

    public List<String> getTailFeathers() {
        return tailFeathers;
    }

    public void addTailFeather(String feather) {
        tailFeathers.add(feather);
    }

    public void removeTailFeathers() {
        tailFeathers.clear();
    }
}
```

  - Represents a chicken in the game.
  - Attributes:
    - 'number': The number assigned to the chicken.
    - 'name': The name of the chicken.
    - 'tailFeathers': List of tail feathers possessed by the chicken.
  - Methods:
    - 'Chicken(int number, String name, String feather)': Constructor to initialize a chicken with a number, name, and initial feather.
    - 'getNumber()': Getter method for the chicken's number.
    - 'getName()': Getter method for the chicken's name.
    - 'getTailFeathers()': Getter method for the chicken's tail feathers.
    - 'addTailFeather(String feather)': Method to add a tail feather to the chicken.
    - 'removeTailFeathers()': Method to remove all tail feathers from the chicken.

- Game:

```
class Game {
    private List<String> octagonalTiles;
    private List<String> eggShapedTiles;
    private List<Chicken> chickens;
    private List<String> tailFeathers;
    private int currentPlayer;
    private Map<Chicken, Integer> board;

public Game() {
    octagonalTiles = new ArrayList<>();
    octagonalTiles.add("flower");       // 1
    octagonalTiles.add("balut");        // 2
    octagonalTiles.add("hedgehog");     // 3
```

```
    octagonalTiles.add("feather");      // 4
    octagonalTiles.add("egg");          // 5
    octagonalTiles.add("rabbit");       // 6
    octagonalTiles.add("chicken");      // 7
    octagonalTiles.add("snail");        // 8
    octagonalTiles.add("caterpillar");  // 9
    octagonalTiles.add("worm");         // 10
    octagonalTiles.add("omelette");     // 11
    octagonalTiles.add("chick");        // 12

    eggShapedTiles = new ArrayList<>();
    for (String tile : octagonalTiles) {
        eggShapedTiles.add(tile);
        eggShapedTiles.add(tile);
    }

    chickens = new ArrayList<>();
    tailFeathers = new ArrayList<>();
    tailFeathers.add("Red");
    tailFeathers.add("Green");
    tailFeathers.add("Blue");
    tailFeathers.add("Yellow");

    currentPlayer = 0;
    board = new HashMap<>();
}
```

- Manages the gameplay and state of the game.
- Attributes:
  - 'octagonalTiles': List of octagonal tile names.
  - 'eggShapedTiles': List of egg-shaped tiles derived from octagonal tiles.
  - 'chickens': List of chickens participating in the game.
  - 'tailFeathers': List of tail feather colors.
  - 'currentPlayer': Index of the current player.
  - 'board': Map representing the position of each chicken on the board.
- Methods:
  - 'Game()': Constructor to initialize game attributes.
  - 'setup()': Method to set up the game by creating chickens and placing them on the board.

```
public void setup() {
    Collections.shuffle(octagonalTiles);
    Collections.shuffle(eggShapedTiles);

    for (int i = 0; i < 4; i++) {
        Chicken chicken = new Chicken(i + 1, "Chicken
            " + tailFeathers.get(i), tailFeathers.get(
            i));
        chickens.add(chicken);
        board.put(chicken, i * 6); // Place each
            chicken on an egg-shaped tile with 5 free
            tiles between them
```

```
    }
}
```

- 'printBoard()': Method to print the current state of the board.

```
public void printBoard() {
    System.out.println();
    System.out.println("Board:");
    for (Chicken chicken : chickens) {
        System.out.println(chicken.getName() + " - Space " + (board.get(chicken) + 1)
        + " - Feather: " + chicken.getTailFeathers());
    }
    System.out.println();
}
```

- 'play()': Method to execute the gameplay logic.

```
public void play() {
    Scanner scanner = new Scanner(System.in);
    while (true) {
        Chicken currentChicken = chickens.get(currentPlayer);

        System.out.println("PLAYER " + (currentPlayer + 1) + ", it's your turn.");
        System.out.println();
        System.out.println("CURRENT CHICKEN: " + currentChicken.getName());

        boolean correctTile = true;

        while (correctTile) {
            printBoard();

            int currentSpace = board.get(currentChicken);
            String nextTile = eggShapedTiles.get(currentSpace);
            System.out.println("Next tile: " + nextTile);

            System.out.print("Guess the number of the octagonal tile (1 to 12): ");
            int guessedNumber = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character

            if (guessedNumber >= 1 && guessedNumber <= octagonalTiles.size()) {
                String chosenTile = octagonalTiles.get(guessedNumber - 1);

                if (chosenTile.equals(nextTile)) {
```

```
117                     int nextSpace = (currentSpace + 1)
                            % eggShapedTiles.size();
118                     Chicken nextChicken =
                            getChickenAtSpace(nextSpace);
119
120                     if (nextChicken != null) {
121                         List<Chicken>
                                overtakenChickens = new
                                ArrayList<>();
122                         overtakenChickens.add(
                                nextChicken);
123
124                         int overtakenSpace = (
                                nextSpace + 1) %
                                eggShapedTiles.size();
125                         Chicken nextAdjacentChicken =
                                getChickenAtSpace(
                                overtakenSpace);
126
127                         while (nextAdjacentChicken !=
                                null) {
128                             overtakenChickens.add(
                                    nextAdjacentChicken);
129                             overtakenSpace = (
                                    overtakenSpace + 1) %
                                    eggShapedTiles.size();
130                             nextAdjacentChicken =
                                    getChickenAtSpace(
                                    overtakenSpace);
131                         }
132                         System.out.println();
133                         System.out.println("Overtaking
                                " + overtakenChickens.
                                size() + " chicken(s)!
                                Moving chicken to the next
                                space(s).");
134
135                         for (Chicken overtakenChicken
                                : overtakenChickens) {
136                             overtakenChicken.
                                    removeTailFeathers();
137                             currentChicken.
                                    addTailFeather(
                                    tailFeathers.get(
                                    overtakenChicken.
                                    getNumber() - 1));
138                         }
139
140                         board.put(currentChicken,
                                overtakenSpace);
141
142                     } else {
```

```
                        System.out.println("CORRECT
                            GUESS! Moving chicken to
                            the next space.");
                        board.put(currentChicken,
                            nextSpace);

                        System.out.println("CURRENT
                            CHICKEN: " +
                            currentChicken.getName());
                    }
                    if (currentChicken.getTailFeathers
                        ().size() == 4) {
                        printBoard();
                        System.out.println();
                        System.out.println("
                            Congratulations! Player "
                            + (currentPlayer + 1) + "
                            wins!");
                        return;
                    }
                } else {
                    System.out.println("WRONG GUESS!
                        Next player's turn.");
                    correctTile = false;
                }
            } else {
                System.out.println("INVALID GUESS!
                    Next player's turn.");
                correctTile = false;
            }
        }
        currentPlayer = (currentPlayer + 1) % chickens
            .size();
        System.out.println();
    }
}
```

- 'getChickenAtSpace(int space)': Helper method to retrieve the chicken at a specified space on the board.

```
private Chicken getChickenAtSpace(int space) {
    for (Chicken chicken : chickens) {
        if (board.get(chicken) == space) {
            return chicken;
        }
    }
    return null;
}
```

- ZickeZackeHuhnerkacke (Main):
  - Contains the main method to run the game.

```
public class ZickeZackeHuhnerkacke {
    public static void main(String[]args) {
```

```
177             Game game = new Game();
178             game.setup();
179             game.play();
180         }
181     }
```

3. **Gameplay**
   - Players take turns guessing the octagonal tile corresponding to the next tile their chicken will land on.
   - If guessed correctly, the chicken moves forward, possibly overtaking other chickens and collecting their tail feathers.
   - If guessed incorrectly, the turn passes to the next player.
   - The game continues until a player collects all four feathers, winning the game.
4. **Conclusion:**
   The provided code implements the ”ZickeZackeHuhnerkacke” game with basic gameplay mechanics. It effectively models the movement of chickens on the board and the interaction between players. Further enhancements could include adding more diverse tiles, implementing advanced gameplay mechanics, and enhancing the user interface. Additionally, thorough testing and error handling should be incorporated to ensure the robustness of the game.

8.4 Application Details and Code Snippets

8.4.1 Main Program

- Firstly, the code includes necessary import statements to access classes from the javax.swing and java.awt packages, which are required for creating the window and handling user events.
- The Main class contains the main method, serving as the entry point of the program. It creates an instance of the JFrame class and assigns it to the window variable. Several methods are called on the window object to configure its behavior, such as setDefaultCloseOperation() to specify the action when the user closes the window (in this case, EXIT_ON_CLOSE), setResizable() to determine whether the window can be resized (set to false in this code), and setTitle() to set the title of the window as ”Zicke Zacke Hühnerkacke”.
- Next, an instance of the GamePanel class is created and assigned to the gamePanel variable. The gamePanel object is then added to the window using the add() method, and the pack() method is called to resize the window to fit the preferred size of its components.
- The code also sets additional properties for the window. The setLocationRelativeTo() method is called to center the window on the screen, and setVisible() is called to make the window visible to the user.

- Finally, the game is started by calling two methods on the gamePanel object. The setupGame() method is invoked to initialize the game, and the startGameThread() method is called to start the game loop in a separate thread.

```
package Main;
import javax.swing.JFrame;
import java.awt.BorderLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;


public class Main {
    public static void main(String[] args){
    JFrame window = new JFrame();
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    window.setResizable(false);
    window.setTitle("Zicke Zacke Huehnerkacke");

    GamePanel gamePanel = new GamePanel();
    window.add(gamePanel);
    window.pack();

    window.setLocationRelativeTo(null);
    window.setVisible(true);
    gamePanel.setupGame();
    gamePanel.startGameThread();
    }
}
```

8.4.2 Class Octagon and Octagon Manager

Octagon class: Class Octagon is responsible about creating an Octagon object. The Octagon class has several attributes

1. The class imports necessary packages and classes, including ”Graphics2D”, ”BufferedReader”, ”InputStream”, ”InputStreamReader”, ”Random”, and ”GamePanel”.

```
package Octagon;

import java.awt.Image;
import java.awt.Rectangle;
import javax.imageio.ImageIO;
```

2. **Attributes:**
   - x, y, width, height: These attributes represent the position and dimensions of the Octagon on the screen. The x and y variables store the coordinates of the top-left corner of the Octagon, while width and height determine its size.
   - name: This attribute stores the name of the Octagon as a string.

- index: This attribute is an integer value that represents the index of the Octagon. (e.g. The game has 12 octagonal tiles; therefore, the index of Octagon class would be 12.)
- image: This attribute is of type Image and represents the image associated with the Octagon.
- collision: This boolean attribute indicates whether a mouse clicked has collided with the Octagon. By setting each Octagon with its own position. When a mouse is clicked in a specific position, the function will detect which Octagon is corresponded to that position and will return a true or false value.
- isBackSide: This boolean attribute determines whether the Octagon is currently showing its backside or frontside.

```
10 public class Octagon {
11     private int x;
12     private int y;
13     private int width;
14     private int height;
15     private String name;
16     private int index;
17
18     public Image image;
19     public boolean collision = false;
20     public boolean isBackSide = true;
```

3. **Constructor:**

- The constructor takes in a name and an index as parameters and initializes the width and height of the Octagon to 70. It also sets the name and index attributes. The constructor then calls the setPicture() method to set the image of the Octagon to the ”backside” image.

```
21 public Octagon(String name, int index)
22 {
23     width =70;
24     height = 70;
25     this.name=name;
26     this.index=index;
27     setPicture("backside");
28 }
29 public String getName()
30 {
31     return name;
32 }
33 public int getIndex()
34 {
35     return this.index;
36 }
```

4. **Getter and Setter Methods:**

- The class provides getter and setter methods for various attributes such as getX(), getY(), getIndex(), getName(), getIsBackSide(), setX(), setY(), setWidth(), setHeight(), and setIsBackSide(). These methods allow access to and modification of the Octagon's attributes.

5. **setPicture() method:**

- The setPicture() method is responsible for loading the image of the Octagon. It reads the image file from the resources directory using the ImageIO.read() method and sets the image attribute to the loaded image. If there is an error while loading the image, it prints the stack trace.

```
37 public void setPicture(String name)
38 {
39     try {
40        image = ImageIO.read(getClass().getResourceAsStream("/
              res/octagonal_shape/"+name+".png"));
41     } catch (Exception e) {
42         e.printStackTrace();
43     }
44 }
```

6. **collision() method:**

- The collision() method takes in the coordinates of a cursor and checks whether the cursor is within the bounds of the Octagon. It creates a new Rectangle object representing the bounds of the Octagon and uses the contains() method to check if the cursor coordinates are within the bounds. If they are, the method returns true, indicating a collision.

7. **flip() method:**

- The flip() method is used to animate the flipping of the Octagon. It modifies the width and x-coordinate of the Octagon to create a flipping effect. The direction variable controls the direction of the flip. When the width reaches 0, the direction is changed to -1, and when the width reaches 70, the direction is changed back to 1. The method returns true until the flipping animation is complete, and then returns false to indicate that the flipping is finished.

```
45 public boolean collision(int x_cursor, int y_cursor){
46     Rectangle octagonRect= new Rectangle(x, y, width, height);
47     return octagonRect.contains(x_cursor, y_cursor);
48 }
49 int direction =1;
50 public boolean flip()
51 {
52     width -=  30*direction;
53     x += 15*direction;
54     if(width <=0)
55     {
56         direction*=-1;
```

```
    }
    if(width==70 && direction == -1)
    {
        direction=1;
        isBackSide= !(isBackSide);
        return false;
    }
    return true;
}
```

OctagonManager class: The OctagonManager class is responsible for managing the Octagons in the game.

1. The class imports necessary packages and classes, including ”Graphics2D” ”Random”, ”ImageIO”, and ”GamePanel”.

```
package Octagon;

import java.awt.Graphics2D;
import java.util.Random;
import javax.imageio.ImageIO;
import Main.GamePanel;
```

2. **Attributes:**
   - gp: This attribute is of type GamePanel and represents the game panel where the Octagons will be rendered.
   - octagons: This is an array of Octagon objects and stores all the Octagons in the game.
   - mapOctagonNum: This is a 2D integer array used to store the number of Octagons in each map.
   - backside and backside2: These are Octagon objects representing the backside images of the Octagons.
   - It also calls the getOctagonImage() method to get Octagon image each time the OctagonManager is called.
3. **getArray() method:**
   - This method returns the octagons attribute, allowing other classes to access the array of Octagons.

```
public class OctagonManager {
    GamePanel gp;
    public static Octagon[] octagons;
    int mapOctagonNum[][];
    Octagon backside;
    Octagon backside2;
    public static Octagon[] getArray()
    {
        return octagons;
    }
```

4. **shuffleArray() method:**

   - This method takes in an array of strings and shuffles its elemtns randomly. It uses the Fisher-Yates algorithm to perform the shuffling.

```
17 public static void shuffleArray(String[] array) {
18     Random random = new Random();
19     int n = array.length;
20 
21     for (int i = 0; i < n; i++) {
22         int randomIndex = i + random.nextInt(n - i);
23         String temp = array[i];
24         array[i] = array[randomIndex];
25         array[randomIndex] = temp;
26     }
27 }
```

5. **Constructor:**

   - The constructor takes in a GamePanel object as a parameter and initializes the gp attribute with it. It also initializes the octagons array with a size of 13 and calls the getOctagonImage() method to load the Octagon images.
   - When the game started, the default index is set to 12, and the index is updated after the first player chooses the first octagon tile. That is why the octagons array has a size of 13.

```
28 public OctagonManager(GamePanel gp)
29 {
30     this.gp = gp;
31     octagons = new Octagon[13];
32     getOctagonImage();
33 }
```

6. **getOctagonImage() method:**

   - This method loads the images for the Octagons from the resources directory. It uses the ImageIO.read() method to read the image files and assigns them to the corresponding Octagon objects in the octagons array. The shuffleArray() method is called to shuffle the names of the Octagons before assigning them to the Octagon objects.

```
34 public void getOctagonImage(){
35     String[] name={"chick", "connhim", "eggs", "feather", "
           flower", "hotvitlon",  "omelette", "rabbit", "
           sadchick", "snail", "worm", "worm7mau"};
36     try {
37         backside = new Octagon("backside",12);
38         backside.image = ImageIO.read(getClass().
               getResourceAsStream("/res/octagonal_shape/
               backside.png"));
39         shuffleArray(name);
```

- Next, a loop is used to iterate over the first 12 elements of the shuffled array, which correspond to the 12 Octagons. Inside the loop, each Octagon object in the octagons array is initialized with its corresponding name and index.
- The X and Y positions of each Octagon are set based on their index using the following logic:
  - The X position is calculated by adding the horizontal distance between the Octagons (48 pixels) to a base X position (300 pixels for the first column, 396 pixels for the second column, and so on).
  - The Y position is calculated by adding the vertical distance between the Octagons (75 pixels) to a base Y position (140 pixels for the first row, 215 pixels for the second row, and so on).

```
40      for(int i=0; i<12; i++)
41      {
42          octagons[i] = new Octagon(name[i],i);
43      }
44      octagons[12] = new Octagon("worm", 12);
45
46      octagons[0].setX(300 + 48/2); octagons[0].setY(140 +
            48/2);
47
48      octagons[1].setX(396 + 48/2); octagons[1].setY(140 +
            48/2);
49
50      octagons[2].setX(300 + 48/2); octagons[2].setY(215 +
            48/2);
51
52      octagons[3].setX(396 + 48/2); octagons[3].setY(215 +
            48/2);
53
54      octagons[4].setX(300 + 48/2); octagons[4].setY(290 +
            48/2);
55
56      octagons[5].setX(396 + 48/2); octagons[5].setY(290 +
            48/2);
57
58      octagons[6].setX(300 + 48/2); octagons[6].setY(365 +
            48/2);
59
60      octagons[7].setX(396 + 48/2); octagons[7].setY(365 +
            48/2);
61
62      octagons[8].setX(204 + 48/2); octagons[8].setY(215 +
            48/2);
63
64      octagons[9].setX(492 + 48/2); octagons[9].setY(215 +
            48/2);
65
66      octagons[10].setX(204 + 48/2); octagons[10].setY(290 +
            48/2);
67
```

```
68      octagons[11].setX(492 + 48/2); octagons[11].setY(290 +
            48/2);
69      } catch (Exception e) {
70          e.printStackTrace();
71      }
72  }
```

7. **draw() method:**

   - This method is responsible for drawing the Octagons on the game panel. It uses a loop to iterate over the octagons array and calls the getPicture() method of each Octagon object to get its image. The image is then drawn on the game panel using the drawImage() method of the Graphics2D object.

```
73  public void draw(Graphics2D g2){
74      for(int i=0; i<12; i++)
75      {
76          g2.drawImage(octagons[i].getPicture(), octagons[i].
                getX(), octagons[i].getY(),
77                      octagons[i].getWidth(), octagons[i].
                        getHeight(), null);
78      }
79  }
```

Overall, the Octagon class allows for the creation, manipulation, and animation of Octagons on the screen. And the OctagonManager class handles the loading and rendering of Octagons in the game. It provides methods to access the Octagon array and to shuffle the positions and names of the Octagons.

8.4.3 Class Tile and Tile Manager

Tile class: The Tile class represents the egg shaped tile in a game. Its image is based on the octagon's image, but doubled.

1. The class imports necessary packages and classes, including "Image" and "ImageIO".

```
1  package Tile;
2
3  import java.awt.Image;
4  import javax.imageio.ImageIO;
```

2. **Attributes:**

   - x, y: Integers representing the x and y coordinates of the tile.
   - width, height: Integers representing the width and height of the tile.
   - name: A string representing the name of the tile.

- index: An integer representing the index of the tile.
- image: An Image object representing the image of the tile.
- collision: A boolean indicating whether there is a collision with the tile.

```
public class Tile {
    private int x;
    private int y;
    private int width;
    private int height;
    private String name;
    private int index;
    public Image image;
```

3. **Constructor:**

- The constructor takes in the name and index of the tile and initializes the corresponding attributes.
- It also calls the setPicture() method to load the image of the tile.

```
public Tile(String name,int index)
{
    this.index=index;
    this.name=name;
    setPicture(name);
}
public void setName(String name) {
    this.name = name;
}
public String getName()
{
    return name;
}
public int getIndex()
{
    return this.index;
}
```

4. **Getter and Setter methods:**

- There are getter and setter methods for name, x, y, width, and height such as getX(), getY(), getWidth(), and getHeight().
- These methods allow other classes to access and modify the attributes of a tile.

```
public void setX(int x)             {this.x=x;}

public int getX()                   {return this.x;}

public void setY(int y)             {this.y=y;}

public int getY()                   {return this.y;}

public void setWidth(int width)     {this.width=width;}
```

```
39
40 public void setHeight(int height)  {this.height=height;}
41
42 public int getWidth()               {return this.width;}
43
44 public int getHeight()              {return this.height;}
```

5. **setPicture() and getPicture() methods:**

- The setPicture() method loads the image of the tile using the ImageIO.read() method from the resources directory.
- It reads the image file from the ”/res/Tile/” directory and appends the name and ”.png” extension to locate the correct file.
- The getPicture() method returns the image of a tile.

```
45 public void setPicture(String name)
46 {
47     try {
48        image = ImageIO.read(getClass().getResourceAsStream("
              /res/Tile/"+name+".png"));
49     } catch (Exception e) {
50         e.printStackTrace();
51     }
52 }
53 public Image getPicture()
54 {
55     return image;
56 }
```

TileManager class: The TileManager class is responsible for managing the tiles in the game. Let’s continue the report by analyzing the code:

1. The class imports necessary packages and classes, including ”Graphics2D”, ”BufferedReader”, ”InputStream”, ”InputStreamReader”, ”Random”, and ”GamePanel”.

```
1 package Tile;
2
3 import java.awt.Graphics2D;
4 import java.io.BufferedReader;
5 import java.io.InputStream;
6 import java.io.InputStreamReader;
7 import java.util.Random;
8 import Main.GamePanel;
```

2. **Attributes:**

- gp: A reference to the GamePanel object.
- tile: An array of Tile objects representing the tiles in the game.
- mapTileNum: A 2D array of integers representing the map layout with tile numbers.
- egg: A static array of Tile objects representing the eggs in the game.

3. **getArray() Method:**
   - This method returns the egg array.

```
public class TileManager {
    GamePanel gp;
    Tile[] tile;
    int mapTileNum [] [];
    public static Tile[] getArray(){
        return egg;
    }
```

4. **shuffleArray() method:**
   - This method shuffles the elements of the provided array using the Fisher-Yates algorithm.
   - It takes in an array of Tile objects and randomly swaps elements within the array.

```
public static Tile[] egg;
public static void shuffleArray(Tile[] array) {
    Random random = new Random();
    int n = array.length;

    for (int i = 0; i < n; i++) {
        int randomIndex = i + random.nextInt(n - i);
        Tile temp = array[i];
        array[i] = array[randomIndex];
        array[randomIndex] = temp;
    }
}
```

5. **Constructor:**
   - The constructor takes in a GamePanel object and initializes the gp, tile, egg, and mapTileNum attributes.
   - It calls the getTileImage() method to load the tile images and shuffles the egg array.

```
public TileManager(GamePanel gp){
    this.gp = gp;
    tile = new Tile[10];
    egg = new Tile[24];
    mapTileNum = new int[gp.maxScreenCol][gp.maxScreenRow];
    getTileImage();
    loadMap("/res/Map/map1.txt");
}
```

6. **getTileImage() Method:**
   - This method is responsible for loading the tile images.
   - It creates a Tile object for the floor tile and initializes the tile array.

- It also initializes the egg array with Tile objects representing different types of eggs.
- The shuffleArray() method is called to shuffle the egg array randomly.

```
public void getTileImage(){
    String[] name={"chick", "connhim", "eggs", "feather", "
        flower", "hotvitlon",  "omelette", "rabbit", "sadchick
        ", "snail", "worm", "worm7mau","chick", "connhim", "
        eggs", "feather", "flower", "hotvitlon", "omelette", "
        rabbit", "sadchick", "snail", "worm", "worm7mau"};
    try {
        tile[0] = new Tile("floor01", 0);
        for(int i=0; i<24; i++)
        {
            egg[i]= new Tile(name[i],i);
        }
        shuffleArray(egg);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

7. **loadMap() Method:**

- This method loads the map layout from a text file.
- It takes in a file path and reads the file line by line.
- Each line represents a row in the map, and the tile numbers are split by spaces.
- The numbers are converted to integers and stored in the mapTileNum array.

```
public void loadMap(String filePath){
    try {
        InputStream is = getClass().getResourceAsStream(
            filePath);
        BufferedReader br = new BufferedReader(new
            InputStreamReader(is));
        int col = 0;
        int row = 0;
        while(col < gp.maxScreenCol && row < gp.maxScreenRow){
            String line = br.readLine();

            while(col < gp.maxScreenCol){
                String numbers[] = line.split(" ");

                int num = Integer.parseInt(numbers[col]);
                mapTileNum[col][row] = num;
                col++;
            }
            if(col == gp.maxScreenCol){
                col = 0;
                row++;
            }
```

```
70
71          }
72          br.close();
73      } catch (Exception e) {
74
75      }
76  }
```

8. **draw() Method:**

   - This method is responsible for drawing the tiles and eggs on the game panel.
   - It uses nested loops to iterate through the mapTileNum array and draw the corresponding tiles.
   - It also iterates through the egg array and draws the eggs on the game panel according to their positions.

```
77  public void draw(Graphics2D g2){
78      int col = 0;
79      int row = 0;
80      int x = 0;
81      int y = 0;
82      int mapX = (gp.screenWidth - gp.maprow*(gp.eggwidth+5)) /
            2;
83      int mapY = (gp.screenHeight - gp.mapcol*(gp.eggheight+5))
            / 2;
84      int maprow = 0;
85      int mapcol = 0;
86      int track = 0;
87      while(col < gp.maxScreenCol && row < gp.maxScreenRow){
88          int tileNum = mapTileNum[col][row];
89          g2.drawImage(tile[tileNum].getPicture(), x, y, gp.
                tileSize, gp.tileSize, null);
90          col++;
91          x += gp.tileSize;
92          if(col == gp.maxScreenCol){
93              col = 0;
94              x = 0;
95              row++;
96              y += gp.tileSize;
97          }
98      }
99      while(maprow < gp.maprow && mapcol < gp.mapcol ){
100         g2.drawImage(egg[track].getPicture(), mapX, mapY, gp.
                eggwidth, gp.eggheight, null);
101         track++;
102         maprow++;
103         mapX += gp.eggwidth + 5;
104         if(maprow == gp.maprow){
105             while(mapcol < gp.mapcol-1){
106                 mapY += gp.eggheight + 5;
107                 egg[track].setX(mapX);
108                 egg[track].setY(mapY);
```

```
                g2.drawImage(egg[track].getPicture(), mapX -
                    gp.eggwidth - 5 , mapY, gp.eggwidth, gp.
                    eggheight, null);
                mapcol++;
                track++;
            }
        }
    }
    if(mapcol == gp.mapcol-1){
        mapX = mapX - gp.eggwidth - 5;
        maprow --;
        while(maprow > 0){
            maprow--;
            mapX = mapX - gp.eggwidth - 5;
            egg[track].setX(mapX);
            egg[track].setY(mapY);
            g2.drawImage(egg[track].getPicture(), mapX, mapY,
                gp.eggwidth, gp.eggheight, null);
            track++;
        }
    }
    if(maprow == 0){
        mapcol --;
        while(mapcol > 0){
            mapcol--;
            mapY = mapY - gp.eggheight - 5;
            egg[track].setX(mapX);
            egg[track].setY(mapY);
            g2.drawImage(egg[track].getPicture(), mapX, mapY,
                gp.eggwidth, gp.eggheight, null);
            track++;
        }
    }
}
```

Overall, the Tile class provides functionality to set the tile's image, retrieve its properties, and handle collisions. And the TileManager class is for loading tile images, shuffling the egg shaped tile, loading the map layout, and handling the drawing of egg shaped tiles on the game panel.

8.4.4 Class Player and Player Manager

1. Class Player

   The provided code represents a class called "Player" in the "Entity" package. This class serves as a blueprint for creating player objects in a game. Each player object has properties such as name, position, and index, which determine their identity and current state within the game.

The ”Player” class extends the ”Entity” class, indicating that it inherits properties and methods from the ”Entity” class. By extending the ”Entity” class, the ”Player” class can incorporate additional functionalities specific to players while leveraging the common functionalities defined in the ”Entity” class.

The class includes a constructor that takes three parameters: ”name,” ”position,” and ”index.” When a new player object is created using this constructor, the provided values are used to initialize the corresponding instance variables.

To access and modify the values of the player's name, position, and index, the class provides getter and setter methods. These methods allow other classes to retrieve and update these properties as needed.

Overall, the ”Player” class forms an essential component of a game's entity system, providing a representation for players and their attributes. By creating instances of the ”Player” class, the game can keep track of individual players, their positions, and other relevant information throughout the gameplay.

```
1  package Entity;
2
3  public class Player extends Entity {
4      String name;
5      int position;
6      int index;
7      public Player(String name, int position, int index){
8          this.name = name;
9          this.position = position;
10         this.index = index;
11     }
12     public int getIndex() {
13         return index;
14     }
15
16     public void setIndex(int index) {
17         this.index = index;
18     }
19     public String getName() {
20         return name;
21     }
22
23     public void setName(String name) {
24         this.name = name;
25     }
26
27     public int getPosition() {
28         return position;
29     }
30
31     public void setPosition(int position) {
32         this.position = position;
```

```
    }
}
```

2. Class Player Manager

- The PlayerManager class extends the Entity class.

```
package Entity;
import Main.GamePanel;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.util.Arrays;
import javax.imageio.ImageIO;
public class PLayerManager extends Entity {
    GamePanel gp;
    Graphics2D g2;
    public MapCoordinate[] xy;
    Feather[] feathers = FeatherManager.getArray();

    public PLayerManager(GamePanel gp){
        this.gp = gp;
        xy = new MapCoordinate[24];
    }
```

- The class has a reference to a GamePanel object (gp) and a Graphics2D object (g2).
- It has an array of MapCoordinate objects called xy.
- The class has a Feather array called feathers which is initialized by calling the getArray() method of FeatherManager.
- The constructor of the class takes a GamePanel object as a parameter and assigns it to the gp field.
- The setUpPlayerManager method is called to set up the player manager. It calls the following methods: setUpPlayer(), setDefaultValue(), and getPlayerImage().

```
public void setUpPlayerManager()
    {
        setUpPlayer();
        setDefaultValue();
        getPlayerImage();
    }
    public static Player[] getArray()
    {
        return players;
    }
    public void setDefaultValue(){
        x = 120;
        y = 37;
        direction = "normal";
        for(int i=0; i<24; i++){
            xy[i] = new MapCoordinate(i,i);
```

```
69
70          }
71          for(int i=0; i<7; i++){
72              xy[i].setX(x);
73              xy[i].setY(y);
74              x += 5 + gp.eggwidth;
75          }
76          for(int i=7; i>=7 &&i<12;i++){
77              xy[i].setX(x);
78              xy[i].setY(y);
79              y += 5 + gp.eggheight;
80          }
81          for(int i=12; i>=12 && i<19; i++){
82              xy[i].setX(x);
83              xy[i].setY(y);
84              x -= 5 + gp.eggwidth;
85          }
86          for(int i=19; i>=19 &&i<=23;i++){
87              xy[i].setX(x);
88              xy[i].setY(y);
89              y -= 5 + gp.eggheight;
90          }
91
92      }
93      public void setUpPlayer()
94      {
95          if(gp.howManyPlayer==2)
96          {
97              players = new Player[2];
98              players[0]= new Player("Blue", playerposition1,
                    1);
99              players[1]= new Player("Yellow", playerposition3
                    , 1);
100         }
101         if(gp.howManyPlayer==3)
102         {
103             players = new Player[3];
104             players[0]= new Player("Blue", 0, 1);
105             players[1]= new Player("Yellow", 8, 1);
106             players[2]= new Player("White", 16 , 1);
107         }
108         if(gp.howManyPlayer==4)
109         {
110             players = new Player[4];
111             players[0]= new Player("Blue", playerposition1,
                    1);
112             players[1]= new Player("Yellow", playerposition2
                    , 1);
113             players[2]= new Player("White", playerposition3,
                     1);
114             players[3]= new Player("Red", playerposition4,
                    1);
115         }
116         //System.out.println(players[0].getName());
```

```
117     }
```

- The getArray() method returns an array of Player objects.
- The setDefaultValue method initializes the xy array with MapCoordinate objects and sets default values for the player's position (x and y), and direction.
- The method then sets the x and y coordinates for each MapCoordinate object in the xy array in a specific pattern.
- The setUpPlayer method creates Player objects based on the value of gp.howManyPlayer (which determines the number of players). The players are assigned names and positions based on the number of players.
- The PlayerManager class has several methods for managing player movement and drawing player sprites in a game, which will be disscussed in the GUI section.

8.4.5 Class Feather and Feather Manager

1. Class Feather

- The Feather class represents a feather entity in a game.
- The class has several instance variables, including a reference to the GamePanel class (gp), x and y coordinates, an image variable to store the feather image, width and height of the feather, name of the feather, position indicating the feather's position, and index indicating the feather's index.
- The class has a constructor that takes the name, position, and index of the feather as parameters. It initializes the width, height, name, position, and index variables.
- The class has getter and setter methods for the index and position variables.
- The class has getter and setter methods for the x and y coordinates.
- The class has a setPicture method that takes a name parameter and sets the image variable by reading an image file from the specified resource path. The image file is loaded using the ImageIO.read method.

```
1  package Entity;
2  import java.awt.Image;
3  import java.awt.image.BufferedImage;
4  import javax.imageio.ImageIO;
5  import Main.GamePanel;
6  public class Feather {
7      GamePanel gp;
8      int x,y;
9      BufferedImage image;
10     private int width;
11     private int height;
12     private String name;
13     public int position;
14     public int index;
```

```
15
16      public Feather(String name, int position, int index)
17      {
18          width = 90;
19          height = 90;
20          this.name=name;
21          this.position=position;
22          this.index=index;
23      }
24      public int getIndex() {
25          return index;
26      }
27      public void setIndex(int index) {
28          this.index = index;
29      }
30      public int getPosition() {
31          return position;
32      }
33      public void setPosition(int position) {
34          this.position = position;
35      }
36      public void setX(int x)
37      {
38          this.x=x;
39      }
40      public void setY(int y)
41      {
42          this.y=y;
43      }
44      public int getX() {
45          return x;
46      }
47
48      public int getY() {
49          return y;
50      }
51      public BufferedImage setPicture(String name)
52      {
53          try {
54             image = ImageIO.read(getClass().getResourceAsStream
                   ("/res/chicken animation deluxe/Feather/"+name+
                   ".png"));
55          } catch (Exception e) {
56              e.printStackTrace();
57          }
58          return image;
59      }
60  }
```

2. Class Feather Manager

   - The class imports necessary packages and classes, including ”Graphics2D”, ”BufferedImage”, ”ImageIO”, and ”GamePanel”.

```
package Entity;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import Main.GamePanel;
```

- The class declares several instance variables, including ”image”, ”bluefeather”, ”yellowfeather”, ”whitefeather”, and ”redfeather”, which are BufferedImage objects representing different feather images. The ”direction” variable stores the direction of the feathers. The ”spriteCounter” and ”spriteNum” variables are used for animating the feathers. The ”gp” variable is a reference to the GamePanel object. The ”p1” variable is a reference to the PlayerManager object. The ”xy” variable is an array of MapCoordinate objects.
- The class declares several static arrays of Feather objects, including ”featheranimation1”, ”featheranimation2”, ”featheranimation3”, ”featheranimation4”, and ”featheranimation5”. These arrays store different animations of feathers.

```
public BufferedImage image;
    public BufferedImage bluefeather, yellowfeather,
        whitefeather, redfeather;
    public String direction;
    public int spriteCounter=0;
    public int spriteNum=1;
    GamePanel gp;
    PLayerManager p1;
    public MapCoordinate[] xy;

    public static Feather[] featheranimation1,
        featheranimation2, featheranimation3,
        featheranimation4, featheranimation5;
    String[] name = {"bluefeather1","yellowfeather1","
        whitefeather1","redfeather1",
                    "bluefeather2","yellowfeather2","
                        whitefeather2","redfeather2",
                    "bluefeather3","yellowfeather3","
                        whitefeather3","redfeather3",
                    "bluefeather4","yellowfeather4","
                        whitefeather4","redfeather4"};
```

- The class defines a constructor that takes a GamePanel object as a parameter. It initializes the instance variables, sets default values, and loads the feather images.
- The class defines a method called ”setUpFeatherManager()” which sets up the feathers and their drawing.

```
 public void setUpFeather()
    {
        if(gp.howManyPlayer==2)
        {
            feathers = new Feather[2];
            feathers[0]= new Feather("xdd", 0, 0);
            feathers[1]= new Feather("xdd", 12, 12);
```

```
        }
        if(gp.howManyPlayer==3)
        {
            feathers = new Feather[3];
            feathers[0]= new Feather("xdd", 0, 0);
            feathers[1]= new Feather("xdd", 8, 8);
            feathers[2]= new Feather("xdd", 16, 16);
        }
        if(gp.howManyPlayer==4)
        {
            feathers = new Feather[4];
            feathers[0]= new Feather("xdd", bluefeathertrack
                , bluefeathertrack);
            feathers[1]= new Feather("xdd",
                yellowfeathertrack, yellowfeathertrack);
            feathers[2]= new Feather("xdd",
                whitefeathertrack, whitefeathertrack);
            feathers[3]= new Feather("xdd", redfeathertrack,
                 redfeathertrack);
        //System.out.println(players[0].getName());
    }
}
```

- The class defines several helper methods, including ”setDefaultValue()”, which sets the default positions of the feathers on the game panel, and ”getFeather-Image()”, which loads the feather images from files.

8.4.6 Game Panel and Game Logic Implementation

1. Game Panel

- Constructor of the GamePanel: The constructor of the GamePanel class sets up the panel by adding a mouse listener, setting the preferred size, background color, double buffering, and focusability. It also creates a timer object that fires an action event every 50 milliseconds.

```
public GamePanel(){
        addMouseListener(this);
        this.addMouseListener(new MouseListen());
        this.setPreferredSize(new Dimension(screenWidth,
            screenHeight));
        this.setBackground(Color.BLACK);
        this.setDoubleBuffered(true);
        this.setFocusable(true);
        timer = new Timer(50, this);
        timer.start();
    }
```

- Game Set Up: The constructor of the GamePanel class sets up the panel by adding a mouse listener, setting the preferred size, background color, double

buffering, and focusability. It also creates a timer object that fires an action event every 50 milliseconds.

- StartGameThread() and run(): The startGameThread() method starts a new thread to run the game loop. The run() method is the main game loop. It calculates the time interval between frames (drawInterval) and updates the game logic and screen rendering based on the desired frames per second (FPS). It uses the delta variable to keep track of the time elapsed since the last frame and updates the game logic and screen rendering when the delta reaches 1. It also keeps track of the frames drawn per second (drawCount) and resets the count and timer every second.

```
    public void setupGame(){
        gameState = titleState;
     }
     public void startGameThread() {
        gameThread = new Thread(this);
        gameThread.start();
     }
     //Basically how fps workds
     @Override
     public void run(){
        double drawInterval = 1000000000/FPS;
        double delta = 0;
        long lastTime = System.nanoTime();
        long currentTime;
        long timer=0;
        int drawCount = 0;
        while(gameThread!=null){
            currentTime = System.nanoTime();
            delta+=(currentTime - lastTime)/drawInterval;
            timer += (currentTime - lastTime);
            lastTime = currentTime;
            if(delta>=1){
                    update();
                    repaint();
                delta--;
                drawCount++;
            }
            if(timer>=1000000000){
                // System.out.println("FPS: " + drawCount);
                drawCount = 0;
                timer = 0;
            }
        }
     }
```

- update() method: The update() method is called within the game loop and updates the game logic for the feather manager, player, end game screen, and user interface.

```
public void update(){
        featherM.update();
```

```
        player.update();
        endG.update();
        ui.update();
    }
```

- paintComponent() method: The paintComponent() method is called by the Swing framework to render the panel. It uses the Graphics object to draw various components based on the current game state. The components include the user interface, player, victory screen, warning, octagon map, and feathers.

```
public void paintComponent(Graphics g){
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        //Titlescreen
        if(gameState==titleState){
            ui.draw(g2);
        }
        if(gameState == HTPState) {
            ui.draw(g2);
            ui.drawPlayer(g2);
        }
        if(gameState==choosePState){
            ui.draw(g2);
        }
        if(gameState == endState) {
            ui.draw(g2);
            endG.drawVictory(g2);
            endG.drawWinPlayer(g2,playerwin);
        }
        if(gameState == playState) {

            tileM.draw(g2);
            ui.drawWarning(g2);
            player.drawArrowTurn(g2,currentPlayer);
            octagonM.draw(g2);
            featherM.draw(g2);
            player.draw(g2);

        }
        g2.dispose();

    }
```

2. Game Logic

   a. Variable
   - direction is an integer variable that is initially set to 1. octagons is an array of Octagon objects obtained from the OctagonManager.getArray() method.
   - egg is an array of Tile objects obtained from the TileManager.getArray() method.
   - players is an array of Player objects.
   - feathers is an array of Feather objects.

- drawFeathers is an array of Feather objects used for drawing.
- trackcount is an integer variable used to track the count of a specific event.
- playercheck is an integer variable used to check the current player.
- feather1, feather2, and feather3 are integer variables.

```
int direction =1;
   Octagon[] octagons = OctagonManager.getArray();
   Tile[] egg = TileManager.getArray();
   Player[] players;
   Feather[] feathers;
   Feather[] drawFeathers;

   int trackcount = 0;
   int playercheck = currentPlayer;
   int feather1;
   int feather2;
   int feather3;
```

b. checkChicken(int currentPlayer) method:
   - This method checks if any player is in a specific position based on the number of players (howManyPlayer).
   - For 2 players, it checks if the current player's position is one less than the next player's position.
   - For 3 players, it checks if the current player's position is one less than the next player's position or two less than the player after the next player's position.
   - For 4 players, it checks if the current player's position is one less than the next player's position, two less than the player after the next player's position, or three less than the player after the player after the next player's position.
   - If a match is found, it increments the trackcount and updates the playercheck variable accordingly.
   - It recursively calls itself to check for additional matches.

```
public void checkChicken(int currentPlayer){
       if(howManyPlayer==2)
       {            if((players[playercheck].getPosition()
           +1)%24==players[(playercheck+1)%howManyPlayer].
           getPosition()){
               trackcount+=1;
               playercheck=(playercheck+1)%howManyPlayer;
               checkChicken(currentPlayer);
           }
       }
       if(howManyPlayer==3)
       {            if((players[playercheck].getPosition()
           +1)%24==players[(playercheck+1)%howManyPlayer].
           getPosition()){
               trackcount+=1;
               playercheck=(playercheck+1)%howManyPlayer;
               checkChicken(currentPlayer);
```

```
            }
            if((players[playercheck].getPosition()+1)%24==
                players[(playercheck+2)%howManyPlayer].
                getPosition()){
                trackcount+=1;
                playercheck=(playercheck+2)%howManyPlayer;
                checkChicken(currentPlayer);
            }
        }

        if(howManyPlayer==4)
        { if((players[playercheck].getPosition()+1)%24==
            players[(playercheck+1)%howManyPlayer].
            getPosition()){
                trackcount+=1;
                playercheck=(playercheck+1)%4;
                checkChicken(currentPlayer);
            }
            if((players[playercheck].getPosition()+1)%24==
                players[(playercheck+2)%howManyPlayer].
                getPosition()){
                trackcount+=1;
                playercheck=(playercheck+2)%4;
                checkChicken(currentPlayer);
            }
            if((players[playercheck].getPosition()+1)%24==
                players[(playercheck+3)%4].getPosition()){
                trackcount+=1;
                playercheck=(playercheck+3)%4;
                checkChicken(currentPlayer);
         }
        }
    }
```

c. checkFeather(int currentPlayer) method:
   - This method checks if any player has collected feathers based on the number of players (howManyPlayer).
   - It follows a similar logic as the checkChicken method to check for matches between player positions.
   - If a match is found and a certain condition (dem % 2 == 1) is met, it calls the takeFeather method.
   - It recursively calls itself to check for additional matches.

```
public void checkFeather(int currentPlayer){
        if(howManyPlayer==2)
        {
        if((players[playercheck].getPosition()+1)%24==
            players[(playercheck+1)%howManyPlayer].
            getPosition()){
            trackcount+=1;
            playercheck=(playercheck+1)%howManyPlayer;
            fraudChicken = playercheck;
            if(dem%2==1){
```

```
                takeFeather(currentPlayer, fraudChicken);
            }
            checkFeather(currentPlayer);
        }
        }
        if(howManyPlayer==3)
        {
        if((players[playercheck].getPosition()+1)%24==
            players[(playercheck+1)%howManyPlayer].
            getPosition()){
            trackcount+=1;
            playercheck=(playercheck+1)%howManyPlayer;
            fraudChicken = playercheck;
            if(dem%2==1){
                takeFeather(currentPlayer, fraudChicken);
            }
            checkFeather(currentPlayer);
        }
        if((players[playercheck].getPosition()+1)%24==
            players[(playercheck+2)%howManyPlayer].
            getPosition()){
            trackcount+=1;
            playercheck=(playercheck+2)%howManyPlayer;
            fraudChicken = playercheck;
            if(dem%2==1){
                takeFeather(currentPlayer, fraudChicken);
            }
            checkFeather(currentPlayer);
        }
    }
        if(howManyPlayer==4)
        {
        if((players[playercheck].getPosition()+1)%24==
            players[(playercheck+1)%4].getPosition()){
            trackcount+=1;
            playercheck=(playercheck+1)%4;
            fraudChicken = playercheck;
            if(dem%2==1){
                takeFeather(currentPlayer, fraudChicken);
            }
            checkFeather(currentPlayer);
        }
        if((players[playercheck].getPosition()+1)%24==
            players[(playercheck+2)%4].getPosition()){
            trackcount+=1;
            playercheck=(playercheck+2)%4;
            fraudChicken = playercheck;
            if(dem%2==1){
                takeFeather(currentPlayer, fraudChicken);
            }
            checkFeather(currentPlayer);
        }
        if((players[playercheck].getPosition()+1)%24==
            players[(playercheck+3)%4].getPosition()){
```

```
            trackcount+=1;
            playercheck=(playercheck+3)%4;
            fraudChicken = playercheck;
            if(dem%2==1){
                takeFeather(currentPlayer, fraudChicken);
            }
            checkFeather(currentPlayer);
        }
    }
}
```

d. takeFeather(int currentPlayer, int fraudChicken) method:
   - This method is called when a player takes a feather from another player (fraudChicken).
   - It checks the index of the fraudChicken player to determine how many feathers they have.
   - If the fraudChicken player has more than 0 feathers, it performs various operations based on the number of players (howManyPlayer).
   - It updates the positions of the feathers and drawFeathers arrays based on the positions of the players.
   - It updates the index of the current player and decreases the index of the fraudChicken player.

```
public void takeFeather(int currentPlayer, int fraudChicken)
    {
        feathercheck = players[fraudChicken].getIndex();
        if(feathercheck!=0){
            if(players[fraudChicken].getIndex()==1){
                for(int i=0; i<howManyPlayer; i++)
                {
                    if(feathers[i].getPosition()==players[
                        fraudChicken].getPosition()){
                            feathers[i].setPosition(
                                players[currentPlayer].
                                getPosition());
                            drawFeathers[i].setPosition(
                                players[currentPlayer].
                                getPosition());
                        }
                }
             }
            if(players[fraudChicken].getIndex()>1){
                if(howManyPlayer==4)
                {
                    if(feathers[fraudChicken].getPosition()
                        ==players[fraudChicken].getPosition
                        ()){
                        feathers[fraudChicken].setPosition(
                            players[currentPlayer].
                            getPosition());
```

```
18                          drawFeathers[fraudChicken].
                                setPosition(players[
                                currentPlayer].getPosition());
19                      }
20                      if(feathers[(fraudChicken+1)%4].
                            getPosition()==players[fraudChicken
                            ].getPosition()){
21                          feathers[(fraudChicken+1)%4].
                                setPosition(players[
                                currentPlayer].getPosition());
22                          drawFeathers[(fraudChicken+1)%4].
                                setPosition(players[
                                currentPlayer].getPosition());
23                      }
24                      if(feathers[(fraudChicken+2)%4].
                            getPosition()==players[fraudChicken
                            ].getPosition()){
25                          feathers[(fraudChicken+2)%4].
                                setPosition(players[
                                currentPlayer].getPosition());
26                          drawFeathers[(fraudChicken+2)%4].
                                setPosition(players[
                                currentPlayer].getPosition());
27                      }
28                      if(feathers[(fraudChicken+3)%4].
                            getPosition()==players[fraudChicken
                            ].getPosition()){
29                          feathers[(fraudChicken+3)%4].
                                setPosition(players[
                                currentPlayer].getPosition());
30                          drawFeathers[(fraudChicken+3)%4].
                                setPosition(players[
                                currentPlayer].getPosition());
31                      }
32                  }
33                  if(howManyPlayer==3)
34                  {
35                      if(feathers[fraudChicken].getPosition()
                            ==players[fraudChicken].getPosition
                            ()){
36                          feathers[fraudChicken].setPosition(
                                players[currentPlayer].
                                getPosition());
37                          drawFeathers[fraudChicken].
                                setPosition(players[
                                currentPlayer].getPosition());
38                      }
39                      if(feathers[(fraudChicken+1)%
                            howManyPlayer].getPosition()==
                            players[fraudChicken].getPosition())
                            {
40                          feathers[(fraudChicken+1)%
                                howManyPlayer].setPosition(
```

```
                        players[currentPlayer].
                        getPosition());
41                  drawFeathers[(fraudChicken+1)%
                        howManyPlayer].setPosition(
                        players[currentPlayer].
                        getPosition());
42              }
43              if(feathers[(fraudChicken+2)%
                    howManyPlayer].getPosition()==
                    players[fraudChicken].getPosition())
                    {
44                  feathers[(fraudChicken+2)%
                        howManyPlayer].setPosition(
                        players[currentPlayer].
                        getPosition());
45                  drawFeathers[(fraudChicken+2)%
                        howManyPlayer].setPosition(
                        players[currentPlayer].
                        getPosition());
46              }
47          }
48          if(howManyPlayer==2)
49          {
50              if(feathers[fraudChicken].getPosition()
                    ==players[fraudChicken].getPosition
                    ()){
51                  feathers[fraudChicken].setPosition(
                        players[currentPlayer].
                        getPosition());
52                  drawFeathers[fraudChicken].
                        setPosition(players[
                        currentPlayer].getPosition());
53              }
54              if(feathers[(fraudChicken+1)%
                    howManyPlayer].getPosition()==
                    players[fraudChicken].getPosition())
                    {
55                  feathers[(fraudChicken+1)%
                        howManyPlayer].setPosition(
                        players[currentPlayer].
                        getPosition());
56                  drawFeathers[(fraudChicken+1)%
                        howManyPlayer].setPosition(
                        players[currentPlayer].
                        getPosition());
57              }
58          }
59      }
60      players[currentPlayer].setIndex(players[
            currentPlayer].getIndex()+players[
            fraudChicken].getIndex());
61      players[fraudChicken].setIndex(players[
            fraudChicken].getIndex()-players[
            fraudChicken].getIndex());
```

```
        }
        System.out.println("player"+currentPlayer+": "+
            players[currentPlayer].getIndex());
        System.out.println("player"+fraudChicken+": "+
            players[fraudChicken].getIndex());
    }
```

e. actionPerformed(ActionEvent e)
   - This method is called when an action event occurs, such as a button click.
   - If the game state is in the ”playState,” it performs certain operations related to flipping octagons.
   - If the flip variable is true, it calls the flip() method on the selected octagon (octagons[selected]).
   - If the width of the selected octagon is less than or equal to 0, it checks if the octagon is on the backside or the front side and updates its picture accordingly.
   - Finally, it repaints the panel to update the graphics.

```
public void actionPerformed(ActionEvent e) {
        if(gameState==playState)
        {
                if(flip)
            {
                flip=octagons[selected].flip();
                    if(octagons[selected].getWidth() <=0)
                {
                    if(octagons[selected].getIsBackSide()==
                        true)
                    {
                        octagons[selected].setPicture(
                            octagons[selected].getName());
                    }
                    else{
                        octagons[selected].setPicture("
                            backside");
                    }
                }
            }
        }
       repaint();
        }
```

f. mousePressed(MouseEvent e)
   - This method is called when the mouse button is pressed.
   - It retrieves the x and y coordinates of the mouse press.
   - It sets the `selected` variable to 12 (initial value).
   - If the game state is in the ”playState,” it performs various operations related to player actions.
   - It sets the `flip` variable to true and the `direction` variable to 1.
   - It iterates over the `octagons` array and checks if any octagon collides with the mouse press coordinates.

- If a collision is detected, it updates the `selected` variable to the index of the collided octagon and increments the `dem` variable.
- It calls the `checkChicken` method with the current player to check for chicken-related conditions.
- It updates the `playercheck` variable to the current player.
- If the `trackcount` is greater than or equal to 1, it checks if the name of the selected octagon matches the name of the egg at a specific position.
- If the condition is true, it resets the `trackcount` and checks if `dem` is odd (`dem % 2 == 1`).
- If `dem` is odd, it calls the `checkFeather` method with the current player to check for feather-related conditions.
- If the current player has an index other than 0, it updates the positions of the feathers and drawFeathers arrays based on the current player's position.
- It plays a sound effect and updates the position of the current player.
- It prints the current player's turn.
- If the condition is false, it resets `dem` to 0.
- If `trackcount` is equal to 0, it checks if the name of the selected octagon matches the name of the egg at the current player's position + 1.
- If the condition is true, it checks if `dem` is odd (`dem % 2 == 1`).
- If `dem` is odd, it updates the positions of the feathers and drawFeathers arrays based on the current player's position.
- It plays a sound effect and updates the position of the current player.
- It prints the current player's turn.
- If the condition is false, it resets `dem` to 0.
- It resets `trackcount` to 0.
- If the index of the current player is equal to `howManyPlayer`, it sets the `playerwin` variable to the current player, updates the game state to "endState," plays a sound effect, resets the current player and `dem` variables.

```
1  public void mousePressed(MouseEvent e) {
2
3          int mx = e.getX();
4          int my = e.getY();
5          selected=12;
6
7          if(gameState==playState){
8              flip=true;
9              direction=1;
10             for(Octagon oct1: octagons)
11             {
12                 if(oct1.collision(e.getX(), e.getY()))
13                 {
14                     selected=oct1.getIndex();
15                     dem++;
16                     checkChicken(currentPlayer);
17
18                     playercheck = currentPlayer;
19                     if(trackcount >= 1){
```

```
20                          if(octagons[selected].getName()==egg
                                [(players[currentPlayer].
                                getPosition()+trackcount+1)%24].
                                getName()){
21                              trackcount = 0;
22                              if(dem%2==1){
23                                  checkFeather(currentPlayer);
24                                  if(players[currentPlayer].
                                        getIndex()!=0){
25                                      for(int i=0; i<
                                            howManyPlayer; i++)
26                                  {
27                                      if(feathers[i].
                                            getPosition()==
                                            players[
                                            currentPlayer].
                                            getPosition())
28                                  {
29                                      feathers[i].setPosition
                                            ((players[
                                            currentPlayer].
                                            getPosition()+1+
                                            trackcount)%24);
30                                      drawFeathers[i].
                                            setPosition((players
                                            [currentPlayer].
                                            getPosition()+1+
                                            trackcount)%24);
31                                  }
32                              }
33                                      // feathers[
                                            currentPlayer].
                                            setPosition((players
                                            [currentPlayer].
                                            getPosition()+
                                            trackcount+1)%24);
34                                      // drawFeathers[
                                            currentPlayer].
                                            setPosition((players
                                            [currentPlayer].
                                            getPosition()+
                                            trackcount+1)%24);
35                                  }
36                                  playSE(1);
37                                  players[currentPlayer].
                                        setPosition((players[
                                        currentPlayer].
                                        getPosition()+trackcount
                                        )%24);
38                                  player.playermovement(
                                        currentPlayer);
39                                  //featherM.Feathermovement(
                                        currentPlayer);
```

```
40                          System.out.println("Player "
                                +players[currentPlayer].
                                getName()+" turn");
41                          //System.out.println(players
                                [currentPlayer].
                                getPosition());
42                      }else dem=0;
43                  } else {
44                          if(dem%2==1){
45                              playSE(3);
46                              currentPlayer=(
                                    currentPlayer+1)%
                                    howManyPlayer;
47                              System.out.println("
                                    Player "+players[
                                    currentPlayer].
                                    getName()+" turn");
48                          }else dem=0;
49                      }
50
51              }
52              if(trackcount==0){
53                  if(octagons[selected].getName()==egg
                        [(players[currentPlayer].
                        getPosition()+1)%24].getName()){
54
55                  if(dem%2==1){
56                      if(players[currentPlayer].
                            getIndex()!=0){
57                      for(int i=0; i<howManyPlayer
                            ; i++)
58                      {
59                          if(feathers[i].
                                getPosition()==
                                players[
                                currentPlayer].
                                getPosition())
60                          {
61                          feathers[i].setPosition
                                ((players[
                                currentPlayer].
                                getPosition()+1)%24)
                                ;
62                          drawFeathers[i].
                                setPosition((players
                                [currentPlayer].
                                getPosition()+1)%24)
                                ;
63                          }
64                      }
65                      }
66                      playSE(2);
67                      player.playermovement(
                            currentPlayer);
```

```
68                          //featherM.Feathermovement(
                                currentPlayer);
69                          System.out.println("Player "+
                                players[currentPlayer].
                                getName()+" turn");
70                          //System.out.println(players[
                                currentPlayer].getPosition()
                                );
71                      }else dem=0;
72                  }else{
73                          if(dem%2==1){
74                              playSE(3);
75                              currentPlayer=(currentPlayer
                                    +1)%howManyPlayer;
76                              System.out.println("Player "
                                    +players[currentPlayer].
                                    getName()+" turn");
77                          }else dem=0;
78                      }
79                  }
80                  trackcount=0;
81                  if(players[currentPlayer].getIndex() ==
                        howManyPlayer) {
82                      playerwin = currentPlayer;
83                      gameState = endState;
84                      playSE(4);
85                      currentPlayer=0;
86                      dem=0;
87                  }
88              }
89
90          }
91      }
```

8.5 GUI Details

8.5.1 Title GUI

Class UI: Class UI is responsible about drawing the title game,instruction, choose number of player and the background of game part. It also have all the attributes included in Entity class(by using extends Entity).

1. **Attributes:**

- The constructor initializes the gp variable with the provided GamePanel object.
- GamePanel gp, PlayerManager p, FeatherManager f: It's a variable of type GamePanel, PlayerManager, FeatherManager which is a class, interface representing a game panel, player manager, controller for feathers in a game.

- Feather[] featheranimation1, featheranimation2, featheranimation3, featheranimation4, featheranimation5: These are arrays of type Feather[] representing different sets of feather animations in the game.
- String[] name: It's an array of type String[] containing names for the feathers. Each name corresponds to a specific feather animation.
- playButton, ExitButton, questionButton, chooseButton2, chooseButton3, chooseButton4, BackButton, MainMenu, ExitButton1: Rectangle objects representing buttons or clickable areas in the game interface. These rectangles define the position and size of the buttons.
- MaruMonika and purisaB: Fonts used in the game.
- title: An array of Tile objects. Used for draw background of the game.
- g2: An instance of the Graphics2D class, used for drawing graphics.
- Feather[] feathers = FeatherManager.getArray(): It's a variable of type Feather[] that is being assigned the return value of the getArray() method of the FeatherManager class.

2. **Constructor:**

- The constructor initializes the gp variable with the provided GamePanel object.
- The constructor also creates a new array of Feather objects with a length of 4 and assigns it to the featheranimation1 instance variable. Similarly, four other arrays named featheranimation2, featheranimation3, featheranimation4, and featheranimation5 are created with the same length. These arrays are used to store feather animations.
- Constructor creates an array of Tile objects called title with a length of 10.
- It calls three methods: getTitleImage(), getHTPImage(), and getFeatherImage(). These methods likely load images for the game's title, instructions, and feathers, respectively.
- The constructor attempts to load two fonts from resource files. It uses the getClass().getResourceAsStream() method to get the input stream of the font files located in the "/res/Font/" directory. The fonts are then created using the Font.createFont() method, passing Font.TRUETYPE_FONT and the input stream as parameters. The loaded fonts are assigned to the MaruMonika and purisaB variables.

```
1  public UI(GamePanel gp){
2     this.gp = gp;
3     title = new Tile[10];
4     featheranimation1 = new Feather[4];
5     featheranimation2 = new Feather[4];
6     featheranimation3 = new Feather[4];
7     featheranimation4 = new Feather[4];
8     featheranimation5 = new Feather[4];
9     getTitleImage();
10    getHTPImage();
11    getFeatherImage();
12    try {
```

```
        InputStream is = getClass().getResourceAsStream("/res/
            Font/Purisa Bold.ttf");
        MaruMonika = Font.createFont(Font.TRUETYPE_FONT, is);
        is = getClass().getResourceAsStream("/res/Font/
            x12y16pxMaruMonica.ttf");
        purisaB = Font.createFont(Font.TRUETYPE_FONT,is);
    } catch (FontFormatException e) {

        e.printStackTrace();
    } catch (IOException e) {

        e.printStackTrace();
    }
}
```

3. **getHTPImage():**

   - Overall, the getHTPImage() method is responsible for loading a series of images representing different frames or states of the chicken animations used in the game's instructions. It also load a warning image run through the game state.

```
public void getHTPImage() {
   try{
       chick1 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/BlueChicken/
           chickenpixel.png"));
       chick2 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/BlueChicken/
           chickenpixel2.png"));
       chick3 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/BlueChicken/
           chickenpixel3.png"));
       chick4 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/BlueChicken/
           chickenpixel4.png"));
       chick5 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/BlueChicken/
           chickenpixel3.png"));
       chick6 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/YellowChicken/
           chickenpixel.png"));
       chick7 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/YellowChicken/
           chickenpixel2.png"));
       chick8 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/YellowChicken/
           chickenpixel3.png"));
       chick9 = ImageIO.read(getClass().getResourceAsStream
           ("/res/chicken animation deluxe/YellowChicken/
           chickenpixel4.png"));
       chick10 = ImageIO.read(getClass().
           getResourceAsStream("/res/chicken animation
           deluxe/YellowChicken/chickenpixel3.png"));
```

```
13          chick11 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/WhiteChicken/chickenpixel.png"));
14          chick12 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/WhiteChicken/chickenpixel2.png"));
15          chick13 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/WhiteChicken/chickenpixel3.png"));
16          chick14 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/WhiteChicken/chickenpixel4.png"));
17          chick15 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/WhiteChicken/chickenpixel3.png"));
18          chick16 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/RedChicken/chickenpixel.png"));
19          chick17 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/RedChicken/chickenpixel2.png"));
20          chick18 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/RedChicken/chickenpixel3.png"));
21          chick19 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/RedChicken/chickenpixel4.png"));
22          chick20 = ImageIO.read(getClass().
                getResourceAsStream("/res/chicken animation
                deluxe/RedChicken/chickenpixel3.png"));
23          warning = ImageIO.read(getClass().
                getResourceAsStream("/res/arrow/warning.png"));
24          warning2 = ImageIO.read(getClass().
                getResourceAsStream("/res/arrow/warning2.png"));
25      }catch(Exception e){
26          e.printStackTrace();
27      }
28  }
```

4. **getFeatherImage():**

   - the getFeatherImage() method creates Feather objects and assigns them to the appropriate positions in the featheranimation1, featheranimation2, featheranimation3, featheranimation4, and featheranimation5 arrays, based on the names and parameters provided.

```
1   public void getFeatherImage(){
2       try{
3
4           for(int i = 0;i < 4;i++){
5               featheranimation1[i] = new Feather(name[i],i, 0)
                    ;
```

```
            featheranimation2[i] = new Feather(name[i+4],i,
                1);
            featheranimation3[i] = new Feather(name[i+8],i,
                2);
            featheranimation4[i] = new Feather(name[i+12],i,
                 3);
            featheranimation5[i] = new Feather(name[i+8],i,
                2);
        }

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

5. **getTitleImage():**

- Overall, the getTitleImage() method initializes and sets up the necessary title images by creating Tile objects and associating them with specific image files. These images are likely used in the title screen and end screen of the game to provide visual elements and create the desired atmosphere or theme.

```
public void getTitleImage(){

        title[0] = new Tile("background2", 0);
        title[1] = new Tile("endbackground",1);
        title[2] = new Tile("question",2);
}
```

6. **update() method:**

- Overall, the update() method is responsible for updating variables related to the game's state and animation frames. It handles the state of the "Instructions" screen (end variable) and updates counters and numbers used for animation sequences (endCounter, endNum, spriteCounter, spriteNum).
- This fuction is call in function update() of GamePanel so that it will update frequently and the chicken will look like it's dancing.

```
public void update(){
    if(gp.gameState == gp.HTPState){
        end = "normal";
    }else{
        end = "off";
    }
    if(gp.gameState == gp.HTPState){
        endCounter++;
        spriteCounter++;
    }
    if(endCounter > 20){
        if(endNum==1){
            endNum=2;
        }
```

```
        else if(endNum==2){
            endNum=1;
        }
        endCounter = 0;
    }
    if(spriteCounter > 20){
        if(spriteNum == 1){
            spriteNum=2;
        }
        else if(spriteNum == 2){
            spriteNum=3;
        }
        else if(spriteNum == 3){
            spriteNum=4;
        }
        else if(spriteNum == 4){
            spriteNum=5;
        }
        else if(spriteNum == 5){
            spriteNum=6;
        }
        else if(spriteNum == 6){
            spriteNum=7;
        }
        else if(spriteNum == 7){
            spriteNum=8;
        }
        else if(spriteNum == 8){
            spriteNum=1;
        }
        spriteCounter = 0;
    }
}
```

7. **drawPlayer():**

- The drawPlayer(Graphics2D g2) method is responsible for drawing the player's character and associated feathers on the game screen.
- The method uses a switch statement based on the value of the end variable. If the end variable is "normal", it executes the corresponding code block. Each time spriteNum is updated, the picture of chicken change and eventually it looks like the chicken and feather is dancing jointly.

```
public void drawPlayer(Graphics2D g2){
    BufferedImage blue = null;
    BufferedImage yellow = null;
    BufferedImage white = null;
    BufferedImage red = null;
    BufferedImage bluefeather = null;
    BufferedImage yellowfeather = null;
    BufferedImage whitefeather = null;
    BufferedImage redfeather = null;
    switch (end) {
```

```
11          case "normal":
12              if(spriteNum==1){
13                  blue = chick1;
14                  yellow = chick6;
15                  white = chick11;
16                  red = chick16;
17                  bluefeather = featheranimation1[0].
                        setPicture(name[0]);
18                  yellowfeather = featheranimation1[1].
                        setPicture(name[1]);
19                  whitefeather = featheranimation1[2].
                        setPicture(name[2]);
20                  redfeather = featheranimation1[3].setPicture
                        (name[3]);
21              }
22              if(spriteNum==2){
23                  blue = chick2;
24                  yellow = chick7;
25                  white = chick12;
26                  red = chick17;
27                  bluefeather = featheranimation2[0].
                        setPicture(name[0+4]);
28                  yellowfeather = featheranimation2[1].
                        setPicture(name[1+4]);
29                  whitefeather = featheranimation2[2].
                        setPicture(name[2+4]);
30                  redfeather = featheranimation2[3].setPicture
                        (name[3+4]);
31              }
32              if(spriteNum==3){
33                  blue = chick3;
34                  yellow = chick8;
35                  white = chick13;
36                  red = chick18;
37                  bluefeather = featheranimation3[0].
                        setPicture(name[0+8]);
38                  yellowfeather = featheranimation3[1].
                        setPicture(name[1+8]);
39                  whitefeather = featheranimation3[2].
                        setPicture(name[2+8]);
40                  redfeather = featheranimation3[3].setPicture
                        (name[3+8]);
41              }
42              if(spriteNum==4){
43                  blue = chick4;
44                  yellow = chick9;
45                  white = chick14;
46                  red = chick19;
47                  bluefeather = featheranimation4[0].
                        setPicture(name[0+12]);
48                  yellowfeather = featheranimation4[1].
                        setPicture(name[1+12]);
49                  whitefeather = featheranimation4[2].
                        setPicture(name[2+12]);
```

```
                redfeather = featheranimation4[3].setPicture
                    (name[3+12]);
            }
            if(spriteNum==5){
                blue = chick5;
                yellow = chick10;
                white = chick15;
                red = chick20;
                bluefeather = featheranimation5[0].
                    setPicture(name[0+8]);
                yellowfeather = featheranimation5[1].
                    setPicture(name[1+8]);
                whitefeather = featheranimation5[2].
                    setPicture(name[2+8]);
                redfeather = featheranimation5[3].setPicture
                    (name[3+8]);
            }
            if(spriteNum>=6 &&spriteNum<=8){
                blue = chick1;
                yellow = chick6;
                white = chick11;
                red = chick16;
                bluefeather = featheranimation1[0].
                    setPicture(name[0]);
                yellowfeather = featheranimation1[1].
                    setPicture(name[1]);
                whitefeather = featheranimation1[2].
                    setPicture(name[2]);
                redfeather = featheranimation1[3].setPicture
                    (name[3]);
            }
            break;
        default:
            break;
    }
            g2.drawImage(bluefeather,28,70,75,75,null);
            g2.drawImage(blue, 28,70,75,75, null);

            g2.drawImage(yellowfeather,720,70,75,75,null);
            g2.drawImage(yellow, 720 ,70, 75,75, null);

            g2.drawImage(whitefeather,28,530,75,75,null);
            g2.drawImage(white, 28 ,530, 75,75, null);

            g2.drawImage(redfeather,720,530,75,75,null);
            g2.drawImage(red, 720 ,530, 75,75, null);
}
```

8. **draw():**

- The draw(Graphics2D g2) method acts as a central hub for drawing different screens or states of the game based on the current game state. It ensures that

the appropriate screen is rendered on the game screen, providing players with the necessary visuals and information to interact with the game.

```
public void draw(Graphics2D g2){
    this.g2 = g2;

    g2.setFont(purisaB);
    g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING
        , RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
    //Title State
    if(gp.gameState == gp.titleState) {
        drawTitleScreen();
    }

    if(gp.gameState == gp.HTPState) {
        drawHTPScreen();
    }
    if(gp.gameState == gp.choosePState) {
        drawPlayerScreen();
    }
    if(gp.gameState == gp.endState) {
        drawEndScreen();
    }
}
```

9. **drawTitleScreen():**

- the drawTitleScreen() method handles the rendering of the title screen, including the background image, visual elements, sub-window, menu options (”Start Game” and ”Exit Game”), and any associated selection indicators.

```
public void drawTitleScreen(){
    // g2.setFont(g2.getFont().deriveFont(Font.BOLD,66F));
    // String text = "Zicke Zacke H hnerkacke";
    String text = "xdd";
    int x;
    int y = gp.tileSize*3;
    //Box

    // g2.setColor(Color.white);
    // g2.drawString(text, x, y);

    g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING
        , RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
    g2.drawImage(title[0].getPicture(), 0, 0, gp.screenWidth
        , gp.screenHeight, null);
    g2.drawImage(title[2].getPicture(), 758, 10, 48, 70,
        null);
    int width = gp.tileSize*5;
    int height = gp.tileSize*2;
    int Xbox = getXforCenteredText(text)-gp.tileSize*3;
    int Ybox = gp.tileSize*6;
    drawSubWindow(286, 303,width,height);
```

```
    //Menu


    g2.setColor(Color.white);
    g2.setFont(g2.getFont().deriveFont(Font.BOLD,30F));
    text = "Start Game";
    x = getXforCenteredText(text);
    y += gp.tileSize*4;
    g2.drawString(text, x, y);
    if(commandNum == 1){
        g2.drawString(">", x - gp.tileSize, y);
    }


    text = "Exit Game";
    x = getXforCenteredText(text);
    y += gp.tileSize;
    g2.drawString(text, x, y);
    if(commandNum == 2){
        g2.drawString(">", x - gp.tileSize, y);
    }
    g2.setColor(new Color(255,255,255, 0));
    g2.draw(playButton);
    g2.draw(ExitButton);


}
```

10. **drawHTPScreen():**

- The drawHTPScreen() method handles the rendering of the ”How to Play” screen, including the background image, sub-window, and the explanations of the game rules. The text is drawn in a structured manner, with headings and paragraphs, to provide clear instructions to the players.

```
public void drawHTPScreen(){
    String text = "xdd";
    int x;
    int y = gp.tileSize + 72;

    //Box
    g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING
        , RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
    g2.drawImage(title[1].getPicture(), 0, 0, gp.screenWidth
        , gp.screenHeight, null);
    g2.drawImage(title[2].getPicture(), 758, 10, 48, 70,
        null);
    int width = gp.tileSize*15;
    int height = gp.tileSize*10;
    int Xbox = getXforCenteredText(text)-gp.tileSize*3;
    int Ybox = gp.tileSize*6;
    drawSubWindow(48, 100, width, height);

```

```
16      //Menu
17      g2.setColor(Color.white);
18      g2.setFont(g2.getFont().deriveFont(Font.BOLD,20F));
19      text = "Taking Turns:";
20      x = 96;
21      y += gp.tileSize;
22      g2.drawString(text, x, y);
23
24      text = "Players take turns by looking at the next space
            in front of their chicken. The current \nplayer then
             choose an octagonal tile. If the picture on the
            tile matches, the player \ncan move their chicken to
             that space. If the picture does not match, the
            current \nplayer stays on the tile it has reached,
            and the next player takes a turn.";
25      x = 72;
26      y += 25;
27      for(String line : text.split("\n")){
28          g2.drawString(line, x, y);
29          y += 25;
30      };
31      text = "Overtaking:";
32      x = 96;
33      y += gp.tileSize/2;
34      g2.drawString(text, x, y);
35
36      text = "If a chicken catches up to another chicken, it
            can attempt to overtake by finding the \nmatching
            tile for the space in front of the other chicken. If
             successful, the \novertaking chicken jumps in front
             and steals all the feathers. The player's turn\
            ncontinues. A chicken can overtake multiple chickens
             in one go if they are standing\nin an unbroken row.
            ";
37      x = 72;
38      y += 25;
39      for(String line : text.split("\n")){
40          g2.drawString(line, x, y);
41          y += 25;
42      };
43
44      text = "Winning Condition:";
45      x = 96;
46      y += gp.tileSize/2;
47      g2.drawString(text, x, y);
48
49      text = "The first chicken to collect all feathers of
            other players is the winner.";
50      x = 72;
51      y += 25;
52      g2.drawString(text, x, y);
53
54
55      g2.setColor(new Color(255,255,255, 0));
```

```
56
57 }
```

11. **drawPlayerScreen():**

- The drawPlayerScreen() method handles the rendering of the player selection screen, including the background image, sub-windows, menu options (”2 Players,” ”3 Players,” ”4 Players,” and ”Back”), and their associated selection indicators. The method also draws the outlines of the buttons on the screen.

```
1  public void drawPlayerScreen(){
2
3      String text = "xdd";
4      int x;
5      int y = gp.tileSize*3;
6
7      //Box
8      g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING
           , RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
9      g2.drawImage(title[0].getPicture(), 0, 0, gp.screenWidth
           , gp.screenHeight, null);
10     int width = gp.tileSize*5;
11     int height = gp.tileSize*3;
12     int Xbox = getXforCenteredText(text)-gp.tileSize*3;
13     int Ybox = gp.tileSize*6;
14     drawSubWindow(286, 303,width,height);
15     drawSubWindow(286, 447,width,gp.tileSize);
16
17     //Menu
18     g2.setColor(Color.white);
19     g2.setFont(g2.getFont().deriveFont(Font.BOLD,30F));
20     text = "2 Players";
21     x = getXforCenteredText(text);
22     y += gp.tileSize*4;
23     g2.drawString(text, x, y);
24     if(commandNum == 3){
25         g2.drawString(">", x - gp.tileSize, y);
26     }
27
28     text = "3 Players";
29     x = getXforCenteredText(text);
30     y += gp.tileSize;
31     g2.drawString(text, x, y);
32     if(commandNum == 4){
33         g2.drawString(">", x - gp.tileSize, y);
34     }
35
36     text = "4 Players";
37     x = getXforCenteredText(text);
38     y += gp.tileSize;
39     g2.drawString(text, x, y);
40     if(commandNum == 5){
41         g2.drawString(">", x - gp.tileSize, y);
```

```
    }
    text = "Back";
    x = getXforCenteredText(text);
    y += gp.tileSize;
    g2.drawString(text, x, y);
    if(commandNum == 6){
        g2.drawString(">", x - gp.tileSize, y);
    }
    g2.setColor(new Color(255,255,255, 0));
    g2.draw(chooseButton2);
    g2.draw(chooseButton3);
    g2.draw(chooseButton4);
    g2.draw(BackButton);
}
```

12. **drawEndScreen():**

- The drawEndScreen() method handles the rendering of the end screen, including the background image, sub-windows, menu options (”Main Menu” and ”Exit Game”), and their associated selection indicators. The method also draws the outlines of the buttons on the screen.

```
public void drawEndScreen() {
    String text = "xdd";
    int x;
    int y = gp.tileSize*3;

    g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING
        , RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
    g2.drawImage(title[1].getPicture(), 0, 0, gp.screenWidth
        , gp.screenHeight, null);
    int width = gp.tileSize*4;
    int height = gp.tileSize;
    int Xbox = getXforCenteredText(text)-gp.tileSize*3;
    int Ybox = gp.tileSize*6;
    drawSubWindow(573, 350,width,height);
    drawSubWindow(53, 350,width,height);

    g2.setColor(Color.white);
    g2.setFont(g2.getFont().deriveFont(Font.BOLD,30F));
    text = "Main Menu";
    x = 80;
    y = gp.tileSize * 8;
    g2.drawString(text, x, y);
    if(commandNum == 7){
        g2.drawString(">", x - gp.tileSize, y);
    }

    text = "Exit Game";
    x = 610;
    y = gp.tileSize*8;
    g2.drawString(text, x, y);
    if(commandNum == 8){
```

```
30          g2.drawString(">", x - gp.tileSize, y);
31      }
32      g2.setColor(new Color(255,255,255, 0));
33      g2.draw(MainMenu);
34      g2.draw(ExitButton1);
35
36  }
```

13. **drawWarning():**

- The drawWarning() method sets up a sub-window as the background and then draws a warning message image on top of it. It is used to display a warning message to the user on the screen.

```
1  public void drawWarning(Graphics2D g2){
2      this.g2 = g2;
3      drawSubWindow(60, 573,720,43);
4      g2.drawImage(warning2, 70, 580, 700, 31, null);
5  }
```

14. **drawSubWindow():**

- The drawSubWindow() method draws a semi-transparent black filled rectangle with rounded corners as the background of a sub-window. It then draws a light blue outline with rounded corners for the sub-window. This method is useful for creating visually appealing sub-windows in the game interface.

```
1  public void drawSubWindow(int x, int y, int width, int
       height){
2      Color c = new Color(0,0,0,200);
3      g2.setColor(c);
4      g2.fillRoundRect(x, y, width, height, 35, 35);
5
6      c = new Color(150, 203, 217);
7      g2.setColor(c);
8      g2.setStroke(new BasicStroke(3));
9      g2.drawRoundRect(x+3, y+3, width-6, height-6, 29, 29);
10 }
```

15. **getXforCenteredText():**

- the getXforCenteredText() method calculates the x-coordinate for drawing text at the center of the screen by first determining the width of the text and then subtracting half of that width from the x-coordinate of the screen's center. This ensures that the text is horizontally centered on the screen.

```
1  public int getXforCenteredText(String text){
2      int length = (int)g2.getFontMetrics().getStringBounds(
           text,g2).getWidth();
3      int x = gp.screenWidth/2 - length/2;
4      return x;
5  }
```

8.5.2 Game-play GUI

GamePlay draw: The graphics of the game is made mostly by Adobe Photoshop and the animation is made by changing the picture of chicken and feather frequently. Noted that all the animation in the game has been done in this way.

1. **How animation is made with the game FPS in GamePanel class**

   - The run() method implements the game loop, which continuously updates and renders the game at the desired frame rate. It also keeps track of the FPS count for monitoring and debugging purposes.

```
public void run(){
        double drawInterval = 1000000000/FPS;
        double delta = 0;
        long lastTime = System.nanoTime();
        long currentTime;
        long timer=0;
        int drawCount = 0;

        while(gameThread!=null){
            currentTime = System.nanoTime();
            delta+=(currentTime - lastTime)/drawInterval;
            timer += (currentTime - lastTime);
            lastTime = currentTime;
            if(delta>=1){
                    update();
                    repaint();
                delta--;
                drawCount++;
            }
            if(timer>=1000000000){
                // System.out.println("FPS: " + drawCount);
                drawCount = 0;
                timer = 0;
            }
        }
    }
```

   - The update() function is called inside of the run function, therefore the game will update constantly. featherM.update(), player.update(), endG.update(), ui.update() will be updated continously.

```
public void update(){
        featherM.update();
        player.update();
        endG.update();
        ui.update();
    }
```

   - In the play state, the direction will be set to ”normal” and the variables call spriteCounter will be update.

- When the spriteCounter reach 20, the spriteNum will be updated as it start from 1 and then reach 8. After reaching 8 the spriteCounter will be set to 0 again and the loop continue.
- While the above loop is run, each time the spriteNum is set from 1 to 8, a setPicture() function is called so that the draw() can draw the chichkens or feathers. spriteNum is modified constanly so the picture will be changed continuesly which results in a chicken dancing with feather.
- **featherM.update() and featherM.draw():**

```
public void update(){
        public void update(){

        if(gp.gameState == gp.playState){
            direction = "normal";
        }else{
            direction = "off";
        }
        if(gp.gameState == gp.playState){
            spriteCounter++;
        }

        if(spriteCounter > 20){
            if(spriteNum == 1){
                spriteNum=2;
            }
            else if(spriteNum == 2){
                spriteNum=3;
            }
            else if(spriteNum == 3){
                spriteNum=4;
            }
            else if(spriteNum == 4){
                spriteNum=5;
            }
            else if(spriteNum == 5){
                spriteNum=6;
            }
            else if(spriteNum == 6){
                spriteNum=7;
            }
            else if(spriteNum == 7){
                spriteNum=8;
            }
            else if(spriteNum == 8){
                spriteNum=1;
            }
            spriteCounter = 0;

        }

    }
    }
    public void draw(Graphics2D g2){
```

```
45          bluefeather = null;
46          yellowfeather = null;
47          whitefeather = null;
48          redfeather = null;
49          switch (direction) {
50              case "normal":
51                  if(spriteNum==1){
52                      bluefeather = featheranimation1[0].
                            setPicture(name[0]);
53                      yellowfeather = featheranimation1[1].
                            setPicture(name[1]);
54                      whitefeather = featheranimation1[2].
                            setPicture(name[2]);
55                      redfeather = featheranimation1[3].
                            setPicture(name[3]);
56                  }
57                  if(spriteNum==2){
58                      bluefeather = featheranimation2[0].
                            setPicture(name[0+4]);
59                      yellowfeather = featheranimation2[1].
                            setPicture(name[1+4]);
60                      whitefeather = featheranimation2[2].
                            setPicture(name[2+4]);
61                      redfeather = featheranimation2[3].
                            setPicture(name[3+4]);
62                  }
63                  if(spriteNum==3){
64                      bluefeather = featheranimation3[0].
                            setPicture(name[0+8]);
65                      yellowfeather = featheranimation3[1].
                            setPicture(name[1+8]);
66                      whitefeather = featheranimation3[2].
                            setPicture(name[2+8]);
67                      redfeather = featheranimation3[3].
                            setPicture(name[3+8]);
68                  }
69                  if(spriteNum==4){
70                      bluefeather = featheranimation4[0].
                            setPicture(name[0+12]);
71                      yellowfeather = featheranimation4[1].
                            setPicture(name[1+12]);
72                      whitefeather = featheranimation4[2].
                            setPicture(name[2+12]);
73                      redfeather = featheranimation4[3].
                            setPicture(name[3+12]);
74                  }
75                  if(spriteNum==5){
76                      bluefeather = featheranimation5[0].
                            setPicture(name[0+8]);
77                      yellowfeather = featheranimation5[1].
                            setPicture(name[1+8]);
78                      whitefeather = featheranimation5[2].
                            setPicture(name[2+8]);
```

```
79                      redfeather = featheranimation5[3].
                            setPicture(name[3+8]);
80                  }
81                  if(spriteNum>=6 &&spriteNum<=8){
82                      bluefeather = featheranimation1[0].
                            setPicture(name[0]);
83                      yellowfeather = featheranimation1[1].
                            setPicture(name[1]);
84                      whitefeather = featheranimation1[2].
                            setPicture(name[2]);
85                      redfeather = featheranimation1[3].
                            setPicture(name[3]);
86                  }
87                  break;
88
89              default:
90                  break;
91          }
92          if(gp.howManyPlayer==2)
93          {
94              g2.drawImage(bluefeather, xy[(feathers[0].
                    getPosition())%24].getX(),xy[(feathers[0].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
95              g2.drawImage(yellowfeather, xy[(feathers[1].
                    getPosition())%24].getX(),xy[(feathers[1].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
96          }
97          if(gp.howManyPlayer==3)
98          {
99              g2.drawImage(bluefeather, xy[(feathers[0].
                    getPosition())%24].getX(),xy[(feathers[0].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
100             g2.drawImage(yellowfeather, xy[(feathers[1].
                    getPosition())%24].getX(),xy[(feathers[1].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
101             g2.drawImage(whitefeather, xy[(feathers[2].
                    getPosition())%24].getX(),xy[(feathers[2].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
102         }
103         if(gp.howManyPlayer==4)
104         {
105             g2.drawImage(bluefeather, xy[(feathers[0].
                    getPosition())%24].getX(),xy[(feathers[0].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
106             g2.drawImage(yellowfeather, xy[(feathers[1].
                    getPosition())%24].getX(),xy[(feathers[1].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
```

```
107             g2.drawImage(whitefeather, xy[(feathers[2].
                    getPosition())%24].getX(),xy[(feathers[2].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
108             g2.drawImage(redfeather, xy[(feathers[3].
                    getPosition())%24].getX(),xy[(feathers[3].
                    getPosition())%24].getY(), gp.chickensize,
                    gp.chickensize, null);
109         }
110     }
```

- **playerM.update() and playerM.update():**

```
1  public void update(){
2          if(gp.gameState == gp.playState){
3              direction = "normal";
4              direction2 = "normal";
5          }else{
6              direction = "off";
7              direction2 = "off";
8          }
9          if(gp.gameState == gp.playState){
10             spriteCounter++;
11             arrowCounter++;
12         }
13         if(arrowCounter > 23){
14             if(arrowNum==1){
15                 arrowNum=2;
16             }
17             else if(arrowNum==2){
18                 arrowNum=1;
19             }
20             arrowCounter = 0;
21         }
22         if(spriteCounter > 20){
23             if(spriteNum == 1){
24                 spriteNum=2;
25             }
26             else if(spriteNum == 2){
27                 spriteNum=3;
28             }
29             else if(spriteNum == 3){
30                 spriteNum=4;
31             }
32             else if(spriteNum == 4){
33                 spriteNum=5;
34             }
35             else if(spriteNum == 5){
36                 spriteNum=6;
37             }
38             else if(spriteNum == 6){
39                 spriteNum=7;
40             }
41             else if(spriteNum == 7){
```

```
42                  spriteNum=8;
43              }
44              else if(spriteNum == 8){
45                  spriteNum=1;
46              }
47              spriteCounter = 0;
48          }
49      }
50      public void draw(Graphics2D g2){
51          BufferedImage blue = null;
52          BufferedImage yellow = null;
53          BufferedImage white = null;
54          BufferedImage red = null;
55
56          switch (direction) {
57              case "normal":
58                  if(spriteNum==1){
59                      blue = chick1;
60                      yellow = chick6;
61                      white = chick11;
62                      red = chick16;
63                  }
64                  if(spriteNum==2){
65                      blue = chick2;
66                      yellow = chick7;
67                      white = chick12;
68                      red = chick17;
69                  }
70                  if(spriteNum==3){
71                      blue = chick3;
72                      yellow = chick8;
73                      white = chick13;
74                      red = chick18;
75                  }
76                  if(spriteNum==4){
77                      blue = chick4;
78                      yellow = chick9;
79                      white = chick14;
80                      red = chick19;
81                  }
82                  if(spriteNum==5){
83                      blue = chick5;
84                      yellow = chick10;
85                      white = chick15;
86                      red = chick20;
87                  }
88                  if(spriteNum>=6 &&spriteNum<=8){
89                      blue = chick1;
90                      yellow = chick6;
91                      white = chick11;
92                      red = chick16;
93                  }
94                  break;
95
```

```
96                 default:
97                     break;
98             }
99
100            if(gp.howManyPlayer==2)
101            {
102                g2.drawImage(blue, xy[players[0].getPosition()].
                       getX() , xy[players[0].getPosition()].getY()
                       , gp.chickensize, gp.chickensize, null);
103                g2.drawImage(yellow, xy[players[1].getPosition()
                       ].getX() , xy[players[1].getPosition()].getY
                       (), gp.chickensize, gp.chickensize, null);
104            }
105            if(gp.howManyPlayer==3)
106            {
107                g2.drawImage(blue, xy[players[0].getPosition()].
                       getX() , xy[players[0].getPosition()].getY()
                       , gp.chickensize, gp.chickensize, null);
108                g2.drawImage(yellow, xy[players[1].getPosition()
                       ].getX() , xy[players[1].getPosition()].getY
                       (), gp.chickensize, gp.chickensize, null);
109                g2.drawImage(white, xy[players[2].getPosition()
                       ].getX() , xy[players[2].getPosition()].getY
                       (), gp.chickensize, gp.chickensize, null);
110            }
111            if(gp.howManyPlayer==4)
112            {
113                g2.drawImage(blue, xy[players[0].getPosition()].
                       getX() , xy[players[0].getPosition()].getY()
                       , gp.chickensize, gp.chickensize, null);
114                g2.drawImage(yellow, xy[players[1].getPosition()
                       ].getX() , xy[players[1].getPosition()].getY
                       (), gp.chickensize, gp.chickensize, null);
115                g2.drawImage(white, xy[players[2].getPosition()
                       ].getX() , xy[players[2].getPosition()].getY
                       (), gp.chickensize, gp.chickensize, null);
116                g2.drawImage(red, xy[players[3].getPosition()].
                       getX() , xy[players[3].getPosition()].getY()
                       , gp.chickensize, gp.chickensize, null);
117            }
118        }
```

8.5.3 Winning GUI

Class endUI: Class endUI is responsible about drawing the end game part. It also have all the attributes included in Entity class(by using extends Entity).

1. **Attributes:**

- GamePanel gp, PlayerManager p, FeatherManager f: It's a variable of type GamePanel, PlayerManager, FeatherManager which is a class, interface representing a game panel, player manager, controller for feathers in a game.
- Feather[] featheranimation1, featheranimation2, featheranimation3, featheranimation4, featheranimation5: These are arrays of type Feather[] representing different sets of feather animations in the game.
- String[] name: It's an array of type String[] containing names for the feathers. Each name corresponds to a specific feather animation.
- MapCoordinate[] xy: It's an array of type MapCoordinate[], which may represent coordinates or positions for the feathers on a map.
- Feather[] feathers = FeatherManager.getArray(): It's a variable of type Feather[] that is being assigned the return value of the getArray() method of the FeatherManager class.

2. **Constructor:**

- The constructor creates a new array of MapCoordinate objects with a length of 24 and assigns it to the xy instance variable.
- The constructor also creates a new array of Feather objects with a length of 4 and assigns it to the featheranimation1 instance variable. Similarly, four other arrays named featheranimation2, featheranimation3, featheranimation4, and featheranimation5 are created with the same length. These arrays are used to store feather animations.
- Then it call setDefaultValue() method, getEndImage() and getFeatherImage() method.

```
public EndUI(GamePanel gp ){
    this.gp = gp;

    xy = new MapCoordinate[24];
    featheranimation1 = new Feather[4];
    featheranimation2 = new Feather[4];
    featheranimation3 = new Feather[4];
    featheranimation4 = new Feather[4];
    featheranimation5 = new Feather[4];
    setDefaultValue();
    getEndImage();
    getFeatherImage();
}
```

3. **getFeatherImage():**

- the getFeatherImage() method creates Feather objects and assigns them to the appropriate positions in the featheranimation1, featheranimation2, featheranimation3, featheranimation4, and featheranimation5 arrays, based on the names and parameters provided.

```
public void getFeatherImage(){
    try{

```

```
        for(int i = 0;i < 4;i++){
            featheranimation1[i] = new Feather(name[i],i, 0)
                ;
            featheranimation2[i] = new Feather(name[i+4],i,
                1);
            featheranimation3[i] = new Feather(name[i+8],i,
                2);
            featheranimation4[i] = new Feather(name[i+12],i,
                 3);
            featheranimation5[i] = new Feather(name[i+8],i,
                2);
        }

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

4. **getEndImage() method:**

   - the getEndImage() method loads the necessary images for the end screen of the game and assigns them to the corresponding variables for later use.

```
public void getEndImage(){
    try{
        victory_up = ImageIO.read(getClass().
            getResourceAsStream("/res/victory animation/
            victory_up.png"));
        victory_down= ImageIO.read(getClass().
            getResourceAsStream("/res/victory animation/
            victory_down.png"));
        chick1 = ImageIO.read(getClass().getResourceAsStream("
            /res/chicken animation deluxe/BlueChicken/
            chickenpixel.png"));
        chick2 = ImageIO.read(getClass().getResourceAsStream("
            /res/chicken animation deluxe/BlueChicken/
            chickenpixel2.png"));
        chick3 = ImageIO.read(getClass().getResourceAsStream("
            /res/chicken animation deluxe/BlueChicken/
            chickenpixel3.png"));
        chick4 = ImageIO.read(getClass().getResourceAsStream("
            /res/chicken animation deluxe/BlueChicken/
            chickenpixel4.png"));
        chick5 = ImageIO.read(getClass().getResourceAsStream("
            /res/chicken animation deluxe/BlueChicken/
            chickenpixel3.png"));
        chick6 = ImageIO.read(getClass().getResourceAsStream("
            /res/chicken animation deluxe/YellowChicken/
            chickenpixel.png"));
        chick7 = ImageIO.read(getClass().getResourceAsStream("
            /res/chicken animation deluxe/YellowChicken/
            chickenpixel2.png"));
```

```
12          chick8 = ImageIO.read(getClass().getResourceAsStream("
                /res/chicken animation deluxe/YellowChicken/
                chickenpixel3.png"));
13          chick9 = ImageIO.read(getClass().getResourceAsStream("
                /res/chicken animation deluxe/YellowChicken/
                chickenpixel4.png"));
14          chick10 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/YellowChicken/
                chickenpixel3.png"));
15          chick11 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/WhiteChicken/
                chickenpixel.png"));
16          chick12 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/WhiteChicken/
                chickenpixel2.png"));
17          chick13 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/WhiteChicken/
                chickenpixel3.png"));
18          chick14 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/WhiteChicken/
                chickenpixel4.png"));
19          chick15 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/WhiteChicken/
                chickenpixel3.png"));
20          chick16 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/RedChicken/
                chickenpixel.png"));
21          chick17 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/RedChicken/
                chickenpixel2.png"));
22          chick18 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/RedChicken/
                chickenpixel3.png"));
23          chick19 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/RedChicken/
                chickenpixel4.png"));
24          chick20 = ImageIO.read(getClass().getResourceAsStream(
                "/res/chicken animation deluxe/RedChicken/
                chickenpixel3.png"));
25      }catch(Exception e){
26          e.printStackTrace();
27      }
28  }
```

5. **update() method:**

   - Overall, the update() method is used to update the end, endNum, and spriteNum variables based on the current game state and counters. These variables are used for managing and animating the end screen of chicken and feathers in the end game UI
   - This fuction is call in function update() of GamePanel so that it will update frequently and the chicken will look like it's dancing.

```
public void update(){

    if(gp.gameState == gp.endState){
        end = "normal";
    }else{
        end = "off";
    }
    if(gp.gameState == gp.endState){
        endCounter++;
        spriteCounter++;
    }
    if(endCounter > 20){
        if(endNum==1){
            endNum=2;
        }
        else if(endNum==2){
            endNum=1;
        }
        endCounter = 0;
    }
    if(spriteCounter > 20){
        if(spriteNum == 1){
            spriteNum=2;
        }
        else if(spriteNum == 2){
            spriteNum=3;
        }
        else if(spriteNum == 3){
            spriteNum=4;
        }
        else if(spriteNum == 4){
            spriteNum=5;
        }
        else if(spriteNum == 5){
            spriteNum=6;
        }
        else if(spriteNum == 6){
            spriteNum=7;
        }
        else if(spriteNum == 7){
            spriteNum=8;
        }
        else if(spriteNum == 8){
            spriteNum=1;
        }
        spriteCounter = 0;
    }
}
```

6. **drawWinPlayer() method:**

- Overall, the drawWinPlayer() method is responsible for selecting and drawing the appropriate images for the winning player and their feathers based on the current game state(how many player played) and the winning player's index.
- The method uses a switch statement based on the value of the end variable. If the end variable is "normal", it executes the corresponding code block. Each time spriteNum is updated, the picture of chicken change and eventually it looks like the chicken and feather is dancing jointly.

```
public void drawWinPlayer(Graphics2D g2, int playerwin){
    BufferedImage blue = null;
    BufferedImage yellow = null;
    BufferedImage white = null;
    BufferedImage red = null;
    BufferedImage bluefeather = null;
    BufferedImage yellowfeather = null;
    BufferedImage whitefeather = null;
    BufferedImage redfeather = null;
    switch (end) {
        case "normal":
            if(spriteNum==1){
                blue = chick1;
                yellow = chick6;
                white = chick11;
                red = chick16;
                bluefeather = featheranimation1[0].setPicture(
                    name[0]);
                yellowfeather = featheranimation1[1].
                    setPicture(name[1]);
                whitefeather = featheranimation1[2].setPicture
                    (name[2]);
                redfeather = featheranimation1[3].setPicture(
                    name[3]);
            }
            if(spriteNum==2){
                blue = chick2;
                yellow = chick7;
                white = chick12;
                red = chick17;
                bluefeather = featheranimation2[0].setPicture(
                    name[0+4]);
                yellowfeather = featheranimation2[1].
                    setPicture(name[1+4]);
                whitefeather = featheranimation2[2].setPicture
                    (name[2+4]);
                redfeather = featheranimation2[3].setPicture(
                    name[3+4]);
            }
            if(spriteNum==3){
                blue = chick3;
                yellow = chick8;
                white = chick13;
                red = chick18;
```

```
37                  bluefeather = featheranimation3[0].setPicture(
                        name[0+8]);
38                  yellowfeather = featheranimation3[1].
                        setPicture(name[1+8]);
39                  whitefeather = featheranimation3[2].setPicture
                        (name[2+8]);
40                  redfeather = featheranimation3[3].setPicture(
                        name[3+8]);
41              }
42              if(spriteNum==4){
43                  blue = chick4;
44                  yellow = chick9;
45                  white = chick14;
46                  red = chick19;
47                  bluefeather = featheranimation4[0].setPicture(
                        name[0+12]);
48                  yellowfeather = featheranimation4[1].
                        setPicture(name[1+12]);
49                  whitefeather = featheranimation4[2].setPicture
                        (name[2+12]);
50                  redfeather = featheranimation4[3].setPicture(
                        name[3+12]);
51              }
52              if(spriteNum==5){
53                  blue = chick5;
54                  yellow = chick10;
55                  white = chick15;
56                  red = chick20;
57                  bluefeather = featheranimation5[0].setPicture(
                        name[0+8]);
58                  yellowfeather = featheranimation5[1].
                        setPicture(name[1+8]);
59                  whitefeather = featheranimation5[2].setPicture
                        (name[2+8]);
60                  redfeather = featheranimation5[3].setPicture(
                        name[3+8]);
61              }
62              if(spriteNum>=6 &&spriteNum<=8){
63                  blue = chick1;
64                  yellow = chick6;
65                  white = chick11;
66                  red = chick16;
67                  bluefeather = featheranimation1[0].setPicture(
                        name[0]);
68                  yellowfeather = featheranimation1[1].
                        setPicture(name[1]);
69                  whitefeather = featheranimation1[2].setPicture
                        (name[2]);
70                  redfeather = featheranimation1[3].setPicture(
                        name[3]);
71              }
72              break;
73          default:
74              break;
```

```
75      }
76      if(playerwin==0){
77          if(gp.howManyPlayer == 2) {
78              g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                    gp.playerWinsize,null);
79              g2.drawImage(yellowfeather,270,260,gp.
                    playerWinsize, gp.playerWinsize,null);
80              g2.drawImage(blue, 270 ,260, gp.playerWinsize, gp.
                    playerWinsize, null);
81          }
82          if(gp.howManyPlayer == 3) {
83              g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                    gp.playerWinsize,null);
84              g2.drawImage(yellowfeather,270,260,gp.
                    playerWinsize, gp.playerWinsize,null);
85              g2.drawImage(whitefeather,270,260,gp.playerWinsize
                    , gp.playerWinsize,null);
86              g2.drawImage(blue, 270 ,260, gp.playerWinsize, gp.
                    playerWinsize, null);
87          }
88          if(gp.howManyPlayer == 4) {
89              g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                    gp.playerWinsize,null);
90              g2.drawImage(yellowfeather,270,260,gp.
                    playerWinsize, gp.playerWinsize,null);
91              g2.drawImage(whitefeather,270,260,gp.playerWinsize
                    , gp.playerWinsize,null);
92              g2.drawImage(redfeather,270,260,gp.playerWinsize,
                    gp.playerWinsize,null);
93              g2.drawImage(blue, 270 ,260, gp.playerWinsize, gp.
                    playerWinsize, null);
94          }
95      }
96      if(playerwin==1){
97          if(gp.howManyPlayer == 2) {
98              g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                    gp.playerWinsize,null);
99              g2.drawImage(yellowfeather,270,260,gp.
                    playerWinsize, gp.playerWinsize,null);
100             g2.drawImage(yellow, 270 ,260, gp.playerWinsize,
                    gp.playerWinsize, null);
101
102         }
103         if(gp.howManyPlayer == 3) {
104             g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                    gp.playerWinsize,null);
105             g2.drawImage(yellowfeather,270,260,gp.
                    playerWinsize, gp.playerWinsize,null);
106             g2.drawImage(whitefeather,270,260,gp.playerWinsize
                    , gp.playerWinsize,null);
107             g2.drawImage(yellow, 270 ,260, gp.playerWinsize,
                    gp.playerWinsize, null);
108
109         }
```

```
110             if(gp.howManyPlayer == 4) {
111                 g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                        gp.playerWinsize,null);
112                 g2.drawImage(yellowfeather,270,260,gp.
                        playerWinsize, gp.playerWinsize,null);
113                 g2.drawImage(whitefeather,270,260,gp.playerWinsize
                        , gp.playerWinsize,null);
114                 g2.drawImage(redfeather,270,260,gp.playerWinsize,
                        gp.playerWinsize,null);
115                 g2.drawImage(yellow, 270 ,260, gp.playerWinsize,
                        gp.playerWinsize, null);
116
117             }
118         }
119         if(playerwin==2){
120
121             if(gp.howManyPlayer == 3) {
122                 g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                        gp.playerWinsize,null);
123                 g2.drawImage(yellowfeather,270,260,gp.
                        playerWinsize, gp.playerWinsize,null);
124                 g2.drawImage(whitefeather,270,260,gp.playerWinsize
                        , gp.playerWinsize,null);
125                 g2.drawImage(white, 270 ,260, gp.playerWinsize, gp
                        .playerWinsize, null);
126
127             }
128             if(gp.howManyPlayer == 4) {
129                 g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                        gp.playerWinsize,null);
130                 g2.drawImage(yellowfeather,270,260,gp.
                        playerWinsize, gp.playerWinsize,null);
131                 g2.drawImage(whitefeather,270,260,gp.playerWinsize
                        , gp.playerWinsize,null);
132                 g2.drawImage(redfeather,270,260,gp.playerWinsize,
                        gp.playerWinsize,null);
133                 g2.drawImage(white, 270 ,260, gp.playerWinsize, gp
                        .playerWinsize, null);
134
135             }
136         }
137         if(playerwin==3){
138             if(gp.howManyPlayer == 4) {
139                 g2.drawImage(bluefeather,270,260,gp.playerWinsize,
                        gp.playerWinsize,null);
140                 g2.drawImage(yellowfeather,270,260,gp.
                        playerWinsize, gp.playerWinsize,null);
141                 g2.drawImage(whitefeather,270,260,gp.playerWinsize
                        , gp.playerWinsize,null);
142                 g2.drawImage(redfeather,270,260,gp.playerWinsize,
                        gp.playerWinsize,null);
143                 g2.drawImage(red, 270 ,260, gp.playerWinsize, gp.
                        playerWinsize, null);
144
```

```
            }
        }
}
```

7. **drawVictory() method:**

   - Overall, the drawVictory() method is responsible for selecting and drawing the appropriate victory graphics based on the current game state and the value of endNum.

```
public void drawVictory(Graphics2D g2) {
    BufferedImage victoryup = null;
    BufferedImage victorydown = null;

    switch (end) {
        case "normal":
            if(endNum==1){
                victoryup = victory_up;
            }
            if(endNum==2){
                victorydown = victory_down;
            }

            break;

        default:
            break;
    }
    g2.drawImage(victoryup, 225 , 5, 355, 270, null);
    g2.drawImage(victorydown, 225 , 5, 355, 270, null);
}
```

Chapter 9
Experimental Results, Statistical Tests and Running Scenarios

9.1 Test

9.1.1 Test 1: Unit test

The test focuses on testing individual functions or components of your code to ensure they work correctly. For a board game, you can write unit tests to check if functions responsible for setting up the game board, moving game pieces, or calculating scores are functioning as expected.

9.1.2 Test 2: Integration test

The test verifies that different components of your game work together correctly. For example, you can write integration tests to check if the game logic interacts correctly with the user interface, if the game state is updated correctly when players perform actions, or if the game correctly handles multiplayer interactions.

9.1.3 Test 3: Functional test

The test evaluates the overall functionality of your board game implementation. They focus on testing the game from a user's perspective, ensuring that it behaves as expected. Functional tests can include scenarios such as starting a new game, making moves, and checking if the game ends correctly.

9.1.4 Test 4: Boundary test

The test examines how your game handles edge cases and boundary conditions. For example, you can test what happens if a player tries to make an invalid move, if the game board becomes full, or if there are multiple players with different winning conditions. These tests help ensure that your game handles unexpected situations gracefully.

9.1.5 Test 5: Performance test

The test assesses the performance of your board game implementation. You can measure factors such as game load time, responsiveness, and the ability to handle a large number of game pieces or players. Performance testing helps identify any bottlenecks or optimizations needed to improve the game's performance.

9.2 Debug

9.2.1 Debug 1: Overtaking Function

The overtaking function is the most important function of our game development. The mistake here is that we forgot to check all of the feathers when an overtaking happens, that means a feather of a chicken can become the feather of another chicken. That why, if we update just the feather of the current chicken for the chicken who overtake this chicken, the game will be wrong.

```
417 -
418 -                   feathers[currentPlayer].setPosition((players[currentPlayer].getPosition()+trackcount+1)%24);
419 -                   drawFeathers[currentPlayer].setPosition((players[currentPlayer].getPosition()+trackcount+1)%24);
441 +                   for(int i=0; i<howManyPlayer; i++)
442 +               {
443 +                   if(feathers[i].getPosition()==players[currentPlayer].getPosition())
444 +                   {
445 +                   feathers[i].setPosition((players[currentPlayer].getPosition()+1+trackcount)%24);
446 +                   drawFeathers[i].setPosition((players[currentPlayer].getPosition()+1+trackcount)%24);
447 +                   }
448 +               }
449 +                   // feathers[currentPlayer].setPosition((players[currentPlayer].getPosition()+trackcount+1)%24);
450 +                   // drawFeathers[currentPlayer].setPosition((players[currentPlayer].getPosition()+trackcount+1)%24);
```

Fig. 9.1 Debug 1: Overtake Function

9.2.2 Debug 2: Performance

When working in the project, many methods and functions with if-else condition are used. When the game play meets the conditions, they still runs and check another condition and do not stop. That is why the performance of the game become worse. For this debug, we try to delete or add functions or return type in order to mitigate the useless function.

9.3 Poster Session and Evaluation

9.3.1 Evaluation Link and Questions

In the poster session, we have created an evaluation form in order to get feedback from our friends and teacher also. The evaluation link is: https://forms.gle/8jCiXZe9FvDW9zJ7A There are 9 questions in the evaluation form, which are:

1. How would you rate the overall game-play experience of the game?
2. Is this game likely to have the same version with the boardgames?
3. Did the game meet your expectations in terms of graphics and visual design?
4. How engaging and challenging did you find the gameplay mechanics?
5. Were the game controls intuitive and easy to understand?
6. How well did the game audio and sound effects enhance your gaming experience?
7. Did you encounter any technical issues or bugs while playing the game? (Yes/No)
8. How likely are you to recommend this game to others?
9. Any additional comments or suggestions for the game development team?

Fig. 9.2 Evaluation link

9.3.2 Results

There are 21 responses in total. The result is visualize in the chart as following. Follow the results, almost people like our game and grade the function with good grade. Also, we receive the message with the flipping function, which can make it better. The detailed result can be seen in the following figures.

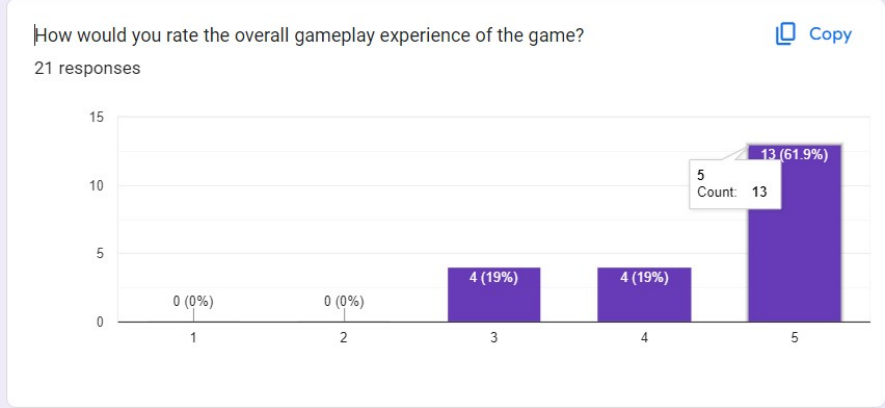

Fig. 9.3 Question 1 Results

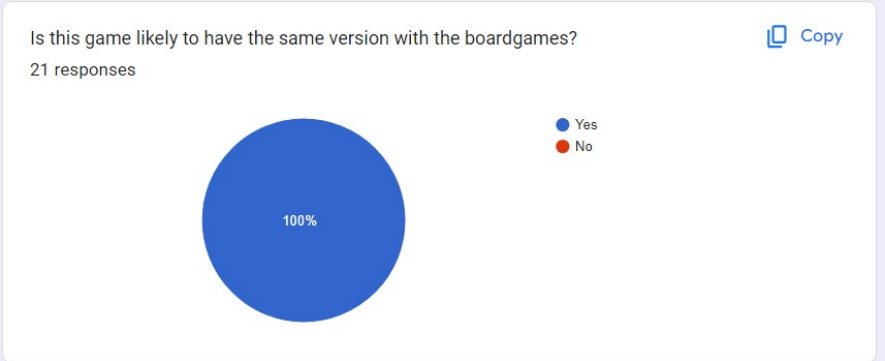

Fig. 9.4 Question 2 Results

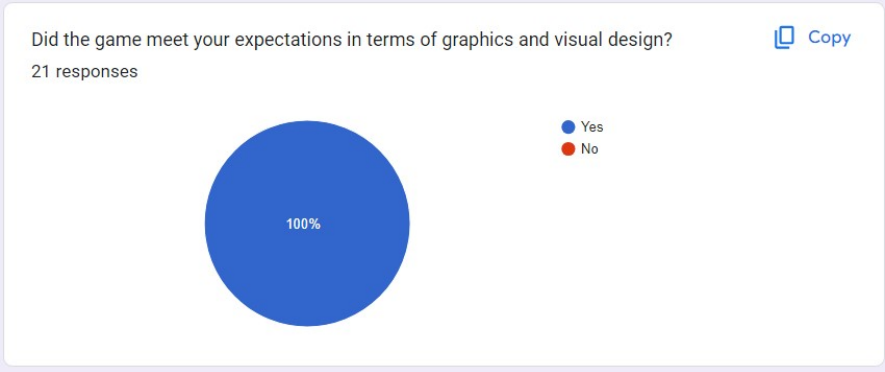

Fig. 9.5 Question 3 Results

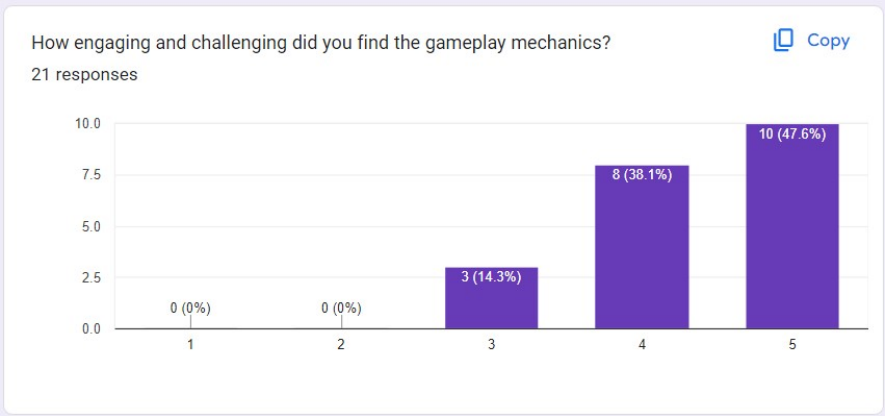

Fig. 9.6 Question 4 Results

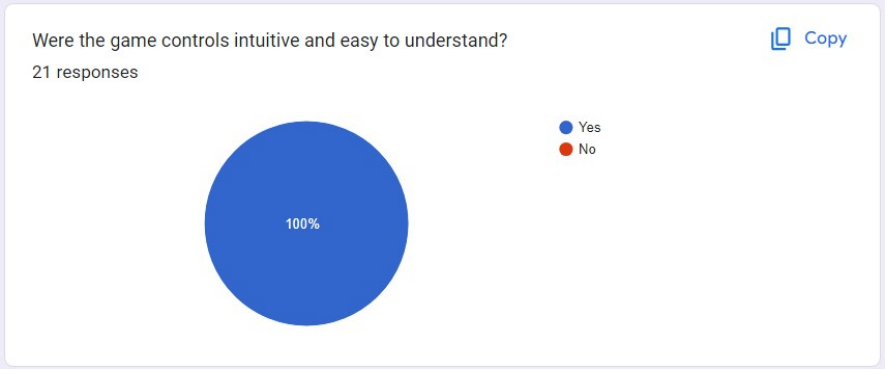

Fig. 9.7 Question 5 Results

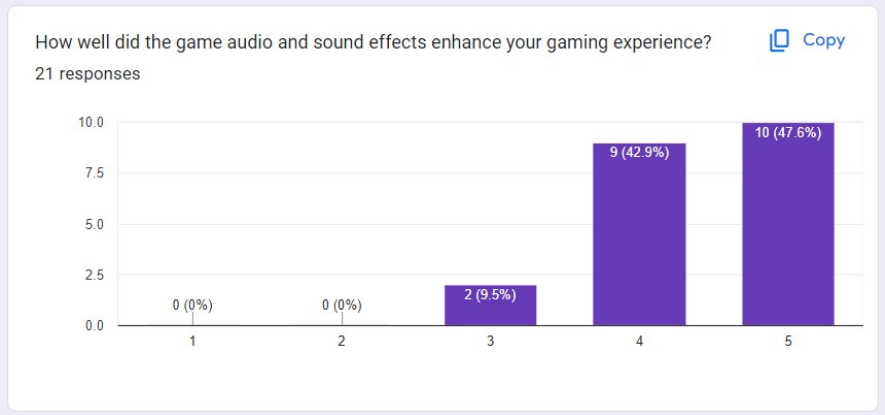

Fig. 9.8 Question 6 Results

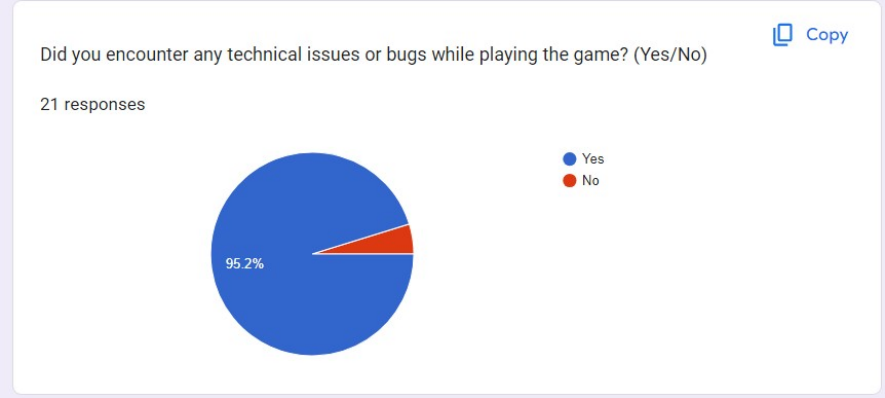

Fig. 9.9 Question 7 Results

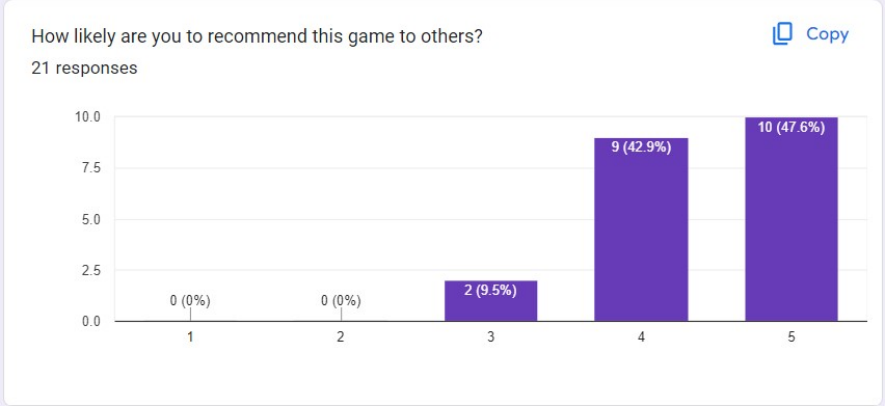

Fig. 9.10 Question 8 Results

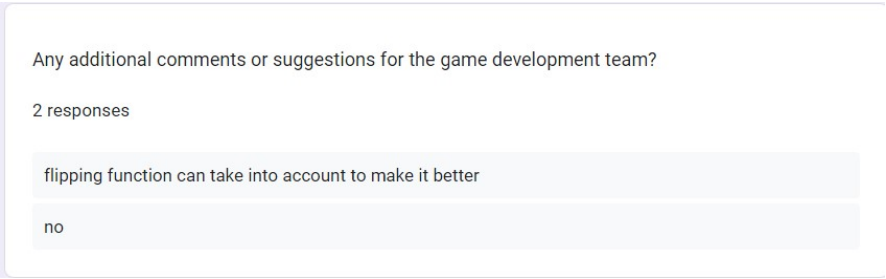

Fig. 9.11 Question 9 Results

Chapter 10
Conclusion and Future Work

10.1 How was the project?

In conclusion, our project of creating the ”Zicke Zacke Hühnerkacke” game in Java using object-oriented programming (OOP) has been a rewarding and enjoyable experience. Throughout the development process, we had a great deal of fun working together as a team and learned valuable lessons in project management and teamwork.

Implementing the game using OOP principles allowed us to structure our code in a modular and organized manner. This approach improved the readability, maintainability, and extensibility of our codebase. We gained a deeper understanding of OOP concepts such as encapsulation, inheritance, and polymorphism, which enhanced our programming skills.
We encountered challenges along the way, such as designing the game logic, implementing the graphical user interface, and ensuring smooth game-play. However, our collaborative efforts and problem-solving skills enabled us to overcome these obstacles and deliver a functional and engaging game.

10.2 What we have learned?

Not only did we acquire technical knowledge and skills during this project, but we also cultivated important soft skills. Effective communication, coordination, and teamwork were vital in ensuring smooth progress and achieving our goals. We learned how to distribute tasks efficiently, collaborate effectively, and resolve conflicts constructively. The following picture is what we added into our poster in the poster session day.

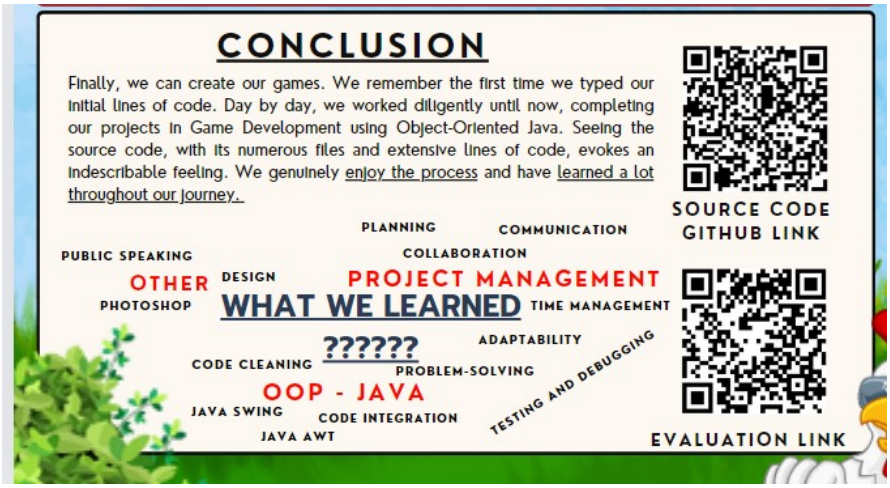

Fig. 10.1 What have we learned?

10.3 Future Work

As for future work, there are several avenues to explore. We can enhance the game by adding additional features, such as different game modes, power-ups, or multiplayer functionality. Improving the user interface and graphics can also contribute to a more visually appealing gaming experience. Additionally, we can conduct user testing and gather feedback to further refine and optimize the game mechanics.

1. Additional Game Modes: Consider expanding the game by introducing different game modes. For example, you could implement a timed mode where players race against the clock to collect the most feathers, or a cooperative mode where players work together to achieve a common goal. Power-Ups and Special Abilities: Introduce power-ups or special abilities that players can acquire during the game. These could provide advantages such as extra moves, the ability to steal feathers from opponents, or temporary invincibility.
2. Multiplayer Functionality: Enhance the game by adding multiplayer functionality, allowing players to compete against each other online. This could be implemented through networked gameplay or by integrating online multiplayer platforms.
3. Improved User Interface: Enhance the visual appeal and usability of the game by improving the user interface. Consider redesigning the game board, refining the graphics and animations, and ensuring intuitive and responsive controls.
4. Sound Effects and Music: Add sound effects and background music to enhance the overall gaming experience. This can create a more immersive atmosphere and add excitement to the gameplay.
5. AI Opponents: Implement computer-controlled opponents with varying levels of difficulty. This would allow players to enjoy the game even when playing alone, providing a challenging and engaging experience.

Chapter 11
References

References

[1] Oracle. *Java Tutorials - Learning the Java Language*. Retrieved from `https://docs.oracle.com/javase/tutorial/java/concepts/`

[2] Sierra, K., & Bates, B. *Head First Java*. O'Reilly Media.

[3] Schildt, H. *Java: A Beginner's Guide*. McGraw-Hill Education.

[4] Bloch, J. *Effective Java*. Addison-Wesley Professional.

[5] Brettspiele-Magazin. (n.d.). *Zicke Zacke Hühnerkacke - Spielregeln und Infos zum Spiel*. In *Brettspiele-Magazin*. Retrieved from https://www.brettspiele-magazin.de/zicke-zacke-huehnerkacke/

[6] RyiSnow. 2022. How to Make a 2D Game in Java . Retrieved from https://www.youtube.com/watch?v=om59cwR7psIlist=PL_QPQmz5C6WUF-pOQDsbsKbaBZqXj4qSq