

2D JAVA GAME: TEMPO, KLEINE FISCHE



Nguyen Minh Tri - 1520110

Vuong Quoc An - 1523083

Tran Viet Trung - 1519264

Content

1	Team Work - Work Structure	4
1.1	Development Team	4
1.2	Communication and Collaboration	4
1.3	Decision-Making	4
1.4	Workflow and Processes	4
1.5	Adaptability and Flexibility	5
1.6	Technology and Tools	5
1.7	Ideas	5
2	Introduction	6
2.1	General Topic	6
2.2	Definitions	6
2.3	Classification	6
2.4	Game Content	7
2.5	Game Play	7
2.6	Examples	8
2.6.1	Color Die	8
2.6.2	Background - Gameboard	8
2.6.3	Fish and Boat	8
2.6.4	Additional Screen:	9
2.6.5	Title Screen:	9
2.6.6	Movements	10
3	Problem Description	12
3.1	Navigating Challenges in Practice	12
3.2	Reliance on Online Resources	12
3.3	Utilizing Piskel for Map Creation	13
3.4	Similar problems in practice	13
4	Related Work	14
5	Proposed Approaches	15
5.1	Input/Output Format:	15
5.1.1	Input:	15
5.1.2	Output:	16
5.1.3	Generation:	16
5.2	Algorithms in Pseudocode	16



6	Implementation Details and Application Structure	18
6.1	GUI Details	18
6.1.1	Swing in Java	18
6.1.2	GUI in Java	18
6.1.3	Code Example	18
6.2	UML Diagram	19
6.3	Used Libraries	22
6.4	Code Snippets	22
6.4.1	Main Class	22
6.4.2	GamePanel Class	24
6.4.3	KeyHandler Class	30
6.4.4	Sounds Class	36
6.4.5	UI Class	39
6.4.6	Player Class	51
6.4.7	TileManager Class	53
6.4.8	EntityHandler Class	56
6.4.9	Boat Class	58
6.4.10	Dice Class	62
6.4.11	Fishes Class	64
7	Experimental Results, Statistical Tests, Running Scenarios	70
7.1	Tables	70
7.2	Charts & Evaluations	70
8	Conclusions and Future Work	72
9	References	73

1 Team Work - Work Structure

1.1 Development Team

- **Vuong Quoc An**
 - Logic of the game
 - Create menu game
 - Sound effects
 - Writing report (15%)
 - Poster design (33.3%)
- **Tran Viet Trung**
 - User Interface
 - End game state
 - Win and lose condition
 - Writing report (15%)
 - Poster design (33.3%)
- **Nguyen Minh Tri**
 - Game state
 - Title Screen
 - Graphic designer
 - Writing report (70%)
 - Poster design (33.3%)

1.2 Communication and Collaboration

Regular team meetings ensure effective communication. Collaborative sessions between designer and coders for seamless integration.

1.3 Decision-Making

Major decisions are made collectively with input from the team.

1.4 Workflow and Processes

Agile development methodology with iterative sprints and feedback loops. Defined workflows for coding and integration.

1.5 Adaptability and Flexibility

Regular feedback sessions allow for adjustments based on project needs.

1.6 Technology and Tools

- **Version Control Systems:** Git for code versioning.
- **Project Management Tools:** Trello for task tracking
- **Brainstorming:** Java-based game development using YouTube tutorials.
- **Visualize Systems and Software:** Magic Draw for plotting diagram.
- **IDE:** IntelliJ IDEA.
- **Collaboration:** Github.
- **Communication Tools:** Google Meet for project planning and discussions.

1.7 Ideas

Basic ideas: "Tempo, Kleine Fische" is an exciting board game specially created for young players, offering thrilling adventures and delightful challenges for children to enjoy. Designed on Piskel, the graphics intricately depict the undersea world. Java Swing ensures a smooth gaming experience. The tempo dice, driven by algorithms, adds a melodic touch to movement precision. The modular board guarantees game variety with computational unpredictability. Algorithmically optimized rules prioritize simplicity for broad accessibility. "Tempo, Kleine Fische" stands as a unique aquatic adventure, showcasing the synergy of scientific design and gaming creativity.

2 Introduction

2.1 General Topic

This topic delves into the fundamentals and advanced techniques of 2D game development using the Java programming language. Covering aspects such as graphics rendering, user input handling, collision detection, and game mechanics, the exploration will encompass both theoretical concepts and practical implementation. By understanding the intricacies of 2D game development, developers can gain valuable insights applicable to a wide range of game projects, fostering a solid foundation for creating engaging and visually appealing gaming experiences.

Exploring the intricacies of 2D game development, this section takes a closer look at the comprehensive design and development journey undertaken for 'Tempo, Kleine Fische,' an engaging underwater adventure. The investigation covers a spectrum of elements, incorporating graphic design tools such as Piskel, execution through Java Swing, and the introduction of innovative features like the tempo dice. Delving into the distinctive attributes of the game, the discussion underscores the harmonious blend of scientific design principles and imaginative gaming innovations. This segment provides a nuanced examination of the decision-making processes, technologies, and creative choices pivotal in realizing the vision of 'Tempo, Kleine Fische.'

2.2 Definitions

A river flows into the big sea in Avanti mare!. At one end of the river, a fishing boat starts moving downstream toward the sea that awaits at the end. Halfway to the sea, four fish are playing in the water – but once that boat starts to move, the fish will flee to the safety of the open water. Will they get away, or end up in a net? Players can decide to back either the fish or the fishermen, but the winners will be determined by the die rolls. On a turn, the player rolls the colored die. If the player rolls one of the two colors assigned to the boat, he removes one of the river segments, thereby moving the boat closer to the fish; if one of the fish colors is rolled, that fish moves one space closer to the sea. If the boat moves onto a fish, it catches that fish; if a fish reaches the sea, it escapes and can't be captured. The game ends when all the fish escape or are caught. Which side will come out on top?

2.3 Classification

The topic of discussion focuses on the development of the game "Tempo, Kleine Fische," with a particular emphasis on its simplicity and design elements. Positioned within the broader area of Game Design and Innovation, the game follows to simple principles, resulting in a user-centered and accessible experience. This simplicity extends beyond game design and into other categories, emphasizing a unique approach. The subject also discusses Practical Learning issues, emphasizing common difficulties in resource selection and reliance on online materials. The exploration's attention to simplicity and design highlights adaptive problem-solving, demonstrating resilience and creativity in practical applications.

2.4 Game Content

An exciting color dice game for 2-6 Fisherman from 3-7years

Content:

2 Fisherman, 4 Fish, 1 Boat, 1 Large part of the river, 11 Small parts of the river, 1 Part of the sea, 1 Color dice

Game Objective:

The color die determines whether the boat or the fish are allowed to move. If there are more fish in the sea at the end, the fish friends win. If the fishermen catch more than two fish, the fishing fans win.

2.5 Game Play

At the beginning you can decide whether you would rather stick with the fishermen or the fish.

Roll the dice and draw

The youngest player starts and rolls the dice. Do you have **Blue**, **Orange**, **Yellow** or **Pink** rolled the dice, you draw it fish of the same color a part of the river further. You roll the dice **Red** or **Green**, you take the small river part directly in front of it boat away and put it aside. You then push the river part with the boat forward until it hits the next river part.

A fish reaches the sea.

If you drag a fish onto the sea part, it is free and can no longer be caught by the boat. If his color is rolled again during the game, you may You can move any other fish one part of the river forward.

A fish was captured.

If you remove a portion of the river from which one or more If there are fish, place them in the boat. They were caught by fishermen.

Attention: As the game progresses, you roll the color of a caught fish, then do the river part away from the boat and move the boat forward.

The end of the game

There are several ways to end the game:

- **All fish reach the sea.** The fish friends win.
- **All fish are caught.** The Fischer fans win.
- **The boat reaches the sea** because the last part of the river was removed. Since it is not seaworthy, the fishermen have to be content with the fish they have caught up to that point. If you stuck with the fish, you win, if more fish could escape into the sea. If you bet on the fishermen, you win, when more fish were caught. If exactly two fish are caught, the game ends in a draw.

2.6 Examples

2.6.1 Color Die



Figure 1: A Color die

2.6.2 Background - Gameboard

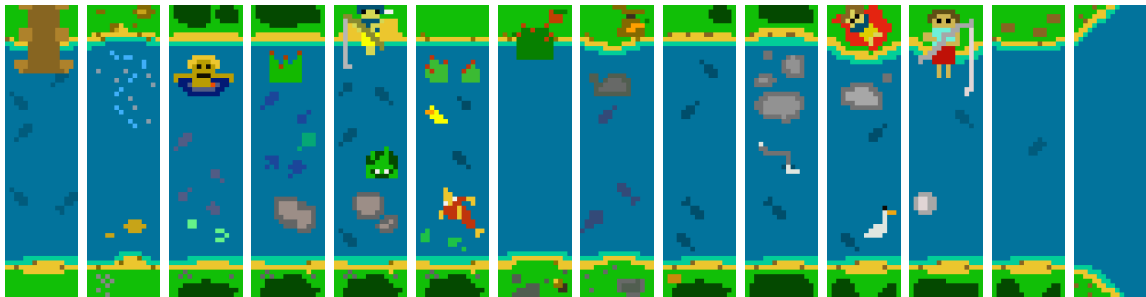


Figure 2: 14 Tiles

2.6.3 Fish and Boat

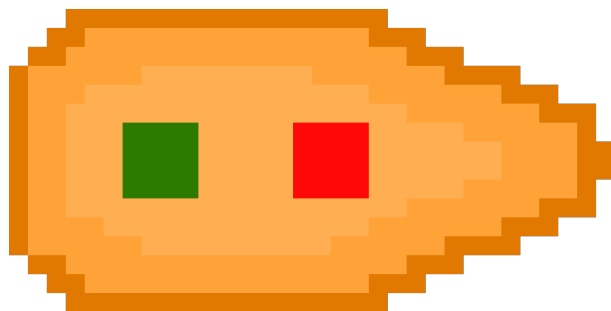


Figure 3: Boat



Figure 4: Fish

2.6.4 Additional Screen:

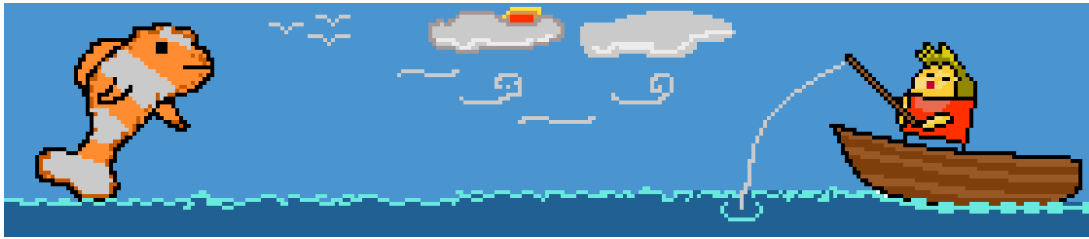


Figure 5: Additional Screen

2.6.5 Title Screen:



Figure 6: Title Screen

2.6.6 Movements

- This is the start of the game, where the boat is placed at position 0 and the fish is placed at position 4.



Figure 7: Start Screen.

- If the color of the die is red or green, the boat moves forward 1 tile, and so on until it reaches the final tile.



Figure 8: Boat movement.

- If the color of the die is yellow, blue, orange, or pink, the fish moves forward 1 tile, and so on until it reaches the final tile.



Figure 9: Fish movement.

- In this case, the 'blue' fish is caught and gets on the boat. As you can see, the remaining fish are 3. The color dice of the caught fish is now added to the boat.



Figure 10: The boat caught the fish.

- And the game continues until it can be defined who wins (fisherman or fishermen).

3 Problem Description

3.1 Navigating Challenges in Practice

The enormous task of choosing appropriate videos looms major in the route toward practical proficiency, frequently posing a significant barrier for learners. A lack of specialized learning tools not only creates a potential bottleneck for skill development, but also leaves practitioners navigating their educational route with ambiguity and unsatisfactory outcomes.

This complex terrain underscores the vital need for the development and refining of comprehensive and user-friendly learning tools. The lack of such tools not only impedes progress, but also emphasizes the need for a nuanced, tailored strategy that accommodates varied learning styles and competence levels. The creation of powerful and accessible learning resources is critical for empowering learners to overcome the multifarious problems inherent in their practical endeavors and building a more inclusive and effective learning environment.

3.2 Reliance on Online Resources

Furthermore, while relying on internet resources to reduce instructional scarcity is a common strategy, it frequently falls short of our expectations. The large amount of available information frequently lacks alignment with our specific needs, worsening rather than reducing the problem. This emphasizes the importance of more targeted and specialized lessons that not only fulfill specific requirements but also provide a more effective and purposeful learning experience.

Dive further into this challenge to see how well-crafted, customized tutorials may impact your code. Their ability to improve the learning journey, meet specific needs, and provide targeted insights demonstrates the enormous impact of tailored educational materials in navigating and eventually overcoming the challenges caused by a scarcity of adequate learning resources.

In the process of discovering viable tutorials, we find that we mostly use resources from a particular playlist on YouTube. The playlist's name is "How to Make a 2D Game in Java" by Ry-iSnow. The playlist is very informative and very suited for our needs, they present a simple start to a quite challenging process for beginner game coders like us. Despite the game we developed is vastly different from the tutorial, it's generalize enough for us to distill crucial information to start our development.

Another resource we often uses in the development process are the website stackoverflow.com. It is a great place to find help when you stumble upon unknown bug and having difficult problems to solve. Despite it being a very useful resource, it's quality is vary from threads to threads, we have to be very cautious when using them, a few problems we encounter are that the question is on an older version of Java, the replies are not applicable to our situation or the problems we encounter are not on the website at all.

3.3 Utilizing Piskel for Map Creation

We strategically went to Piskel for map construction as part of our unwavering dedication to overcoming obstacles. Recognizing the limitations of current tutorials and online resources, the decision to use Piskel proved to be an imaginative and effective alternative. Piskel's instrumental abilities were critical in the diligent creation of accurate and visually appealing maps adapted directly to our demands.

This strategic change not only underlines our adaptability in problem solving, but it also underscores the importance of experimenting with different tools when faced with limits in traditional learning channels. The seamless integration of Piskel into our process demonstrates the resilience and ingenuity inherent in our strategy, demonstrating our capacity to successfully manage hurdles and generate impactful solutions in practical applications.

3.4 Similar problems in practice

Decision for this game to be only one player game:

- In the original board game, it is multiplayer, and each player chooses a fish or a boat. However, when we bring it to the situation when we play it on just a computer, it is unwise to share just a computer with each other, and each player just hits the Space key, and the action of the entities in there is the same for any person that rolls the dice.
- Therefore, we conclude that this game can only be played by one person on the computer, just like many other roll-and-write board games out there. We adjust the game rule a little by instead of choosing which fish, the player can choose to be with the fisherman side or the fisherman side, and based on that, the result can be either you win or lose.

Program for special cases when rolling the dice of the fish that has finished the race:

- In the original game, when we roll the dice of the fish that has finished the race, we can choose which other fish should advance forward. However, after careful consideration, that logic may not be suitable for our game. The main reason is that in the original board game, it was multiplayer, but in our game, it is just one person who rolls the dice all the time. When that person chooses the fisherman side, it will not be so logical that he or she chooses which fish to race further.
- The solution for this is a small algorithm to find the currently slowest fish. Basically, it is just a loop to all the fish and get their positions and compare them to each other.

Multiple message appear at the same time:

- During the coding step, we see that it is more intuitive to have more messages appear at once. At the time, we can only show 1 message, but through the utilization of Java arrays and nights of debugging, we can show up to 4 messages at a time.
- We took a more limited approach to this problem since the scale of the game is only limited to four fish. First, I created an array to store the messages and modified all the dependencies variables accordingly, but that proved to be quite troublesome as I had to add many for loops and if conditions to check what messages were shown. So instead, I created a List object that makes it easier to tell if the elements are in or not with the built-in method.
- After adding the messages and drawing the messages onto the game panel, I wait for all the messages to go out, then remove them all from the list with the help of the built-in method.

Restart new game:

- When a current game has ended, you would like to have another go at this super interesting game. So, you press the new game button, and you realize that the fish are still escaped or that the boat has captured all the fish, and you are still won or lost. It happened that way because the state of the fish and the boat are still the same despite the game changing.
- The solution to this problem is to make a method to restart all attributes of the game. So I created a smaller public method in every relevant entity and then group them and call them the `newGame()` method whenever a new game is initialized.

4 Related Work

Overview

This is our first ever game that was made on a computer, so we do not have any experience to compare this game to others. However, we did play many other board games that had the same roll and write properties as this game, so this section is to compare this game to others. We want to compare this game to another board game called Cubitos.

Comparison between the two games:

- **Rolling properties:**

In this game, Tempo and Kleine Fische have rolling properties, but it is just one dice, and the result is not so diverse, so it is easy to code. Based on the color, the fish or the boat with the same color will move forward. However, in the other game, Cubitos, there are a lot more types of dice with different colors; each color has a different set of dice, and the player can choose the set of dice according to their interest, and each set will affect

the gameplay a little. Therefore, the game is much harder to code and implement on the computer.

- **Multiplayer properties:**

In Tempo, Kleine Fische, we still have multiplayer, but it is not really multiplayer. One player can hold and roll the dice all the time, and the result of the game is still the same as passing the dice to other players. Therefore, we must implement a one-person game roll and write on the computer. However, in Cubitos, this game is truly a multiplayer game. Each player will have their own set of dice, which they all roll simultaneously. Each will have a different result based on the player's decision, which will affect the gameplay and their experience with the game. So in order to implement this game on a computer, we will have to code it to play online because keyboards on one computer may not satisfy the simultaneously rolling properties of this game.

- **Racing properties:**

Both games have racing properties, with each side trying to reach their goal first. In Tempo, Kleine Fische, the goal is to reach the finish line based on the color. In Cubitos, it is based on both the color and the result of that color too. Therefore, the code is much more complicated.

- **Suitable for children:**

Both games are easy and appealing to children. Both have many colors, have random properties, and are replayable many times. The price of the Cubitos board game may be higher, but it will provide some cute animated characters that may be more interesting to a child.

Conclusion:

In conclusion, Tempo, Kleine Fische may not be as complicated to play and to code as other roll-and-write games. For our future development, we may implement Cubitos, as it still has some similarities to Tempo and Kleine Fische.

5 Proposed Approaches

5.1 Input/Output Format:

5.1.1 Input:

- Selecting the appropriate player starting (fish or fishermen).
- The result of the dice roll (color).
- Present condition of the river (locations of fish and boat).

5.1.2 Output:

- Current status of the river following the player's action.
- Information about any fish reaching the sea or being caught.
- The movement of the boat and the fish.
- Update the criteria for winning.

5.1.3 Generation:

- Efficient generation of random dice rolls and their corresponding actions.
- The river state is dynamically generated dependent on the player's movements.

5.2 Algorithms in Pseudocode

Explanation of Pseudocode: Handling Dice Results in Fische Game

1. Dice Timer and Done Check:

- The condition `'(!diceTimer && !done)'` guarantees that the subsequent actions will only be executed if the dice timer is inactive and the game has not been completed.

2. Dice Result Evaluation:

- The code verifies the outcome of the dice roll by accessing the `'dice.result'`.
- If the outcome of the dice is below 4, it additionally assesses if a fish has been caught.
- If a fish is captured (`'fishes.getCaught(dice.result)'` evaluates to true), the boat is activated (`'boat.run'` is set to true).
- If no fish is caught, it verifies whether the fish corresponding to the dice result has completed its task (`'fishes.getFinished(dice.result)'`).
- Following the completion of the program, the fish that has the lowest X-coordinate among the available possibilities is selected, and the dice result is then updated accordingly.
- If the task is incomplete, it assigns the fish to run based on the result of the dice (`'fishes.run = dice.result'`).
- If the outcome of the dice roll is 4 or higher, the boat is configured to operate (`'boat.run = true'`).

Listing 1: Pseudocode of the algorithm

```
1  // Action base on the result
2      if(!diceTimer && !done){
3          if(dice.result <4){
4              //If this fish is caught
5              if(fishes.getCaught(dice.result)){
6                  boat.run = true;
7              }
8              else {
9                  if(fishes.getFinished(dice.result)){
10                     int min = gp.tileWidth*15;
11                     int fishNum = -1;
12                     for(int i=0;i<4;i++){
13                         if(min>fishes.getFishX(i) && !fishes.↵
                           getCaught(i)){
14                             min = fishes.getFishX(i);
15                             fishNum = i;
16                         }
17                     }
18                     //If the fish is finished -> change to the ↵
                           lowest fish
19                     dice.result = fishNum;
20                 }
21
22                 fishes.run = dice.result;
23             }
24         }
25         else{
26             boat.run=true;
27         }
28     }
```

Summary:

- The pseudocode contains the algorithm for managing various situations depending on the outcome of a dice roll in the game.
- It is responsible for determining whether a fish has been caught, whether a fish with a specific result has arrived at its destination, and adjusting the status of the game accordingly.
- The decision-making process entails taking into account factors such as the quantity of fish captured, the quantity of fish processed, and the value derived from the outcome of rolling the dice.

6 Implementation Details and Application Structure

6.1 GUI Details

6.1.1 Swing in Java

Swing in Java is a toolkit for creating Graphical User Interfaces (GUIs) that consists of several GUI components. Swing offers an extensive range of widgets and packages to create advanced GUI components for Java applications. Swing is an integral component of Java Foundation Classes (JFC), an application programming interface (API) for Java that facilitates the creation of graphical user interfaces (GUIs).

The Java Swing library is constructed upon the Java Abstract Widget Toolkit (AWT), which is an older GUI toolkit that is dependent on the platform. You can utilize the Java simple GUI programming components like button, textbox, etc., from the library and do not have to construct the components from scratch.

6.1.2 GUI in Java

Container classes are classes that are capable of containing and managing other components. So for developing a Java Swing GUI, we need at least one container object. Java Swing consists of three distinct types of containers.

1. **Frame:** A window that is completely operational and includes its own title and icons.
2. **Panel:** It serves as a mere receptacle and does not possess the characteristics of a window. A Panel serves the exclusive function of arranging the components onto a window.
3. **Dialog:** It can be conceptualized as a pop-up window that appears when a message has to be shown. The window is not fully operational, unlike the Frame.

6.1.3 Code Example

Listing 2: Java GUI example

```
1 public class Main {
2     public static JFrame window;
3     public static void main(String[] args) {
4         window = new JFrame();
5         window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
6         window.setResizable(false);
7         window.setTitle("Fische");
8         // window.setUndecorated(true);
9
10        GamePanel gamePanel = new GamePanel();
11        window.add(gamePanel);
```

```
12
13     window.pack();
14
15     window.setLocationRelativeTo(null);
16     window.setVisible(true);
17
18     gamePanel.setupGame();
19     gamePanel.startGameThread();
20
21 }
22 }
```

6.2 UML Diagram

Overview:

- *"Tempo, Kleine Fische" is a compelling board game set in a vibrant harbor, where players take on the role of fishermen and compete to catch the most valuable fish. The game's immersive gameplay is supported by a carefully designed architecture, consisting of several classes and components that give the game a realistic feel. This essay will analyze the architectural structure of "Tempo, Kleine Fische," examining its fundamental classes and the complexities of its design.*
- *The game's core functionality is built upon a collection of essential classes. The "Main" class functions as the entry point to the game, overseeing the initialization process and initiating the game loop. It establishes the context for the players' sea voyages, providing the foundation for their thrilling escapade.*
- *While players explore the vast port, the "KeyHandler" class efficiently handles player input. The "KeyHandler" facilitates smooth interaction between players and the game world, whether through strategic maneuvers or decisive actions.*
- *Within the domain of player management, the "Player" class assumes a prominent role, representing the individual participants in the marine competition. At this location, players monitor their advancement, calculate their scores, and plan their next action as they compete for dominance in the sparkling ocean.*
- *Prior to embarking on the journey, participants are given the chance to select their avatar or preferences using the "ChoosePlayer" class. This crucial moment establishes the atmosphere for the upcoming adventure, enabling players to customize their experience and fully engage themselves in the universe of "Tempo, Kleine Fische."*
- *The "Sounds" class enhances the gameplay by managing the audio components in the game, providing an additional layer of sensory enjoyment. The "Sounds" class enhances the game*

by providing ambiance, feedback, and immersion, so increasing the player experience with the soothing sounds of waves and the triumphant yells of victory.

- *In addition to the classes focused on the players, the game also features complex systems that regulate the entities and tiles. The "EntityHandler" class serves as the central control for managing entities, effectively orchestrating the interactive parts of the game. The "EntityHandler" is responsible for managing the production, manipulation, and interactions of many entities, like boats, dice, and fishes, through careful structure and inheritance.*
- *The "TileManager" class holds ultimate authority within the realm of tiles, governing the arrangement and administration of the game world's terrain. The "TileManager", with its meticulous attention to detail and exceptional organizational skills, ensures a seamless and effortless experience for players as they navigate the crowded waterfront.*

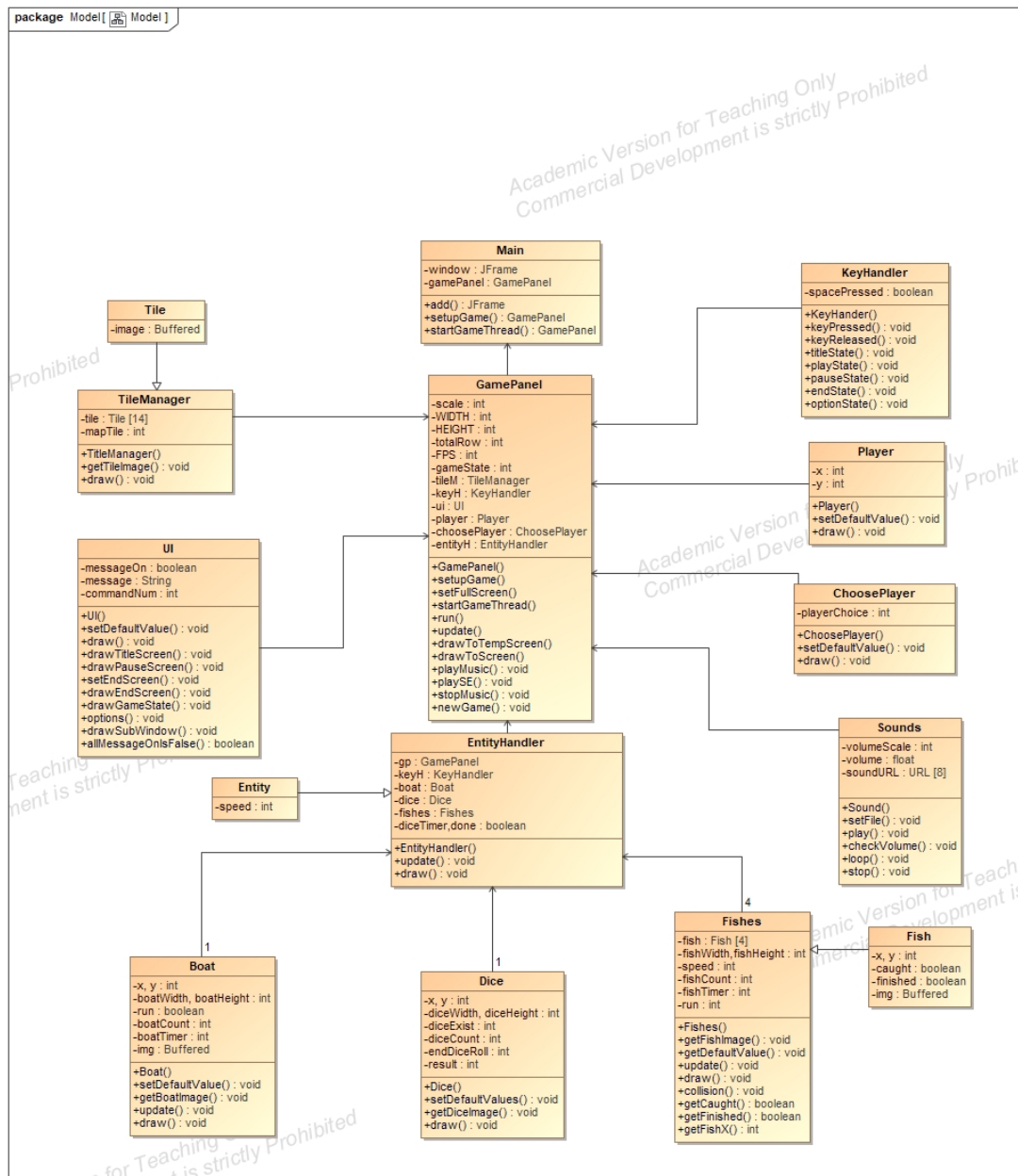


Figure 11: UML Diagram for "Tempo, Kleine Fische" Game

6.3 Used Libraries

1. javax.swing

- **JFrame:** is a class within the Java Swing framework that serves as a high-level container for a graphical user interface (GUI) application. It offers a graphical interface where you may position different user interface elements.
- **Usage:** JFrame is utilized to construct the primary window for our program, enabling the incorporation of buttons, labels, and other components.

2. javax.imageio

- **ImageIO:** is a class in the javax.imageio package that provides methods for reading and writing images in various formats.
- **Usage:** ImageIO is employed to import images into your application, usually in the BufferedImage format, and to export images if necessary.

3. java.awt

- **java.awt** is a package providing fundamental classes for creating graphical user interfaces and performing basic graphics operations.
- **Usage:** The package provides classes for creating graphical elements, performing graphics operations, and interacting with components such as buttons and labels.
- **BufferedImage:** is a class in the **java.awt.image** package that represents an image with an accessible buffer of image data.
- **Usage:** The BufferedImage class is utilized to store images in computer memory. The library offers functions for manipulating and retrieving pixel information, which makes it well-suited for image processing tasks.

4. java.io

- **IOException:** is an exception class in the java.io package that is thrown when an I/O operation fails or is interrupted.
- **Usage:** is employed for managing exceptions pertaining to input/output operations, such as the reading or writing of files.

6.4 Code Snippets

6.4.1 Main Class

- **Introduction:**

In the realm of Java game development, the main function serves as the gateway to initializing and launching a game. Within this function, various essential steps are undertaken to set up the game environment, including creating the graphical user interface (GUI),

initializing game components, and starting the game loop for continuous gameplay. In this essay, we will dissect the main function's role in initializing a game, focusing on a scenario involving a JFrame window and a JPanel.

- **Creating the Game Window:**

The main function begins by creating an instance of a JFrame window, which serves as the main container for the graphical interface of the game. This JFrame serves as the platform for displaying and interacting with the game by the player. During the window generation process, specific criteria such as the title and dimensions of the window are defined, establishing the framework for the visual display of the game.

- **Instantiating the JPanel:**

Following the creation of the JFrame window, the main function proceeds to instantiate a JPanel object. The JPanel renders and manipulates the game's elements, such as characters, objects, and backgrounds, serving as the central component. This panel encapsulates the core functionality of the game, including its logic, rendering pipeline, and event handling mechanisms.

- **Integration of JPanel with JFrame:**

After being created, the JPanel is smoothly included into the JFrame window using the add() method. This phase establishes a hierarchical connection in which the JPanel is positioned as a child component of the JFrame, taking up the entire graphical area of the JFrame. By incorporating the JPanel into the JFrame, the game's visual interface is seamlessly integrated into the window, providing players with an immersive gameplay experience.

- **Initialization of the Game:**

After establishing the graphics framework, the main function proceeds to initiate the game by invoking the setupGame() method. This approach involves a sequence of preliminary operations, including loading resources, configuring game settings, and initializing game state variables. By using the setupGame() method, the game environment is carefully configured, guaranteeing that all necessary components are present for smooth gaming execution.

- **Commencement of the Game Loop:**

The final crucial step undertaken by the main function is the initiation of the game loop through the startGameThread() method. The game loop represents the heart of the game's execution, orchestrating the continuous flow of gameplay by iteratively updating game state, processing user input, and rendering graphical output. By spawning a dedicated thread for the game loop, the main function ensures that gameplay proceeds smoothly without blocking the main application thread, thereby guaranteeing responsiveness and fluidity in player interactions.

Listing 3: Main Class

```
1 public class Main {
2     public static JFrame window;
3     public static void main(String[] args) {
4         window = new JFrame();
5         window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
6         window.setResizable((false));
7         window.setTitle("Fische");
8         // window.setUndecorated(true);
9
10        GamePanel gamePanel = new GamePanel();
11        window.add(gamePanel);
12
13        window.pack();
14
15        window.setLocationRelativeTo(null);
16        window.setVisible(true);
17
18        gamePanel.setupGame();
19        gamePanel.startGameThread();
20
21    }
22 }
```

6.4.2 GamePanel Class

- **Initialization and Configuration:**

The `GamePanel()` constructor acts as the starting point for initializing instances of the `GamePanel` class. This function performs crucial setup chores, including specifying the dimensions of the panel, setting up event listeners, and initializing internal game state variables. The `GamePanel` achieves a streamlined method to initializing game environments by encapsulating these initialization functions within the constructor. This ensures consistency and coherence across different game instances.

- **Preparing the Game Environment:**

The `setupGame()` method is a crucial step in configuring the game environment for gameplay. This function carries out essential initialization activities, such as loading game assets, configuring game mechanisms, and initializing game entities. The `setupGame()` method is responsible for initializing several aspects of the game, such as loading character sprites, implementing collision detection systems, and establishing game rules. Its purpose is to ensure that all necessary components are properly set up before the game starts, thereby assuring a smooth and uninterrupted gameplay experience.

- **Configuring Display Modes:**

The `setFullScreen()` method enables developers to dynamically switch between windowed and full-screen modes by changing the display mode of the game panel. Developers may enhance interoperability and user experience across multiple gaming platforms and devices by changing between display modes to improve the visual presentation of the game for different screen sizes and resolutions.

- **Initiating the Game Loop:**

The game loop is the central component of real-time gaming experiences, as it manages the uninterrupted progression of games. The `startGameThread()` method initiates this crucial step by creating a specialized thread to conduct the game loop separately from the main application thread. The `GamePanel` utilizes multi-threading techniques to enable seamless gameplay, with consistent updates to game state, user input processing, and graphical rendering. This results in a smooth and engaging gaming experience.

- **Executing the Game Loop:**

The `run()` method encapsulates the fundamental concept of the game loop, acting as the main execution thread that is responsible for propelling gameplay ahead. This method employs iterative operations, including updating the game state, processing user input, and producing graphical output, in a continuous loop to maintain a responsive and interactive game. The `GamePanel` effectively maintains the appearance of dynamic gameplay by carefully overseeing these operations, which captivates players and fully engages them in vivid digital environments.

- **Updating Game State:**

The `update()` method is crucial for ensuring the coherence and consistency of the game world. It achieves this by modifying its internal state according to the passage of time and user input. This method is responsible for the movement of game entities, detection of collisions, and processing of game events, which in turn represent the changes occurring in the game world as time progresses. The `update()` method guarantees flawless execution of game logic by synchronizing with the game loop, resulting in interesting and dynamic gameplay experiences.

- **Rendering Graphical Output:**

The `drawToTempScreen()` and `drawToScreen()` functions encapsulate the rendering pipeline, which is responsible for converting the game state into graphical output that is shown to the player. These techniques involve rendering graphical elements, including as sprites, backdrops, and UI components, onto temporary and final buffers, which are then shown on the screen. The `GamePanel` achieves separation of concerns by divorcing rendering from game logic, which in turn promotes modularity and extensibility in graphical presentation.

- **Playing Music:**

The `playMusic()` allows for the seamless integration of background music into the game

environment. By providing the file path of the desired music file, developers can invoke this function to initiate playback of background music, enhancing the ambiance and immersion of the game world.

- **Playing Sound Effects:**

Sound effects play a crucial role in providing feedback and enhancing interactivity within the game. The playSE() function enables developers to trigger the playback of sound effects at specific events or interactions, adding depth and realism to the gaming experience.

- **Stopping the music:**

There are times when it's necessary to halt the playback of background music, either temporarily or permanently. The stopMusic function provides a convenient mechanism for stopping the currently playing background music, giving players control over the auditory atmosphere of the game.

- **Starting New Game:**

For games with structured progression or level-based gameplay, the ability to initiate a new game session is essential. The newGame() function resets the game state and prepares the environment for a fresh game session, allowing players to embark on new adventures with a clean slate.

Listing 4: GamePanel Class

```
1 package main;
2
3 import entity.EntityHandler;
4 import tile.TileManager;
5
6 import javax.swing.*;
7 import java.awt.*;
8 import java.awt.image.BufferedImage;
9
10 public class GamePanel extends JPanel implements Runnable {
11
12     public final int scale = 5;
13     final int originTileWidth = 16;
14     final int originTileHeight = originTileWidth * 4; //64
15     //final int originBigTileWidth = originNormalTileWidth *3; //48
16
17     public final int tileWidth = originTileWidth * scale; // 80
18     public final int tileHeight = originTileHeight * scale; // 320
19
20     public final int totalRow = 14;
21     public final int WIDTH = 14 * tileWidth; //1120
22     public final int HEIGHT = tileHeight + 48*scale; //560
```

```
23     // FOR FULL SCREEN
24     int screenWidth2 = WIDTH;
25     int screenHeight2 = HEIGHT;
26     BufferedImage tempScreen;
27     Graphics2D g2;
28     //FPS
29     int FPS = 60;
30
31     // SYSTEM
32     public Sound music = new Sound();
33     public Sound se = new Sound();
34     TileManager tileM = new TileManager(this);
35     KeyHandler keyH = new KeyHandler(this);
36     public UI ui = new UI(this);
37     Player player = new Player(this);
38     EntityHandler entityH = new EntityHandler(this, keyH);
39     Thread gameThread;
40     // For when we need to pause the game conveniently
41     public int counter = 0;
42     public Font plainFont = new Font("TimesRoman", Font.PLAIN, 32);
43     public Font boldFont = new Font("TimesRoman", Font.BOLD, 32);
44
45     // GAME STATE
46     public int gameState;
47     public final int titleState = 0;
48     public final int playState = 1;
49     public final int pauseState = 2;
50     public final int chooseState = 3;
51     public final int endState = 4;
52     public boolean optionStateOn = false;
53     public GamePanel(){
54         this.setPreferredSize(new Dimension(WIDTH, HEIGHT)); //???
55         this.setBackground(Color.black);
56         this.setDoubleBuffered(true);
57         this.addKeyListener(keyH);
58         this.setFocusable(true);
59     }
60
61     public void setupGame() {
62         gameState = titleState;
63
64         tempScreen = new BufferedImage(WIDTH, HEIGHT, BufferedImage.↵
65             TYPE_INT_ARGB);
66         g2 = (Graphics2D)tempScreen.getGraphics();
```

```
67         playMusic(0);
68
69
70
71     //         setFullScreen();
72     }
73
74     public void setFullScreen() {
75
76         // GET LOCAL SCREEN DEVICE
77         GraphicsEnvironment ge = GraphicsEnvironment.↵
            getLocalGraphicsEnvironment();
78         GraphicsDevice gd = ge.getDefaultScreenDevice();
79         gd.setFullScreenWindow(Main.window);
80
81         // GET FULL SCREEN WIDTH AND HEIGHT
82         screenWidth2 = Main.window.getWidth();
83         screenHeight2 = Main.window.getHeight();
84     }
85     public void startGameThread(){
86         gameThread = new Thread(this);
87         gameThread.start();
88     }
89
90     @Override
91     public void run(){
92         double drawInterval = 1000000000.0/FPS;
93         double nextDrawTime = System.nanoTime() + drawInterval;
94         while(gameThread != null){
95             update();
96
97             drawToTempScreen(); // draw everything to the buffered ↵
                image
98             drawToScreen(); // draw the buffered image to the screen
99
100             //60FPS
101             try{
102                 double remainingTime = nextDrawTime - System.nanoTime↵
                    ();
103                 remainingTime/=1000000;
104
105                 if(remainingTime<0){
106                     remainingTime=0;
107                 }
108
```

```
109         Thread.sleep((long)remainingTime);
110
111         nextDrawTime +=drawInterval;
112
113         } catch (InterruptedException e){
114             e.printStackTrace();
115         }
116     }
117 }
118
119 public void update(){
120     //Now it is only use for when switching titleState and ↵
121     playState
122     if(counter!=0){counter--;}
123
124     if (gameState == playState) {
125         entityH.update();
126     }
127     if (gameState == pauseState) {
128         // Nothing
129     }
130
131     public void drawToTempScreen() {
132         // TITLE SCREEN
133         if (gameState == titleState) {
134             ui.draw(g2);
135         }
136         else
137         {
138             tileM.draw(g2);
139             player.draw(g2);
140             entityH.draw(g2);
141
142             ui.draw(g2);
143         }
144     }
145     public void drawToScreen() {
146         Graphics g = getGraphics();
147         g.drawImage(tempScreen, 0, 0, screenWidth2, screenHeight2, ↵
148             null);
149         g.dispose();
150     }
151     public void playMusic(int i){
152         music.setFile(i);
```

```
152         music.play();
153         music.loop();
154     }
155     public void stopMusic(){
156         music.stop();
157     }
158     public void playSE(int i){
159         se.setFile(i);
160         se.play();
161     }
162     public void newGame(){
163         entityH.getBoat().setDefaultValues();
164         entityH.getFishes().setDefaultValues();
165         ui.restoreDefaultValues();
166     }
167 }
```

6.4.3 KeyHandler Class

- **Setting Up Input Processing:**

The KeyHandler() constructor is essential for initializing instances of the KeyHandler class, providing the basis for input processing systems. This function performs crucial setup activities, such as registering event listeners and initializing internal data structures to monitor the status of keyboard inputs. The KeyHandler class enhances input management by incorporating these initialization functions within its constructor, resulting in a more efficient and dependable technique to capture player actions, thereby ensuring responsiveness and reliability.

- **Interpreting User Input:**

The keyPressed() and keyReleased() methods serve as the main channels for processing and interpreting user input from keyboard events. When a key is pushed or released, these methods are called, giving the KeyHandler class with essential information about the exact keys involved and their accompanying actions. The KeyHandler class utilizes event handling technologies to convert keyboard inputs into game commands, allowing players to navigate characters, perform activities, and interact with the game environment in real-time.

- **Managing Game States:**

Within the domain of game development, effectively handling various phases of gameplay is crucial for coordinating a wide range of player experiences. The KeyHandler class is crucial in this context, as it has methods like titleState(), playState(), pauseState(), endState(), and optionState() that are responsible for managing input events that are particular to various game states. The KeyHandler class guarantees smooth transitions between different

states, promoting continuity and coherence in player interactions, whether it be browsing through menu panels, commanding characters during gaming, or modifying settings in the options menu.

Sample Scenarios:

1. Title State:

The KeyHandler class in the title state actively monitors and detects particular key inputs, such as the Space key, to trigger transitions to other game states, such as the play state or options menu.

2. Play State:

During gameplay, the KeyHandler class interprets key inputs for controlling player characters, executing actions such as movement in response to player commands.

3. Pause State:

When the game is paused, the KeyHandler class captures key inputs for navigating pause menus, adjusting settings, and resuming or exiting the game session.

4. End State:

Upon completion of a game level or session, the KeyHandler class responds to key inputs for navigating end-game screens, displaying scores, and initiating transitions to subsequent levels or game over screens.

5. Option State:

In the options menu, the KeyHandler class facilitates user interaction with settings and preferences, capturing key inputs for adjusting audio, control configurations, and other game parameters.

Listing 5: KeyHandler Class

```
1 package main;
2
3 import entity.EntityHandler;
4
5 import java.awt.event.KeyEvent;
6 import java.awt.event.KeyListener;
7
8 public class KeyHandler implements KeyListener {
9     GamePanel gp;
10    public boolean spacePressed;
11    public boolean enterPressed;
12
13    @Override
14    public void keyTyped(KeyEvent e) {
15
16    }
```

```
17     public KeyHandler(GamePanel gp) {
18         this.gp = gp;
19     }
20     @Override
21     public void keyPressed(KeyEvent e) {
22         int code = e.getKeyCode();
23
24         //OPTION STATE
25         if(gp.optionStateOn){
26             optionState(code);
27         }
28         // TITLE STATE
29         else if (gp.gameState == gp.titleState) {
30             titleState(code);
31         }
32         //CHOOSE PLAYER STATE
33         else if(gp.gameState == gp.chooseState){
34             choosePlayerState(code);
35         }
36         // PLAY STATE
37         else if (gp.gameState == gp.playState) {
38             playState(code);
39         }
40         // PAUSE STATE
41         else if (gp.gameState == gp.pauseState) {
42             pauseState(code);
43         }
44         // END STATE
45         else if (gp.gameState == gp.endState){
46             endState(code);
47         }
48     }
49
50     void titleState(int code){
51         if (code == KeyEvent.VK_W || code == KeyEvent.VK_UP) {
52             gp.ui.commandNum--;
53             if (gp.ui.commandNum < 0) {
54                 gp.ui.commandNum = 2;
55             }
56             gp.playSE(5);
57         }
58         if (code == KeyEvent.VK_S || code == KeyEvent.VK_DOWN) {
59             gp.ui.commandNum++;
60             if (gp.ui.commandNum > 2) {
61                 gp.ui.commandNum = 0;
```



```
62         }
63         gp.playSE(5);
64     }
65     if (code == KeyEvent.VK_SPACE) {
66         if (gp.ui.commandNum == 0) {
67             gp.gameState = gp.chooseState;
68             gp.playSE(1);
69             gp.counter = 3;
70         }
71         if (gp.ui.commandNum == 1) {
72             // add later
73             gp.optionStateOn = true;
74             gp.ui.previousState = gp.titleState;
75         }
76         if (gp.ui.commandNum == 2) {
77             System.exit(0);
78         }
79     }
80 }
81 void choosePlayerState(int code){
82     if (code == KeyEvent.VK_A || code == KeyEvent.VK_LEFT) {
83         gp.ui.playerChoice--;
84         if(gp.ui.playerChoice==0){
85             gp.ui.playerChoice = 2;
86         }
87         gp.playSE(5);
88     }
89     if (code == KeyEvent.VK_D || code == KeyEvent.VK_RIGHT) {
90         gp.ui.playerChoice++;
91         if(gp.ui.playerChoice==3){
92             gp.ui.playerChoice = 1;
93         }
94         gp.playSE(5);
95     }
96     //SELECT THE PLAYER
97     if (code == KeyEvent.VK_SPACE){
98         gp.gameState = gp.playState;
99         gp.ui.commandNum = 0;
100        gp.counter = 3;
101        gp.playSE(1);
102    }
103 }
104 void playState(int code){
105     if (code == KeyEvent.VK_SPACE && gp.counter==0){
106         spacePressed = true;
```

```
107     }
108     if (code == KeyEvent.VK_P){
109         gp.gameState = gp.pauseState;
110         gp.playSE(5);
111     }
112     if (code == KeyEvent.VK_ESCAPE) {
113         gp.ui.previousState = gp.playState;
114         gp.optionStateOn = true;
115         gp.gameState = gp.pauseState;
116         gp.playSE(5);
117     }
118 }
119 void pauseState(int code) {
120     if (code == KeyEvent.VK_P) {
121         gp.gameState = gp.playState;
122         gp.playSE(5);
123     }
124 }
125 void endState(int code){
126     if (code == KeyEvent.VK_W || code == KeyEvent.VK_UP) {
127         gp.ui.commandNum--;
128         if (gp.ui.commandNum < 0) {
129             gp.ui.commandNum = 2;
130         }
131         gp.playSE(5);
132     }
133     if (code == KeyEvent.VK_S || code == KeyEvent.VK_DOWN) {
134         gp.ui.commandNum++;
135         if (gp.ui.commandNum > 2) {
136             gp.ui.commandNum = 0;
137         }
138         gp.playSE(5);
139     }
140     if (code == KeyEvent.VK_SPACE) {
141         if (gp.ui.commandNum == 0) {
142             gp.gameState = gp.chooseState;
143             gp.newGame();
144             gp.counter = 3;
145         }
146         if (gp.ui.commandNum == 1) {
147             gp.gameState = gp.titleState;
148             gp.newGame();
149         }
150         if (gp.ui.commandNum == 2) {
151             System.exit(0);
```

```
152         }
153     }
154 }
155 void optionState(int code){
156     if(code == KeyEvent.VK_ESCAPE){
157         gp.gameState = gp.ui.previousState;
158         gp.optionStateOn = false;
159         gp.ui.commandNum = 0;
160     }
161     if(code == KeyEvent.VK_ENTER || code == KeyEvent.VK_SPACE){
162         spacePressed = true;
163     }
164     if (code == KeyEvent.VK_W || code == KeyEvent.VK_UP) {
165         gp.ui.commandNum--;
166         if (gp.ui.commandNum < 0) {
167             gp.ui.commandNum = 3;
168         }
169         gp.playSE(5);
170     }
171     if (code == KeyEvent.VK_S || code == KeyEvent.VK_DOWN) {
172         gp.ui.commandNum++;
173         if (gp.ui.commandNum > 3) {
174             gp.ui.commandNum = 0;
175         }
176         gp.playSE(5);
177     }
178     if (code == KeyEvent.VK_D || code == KeyEvent.VK_RIGHT) {
179         if(gp.ui.commandNum == 0 && gp.music.volumeScale < 5){
180             gp.music.volumeScale++;
181             gp.music.checkVolume();
182             gp.playSE(5);
183         }
184         if(gp.ui.commandNum == 1 && gp.se.volumeScale < 5){
185             gp.se.volumeScale++;
186             gp.playSE(5);
187         }
188     }
189     if (code == KeyEvent.VK_A || code == KeyEvent.VK_LEFT) {
190         if(gp.ui.commandNum == 0 && gp.music.volumeScale > 0){
191             gp.music.volumeScale--;
192             gp.music.checkVolume();
193             gp.playSE(5);
194         }
195         if(gp.ui.commandNum == 1 && gp.se.volumeScale > 0){
196             gp.se.volumeScale--;
```

```
197         gp.playSE(5);
198     }
199 }
200 }
201
202 @Override
203 public void keyReleased(KeyEvent e) {
204     int code = e.getKeyCode();
205
206     if (code == KeyEvent.VK_SPACE) {
207         spacePressed = false;
208     }
209 }
210 }
```

6.4.4 Sounds Class

- **Setting Up Sound Objects:**

The core functionality of the Sounds class is its capability to create instances of sound objects, which individually represent different audio assets like sound effects or background music. The `Sound()` constructor performs crucial setup chores, such as loading audio files, specifying playback parameters, and initializing internal data structures for sound management. The Sounds class ensures consistency and dependability in sound playing by enclosing initialization routines within the constructor, providing a streamlined approach to audio setup.

- **Configuring Sound Files:**

The `setFile()` method functions as a means to configure the audio file linked to a certain sound object. Using this approach, developers have the ability to indicate the precise file path or resource location of the required audio asset, allowing for the dynamic loading and playback of sound effects and music tracks. The `setFile()` method improves flexibility and modularity in audio asset management by separating sound file setup from sound object initialization. This allows for easy integration of various audio resources into the game environment.

- **Starting Sound Playback:**

The `play()` method initiates sound playback by causing the selected sound object to emit audio output through the system's speakers or headphones. When called, this method instructs the sound object to begin playback from the beginning of the audio file, providing auditory feedback to the player in real time. The `play()` method uses event-driven techniques to provide synchronized playback of sound effects during critical gameplay moments, which improves immersion and feedback in player interactions.

- **Monitoring Audio Levels:**

The `checkVolume()` method allows you to monitor and alter the volume levels of sound objects dynamically. By interrogating the current volume settings of sound objects, developers can determine the strength of audio output and customize it to player preferences or gameplay requirements. Whether altering volume levels in reaction to in-game events or offering player-controlled volume settings in the options menu, the `checkVolume()` method ensures ideal audio balance and immersion in the gaming experience.

- **Enabling Continuous Playback:**

In scenarios where continuous audio playback is desired, the `loop()` method comes into play, enabling seamless looping of sound objects to create ambient soundscapes or background music tracks. By instructing the sound object to loop indefinitely or for a specified number of iterations, developers can create immersive audio environments that captivate players and enrich the atmosphere of virtual worlds. Whether simulating the chirping of birds in a forest or the rumble of machinery in a futuristic cityscape, the `loop()` method imbues games with depth and immersion through continuous audio playback.

- **Stopping the Sound Playback:**

When it is necessary to discontinue sound playback abruptly, the `stop()` function offers a quick and decisive technique for silencing sound objects. The `stop()` method allows developers to seamlessly manage sound cues and transitions by providing a command that stops playback and resets the playback position to the beginning of the audio file. Whether transitioning between game states, ending sound effects after a predetermined length, or pausing background music during menu navigation, the `stop()` method ensures accuracy and control in sound management, improving the coherence and polish of the gaming experience.

Listing 6: Sounds Class

```
1 package main;
2
3 import javax.sound.sampled.AudioInputStream;
4 import javax.sound.sampled.AudioSystem;
5 import javax.sound.sampled.Clip;
6 import javax.sound.sampled.FloatControl;
7 import java.net.URL;
8
9 public class Sound {
10     Clip clip;
11     URL soundURL[] = new URL[9];
12     FloatControl fc;
13     int volumeScale = 3;
14     float volume;
```

```
15
16     public Sound(){
17         soundURL[0] = getClass().getResource("/Sound/SonicFrontiers.↵
           wav");
18         soundURL[1] = getClass().getResource("/Sound/↵
           CollectItemSoundEffect.wav");
19         soundURL[2] = getClass().getResource("/Sound/FishSplashWater.↵
           wav");
20         soundURL[3] = getClass().getResource("/Sound/↵
           WinningSoundEffect.wav");
21         soundURL[4] = getClass().getResource("/Sound/DisappointedSE.↵
           wav");
22         soundURL[5] = getClass().getResource("/Sound/cursor.wav");
23         soundURL[6] = getClass().getResource("/Sound/Dice.wav");
24         soundURL[7] = getClass().getResource("/Sound/steamWhistle.wav"↵
           );
25         soundURL[8] = getClass().getResource("/Sound/Ohno.wav");
26     }
27     public void setFile(int i){
28         try{
29             AudioInputStream ais = AudioSystem.getAudioInputStream(↵
               soundURL[i]);
30             clip = AudioSystem.getClip();
31             clip.open(ais);
32             fc = (FloatControl) clip.getControl(FloatControl.Type.↵
               MASTER_GAIN);
33             checkVolume();
34         }catch (Exception e){
35         }
36     }
37     public void play(){
38         clip.start();
39     }
40     public void checkVolume(){
41         switch (volumeScale){
42             case 0:
43                 volume = -80f;
44                 break;
45             case 1:
46                 volume = -20f;
47                 break;
48             case 2:
49                 volume = -12f;
50                 break;
51             case 3:
```

```
52         volume = -5f;
53         break;
54     case 4:
55         volume = 1f;
56         break;
57     case 5:
58         volume = 6f;
59         break;
60     }
61     fc.setValue(volume);
62 }
63 public void loop(){
64     clip.loop(Clip.LOOP_CONTINUOUSLY);
65 }
66 public void stop(){
67     clip.stop();
68 }
69 }
```

6.4.5 UI Class

- **Setting up parts of the User Interface:** The main feature of the UI class is its capability to initialize instances of user interface components, which serves as the basis for constructing and interacting with graphical user interfaces. The `UI()` constructor performs necessary setup operations, such as specifying display settings, loading graphical assets, and initializing internal data structures for managing UI elements. The UI class achieves a streamlined approach to UI setup by enclosing these initialization routines within the constructor. This ensures consistency and coherence in interface design.

- **Setting Default Values for UI Elements:**

The `setDefaultValue()` method functions as a means of establishing default values and states for UI elements, promoting uniformity and predictability in user engagements. The `setDefaultValue()` function is used to provide a baseline for user interactions in GUI design. It can be used to set default text for input fields, initialize default options for dropdown menus, or pre-set default settings for checkboxes. This method enhances usability and accessibility in GUI design.

- **Rendering User Interface Elements:**

The `draw()` function serves as the fundamental rendering process for presenting user interface items on the screen. This technique involves rendering graphical elements and text onto the graphical canvas, resulting in visually captivating and interactive interfaces for players to interact with. The `draw()` method guarantees the precise and visually appealing presentation of UI components such as dies, labels, panels, and others. This enhances

engagement and immersion in the gaming experience.

- **Designing Title Screens:**

During the early stages of gaming, the `drawTitleScreen()` method is utilized to coordinate the creation and arrangement of title screens that welcome players when they start the game. Using this technique, developers may create visually stunning title screens that establish the mood and ambiance of the gaming experience. These panels showcase artwork, logos, and menu choices that attract players and encourage them to begin their adventure.

- **Creating Pause Screens:**

The `drawPauseScreen()` method facilitates the generation of pause screens during gaming, which offer users the ability to pause the game, modify settings, or continue gameplay. Using this approach, developers may create pause screens that superimpose on the game world, showcasing menus, buttons, and user interface elements that enable players to smoothly navigate pause menus without disrupting the gameplay experience.

- **Concluding Game Sessions:**

The `setEndScreen()` method prepares the user interface (UI) to present end screens that summarize player achievements, display scores, or prompt replay after game sessions. The `drawEndScreen()` method is responsible for rendering end screens that graphically memorialize players' journeys and bring closure to their game experience.

- **Rendering Game States:**

The draw refers to the outcome of a competition or game in which no side emerges as the winner. The `GameState()` method is responsible for presenting the various states of the game during gameplay. It offers players real-time updates on the progress of the game, status indications, and other information. The `drawGameState()` method ensures that players are educated and engaged with the evolving dynamics of the game world through the presentation of music and sound effects.

- **Configuring Game Settings:**

In the options menu, the `options()` method provides players with the ability to customize game settings and preferences to suit their preferences. Through this method, developers can design options menus that offer a plethora of configuration options, including audio settings, graphics options, control configurations, and other game parameters.

- **Creating Sub-Windows for Additional Information:**

The `drawSubWindow()` method allows the construction of sub-windows that appear on top of the main game interface. These sub-windows provide players with extra information, context, tooltips, or dialogues as needed.

- **Managing Message Visibility:**

In scenarios where messages or notifications need to be displayed selectively, the `allMessageOnIsFalse()` method provides functionality for managing message visibility based on

predefined conditions. By toggling message visibility on or off dynamically, developers can control the timing and context of message displays, ensuring that important information reaches players at opportune moments without overwhelming the interface.

Listing 7: UI Class

```
1 package main;
2
3 import javax.imageio.ImageIO;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6 import java.io.IOException;
7 import java.text.DecimalFormat;
8 import java.util.ArrayList;
9 import java.util.Arrays;
10
11 public class UI {
12
13     GamePanel gp;
14     Graphics2D g2;
15     Font arial_40, arial_80B, arial_20;
16     BufferedImage menuImage;
17
18
19     public boolean[] messageOn = {false, false, false, false};
20     ArrayList<String> messages = new ArrayList<>();
21
22     int[] messageCounter = {0, 0, 0, 0};
23     public boolean gameFinished = false;
24     public int commandNum = 0;
25     //USE FOR OPTION
26     public int previousState = -1;
27
28     public boolean fishWin = false;
29     public boolean boatWin = false;
30     public boolean tie = false;
31
32     double playTime;
33     DecimalFormat dFormat = new DecimalFormat("#0.00");
34     public int playerChoice = 1;
35     public int fishChoice = 1;
36     public int fishermanChoice = 2;
37     BufferedImage fisherman, fishes;
38
39     public UI(GamePanel gp) {
```

```
40         this.gp = gp;
41         setDefaultValue();
42     }
43
44     //For restarting the game
45     public void restoreDefaultValue(){
46         gameFinished = false;
47
48         fishWin = false;
49         boatWin = false;
50         tie = false;
51
52         Arrays.fill(messageCounter, 0);
53         Arrays.fill(messageOn, false);
54     }
55
56     void setDefaultValue(){
57         try {
58             arial_40 = new Font("Arial", Font.PLAIN, 40);
59             arial_80B = new Font("Arial", Font.BOLD, 80);
60             arial_20 = new Font("Arial", Font.PLAIN, 20);
61
62
63             menuImage = ImageIO.read(getClass().getResourceAsStream("/↵
64                 Menu/sea3.png"));
65         } catch (IOException e) {
66             throw new RuntimeException(e);
67         }
68
69         try {
70             fisherman = ImageIO.read(getClass().getResourceAsStream("/↵
71                 Menu/fisherman.png"));
72             fishes = ImageIO.read(getClass().getResourceAsStream("/↵
73                 Menu/fishtitle1.png"));
74         } catch (IOException e) {
75             throw new RuntimeException(e);
76         }
77
78         public void showMessage(String text) {
79             messages.add(text);
80             int messagesSize = messages.size() - 1;
81             if (messagesSize > 3) messagesSize = 3;
82             messageOn[messagesSize] = true;
```

```
82     }
83
84     public void draw(Graphics2D g2) {
85
86         this.g2 = g2;
87
88         g2.setFont(arial_40);
89         g2.setColor(Color.white);
90         //TITLE STATE
91         if (gp.gameState == gp.titleState) {
92             drawTitleScreen();
93         }
94         // CHOOSE PLAYER STATE
95         if( gp.gameState == gp.chooseState){
96             drawChoosePlayer();
97         }
98         // PLAY STATE
99         if (gp.gameState == gp.playState) {
100             drawGameState();
101         }
102
103         // PAUSE STATE
104         if (gp.gameState == gp.pauseState) {
105             drawPauseScreen();
106         }
107
108         // END GAME STATE
109         if (gp.gameState == gp.endState){
110             setEndScreen();
111             drawEndScreen();
112         }
113         if(gp.optionStateOn){
114             drawOptionScreen();
115         }
116     }
117
118     public void drawTitleScreen() {
119         g2.drawImage(menuImage,0,0,gp.WIDTH,gp.HEIGHT,null);
120
121         // TITLE NAME;
122         g2.setFont(g2.getFont().deriveFont(Font.BOLD,90F));
123         String text = "Tempo, Kleine Fische";
124         int x = getXforCenteredText(text);
125         int y = gp.HEIGHT-400;
126         // SHADOW COLOR
```

```
127         g2.setColor(Color.black);
128         g2.drawString(text,x+5,y+5);
129
130
131         // MAIN COLOR
132         g2.setColor(Color.white);
133         g2.drawString(text, x, y);
134
135         // MENU
136         int gap = 40;
137
138         g2.setFont(g2.getFont().deriveFont(Font.BOLD,20F));
139         text = "New Game";
140         x = getXforCenteredText(text);
141         y += gp.HEIGHT-375;
142
143         if (commandNum == 0) {
144             g2.setColor(Color.YELLOW);
145             g2.drawString(">", x-30, y);
146         } else {
147             g2.setColor(Color.white);
148         }
149         g2.drawString(text, x, y);
150
151         text = "Options";
152         x = getXforCenteredText(text);
153         y += gap;
154         if (commandNum == 1) {
155             g2.setColor(Color.YELLOW);
156             g2.drawString(">", x-30, y);
157         } else {
158             g2.setColor(Color.white);
159         }
160         g2.drawString(text, x, y);
161         text = "Quit";
162         x = getXforCenteredText(text);
163         y += gap;
164         if (commandNum == 2) {
165             g2.setColor(Color.YELLOW);
166             g2.drawString(">", x-30, y);
167         } else {
168             g2.setColor(Color.white);
169         }
170         g2.drawString(text, x, y);
171     }
```

```
172     public void drawChoosePlayer(){
173         //Choose Player
174         int shadowGap = 8;
175
176         int gap = gp.tileWidth*5;
177
178         int playerSide = 3*gp.tileWidth;
179         int y = gp.HEIGHT/2-playerSide/2;
180         int x = 48* gp.scale;
181
182         Color c = new Color(0,0,0,150);
183         g2.setColor(c);
184         g2.fillRect(0,0,gp.WIDTH,gp.HEIGHT);
185
186         g2.setColor(Color.black);
187         if (playerChoice == fishChoice) {
188             g2.fillRoundRect(x-shadowGap,y-shadowGap,playerSide+2*←
                shadowGap,playerSide+2*shadowGap,35,35);
189         }
190         else{
191             g2.fillRoundRect(x+gap-shadowGap,y-shadowGap,playerSide+2*←
                shadowGap,playerSide+2*shadowGap,35,35);
192         }
193         g2.setColor(Color.blue);
194         g2.fillRoundRect(x,y,playerSide,playerSide,35,35);
195         g2.fillRoundRect(x+gap,y,playerSide,playerSide,35,35);
196
197         g2.drawImage(fishes,x,y,playerSide,playerSide,null);
198         g2.drawImage(fisherman,x+gap,y,playerSide,playerSide,null);
199     }
200
201     public void drawPauseScreen() {
202
203         g2.setFont(g2.getFont().deriveFont(Font.PLAIN,60F));
204         String text = "PAUSED";
205         int x = getXforCenteredText(text);
206
207         int y = gp.HEIGHT/2;
208
209         g2.drawString(text, x, y);
210     }
211
212     public void setEndScreen(){
213         g2.setColor(new Color(0, 0, 0, 0.5f)); // 50% darker (change ←
            to 0.25f for 25% darker)
```

```
214         g2.fillRect(0, 0, gp.WIDTH, gp.HEIGHT);
215         g2.setFont(arial_80B);
216         g2.setColor(Color.white);
217         String text = "Game end!";
218
219         if (boatWin)
220             if(playerChoice == fishermanChoice){
221                 text = "YOU WIN!!!";
222             }
223             else{
224                 text = "YOU LOSE!!!";
225             }
226         else if (fishWin) {
227             if(playerChoice == fishChoice){
228                 text = "YOU WIN!!!";
229             }
230             else{
231                 text = "YOU LOSE!!!";
232             }
233         }
234         else if (tie){
235             text = "The game is tie ._.";
236         }
237
238         int x;
239         x = getXforCenteredText(text);
240         g2.drawString(text, x, 200);
241     }
242
243     public void drawEndScreen() {
244
245
246         // MENU
247         int gap = 40;
248         g2.setColor(Color.white);
249
250         g2.setFont(g2.getFont().deriveFont(Font.BOLD,20F));
251         String text = "New Game";
252         int x = getXforCenteredText(text);
253         int y = gp.HEIGHT-300;
254         if (commandNum == 0) {
255             g2.setColor(Color.YELLOW);
256             g2.drawString(">", x-30, y);
257         } else {
258             g2.setColor(Color.white);
```

```
259         }
260         g2.drawString(text, x, y);
261
262
263         text = "Back to Title";
264         x = getXforCenteredText(text);
265         y += gap;
266         if (commandNum == 1) {
267             g2.setColor(Color.YELLOW);
268             g2.drawString(">", x-30, y);
269         } else {
270             g2.setColor(Color.white);
271         }
272         g2.drawString(text, x, y);
273
274         text = "Quit";
275         x = getXforCenteredText(text);
276         y += gap;
277         if (commandNum == 2) {
278             g2.setColor(Color.YELLOW);
279             g2.drawString(">", x-30, y);
280         } else {
281             g2.setColor(Color.white);
282         }
283         g2.drawString(text, x, y);
284     }
285
286     public void drawOptionScreen(){
287         g2.setColor(Color.WHITE);
288
289         //SUB WINDOW
290         int frameWidth = gp.tileWidth*8;
291         int frameHeight = gp.HEIGHT-gp.tileWidth ;
292         int frameX = gp.WIDTH/2-frameWidth/2;
293         int frameY = gp.HEIGHT/2-frameHeight/2;
294         drawSubWindow(frameX,frameY,frameWidth,frameHeight);
295
296         options(frameX,frameY);
297
298         gp.keyH.enterPressed = false;
299     }
300
301     public void drawGameState(){
302         if (gameFinished == true){
303
```

```
304         gp.gameState = gp.endState;
305     }
306     else {
307         g2.setFont(arial_20);
308         g2.setColor(Color.white);
309         g2.drawString("Fishes remaining = " + gp.entityH.getFishes()
310             .getFishRemaining(), 30, 335);
311         for(int messageIndex = 0; messageIndex < 4; messageIndex++) {
312             if (messageOn[messageIndex] && messages.size() > messageIndex) {
313                 g2.drawString(messages.get(messageIndex), 800, 335 + messageIndex * 30);
314
315                 messageCounter[messageIndex]++;
316
317                 if (messageCounter[messageIndex] > 130) {
318                     messageCounter[messageIndex] = 0;
319                     messageOn[messageIndex] = false;
320                     for(int i = 0; i < 4; i++){
321                         System.out.print(messageOn[i] + ", ");
322                     }
323                     System.out.println();
324                     if (allMessageOnIsFalse()){
325                         System.out.println("Size of messages before remove: " + messages.size());
326                         messages.removeAll(messages);
327                         System.out.println("Size of messages after remove: " + messages.size());
328                     }
329                 }
330             }
331         }
332     }
333
334     public void options(int frameX,int frameY){
335         int textX;
336         int textY;
337         int gap = 14*gp.scale;
338         g2.setFont(gp.boldFont);
339
340         //TITLE
341         String text = "Options";
342         textX = getXforCenteredText(text);
```



```
343         textY = frameY + gap;
344         g2.drawString(text, textX, textY);
345         g2.setFont(gp.plainFont);
346
347         //MUSIC
348         textX = frameX + gp.tileWidth;
349         textY += gap;
350         if(commandNum == 0){
351             g2.setFont(gp.boldFont);
352             g2.drawString(">", textX-25, textY);
353         }
354         g2.drawString("Music", textX, textY);
355         g2.setFont(gp.plainFont);
356
357         //SE
358         textY += gap;
359         if(commandNum == 1){
360             g2.setFont(gp.boldFont);
361             g2.drawString(">", textX-25, textY);
362         }
363         g2.drawString("SE", textX, textY);
364         g2.setFont(gp.plainFont);
365
366         //END GAME
367         textY += gap;
368         if(commandNum == 2){
369             g2.setFont(gp.boldFont);
370             g2.drawString(">", textX-25, textY);
371             if(gp.keyH.spacePressed == true){
372                 if(previousState == gp.playState){
373                     commandNum = 0;
374                     gp.gameState = gp.titleState;
375                     gp.newGame();
376                     gp.stopMusic();
377                     gp.playMusic(0);
378                     gp.keyH.spacePressed = false;
379                     gp.optionStateOn = false;
380                 }
381                 else{
382                     System.exit(0);
383                 }
384             }
385         }
386     }
387     if(previousState == gp.playState){
```

```
388         text = "End Game";
389     }
390     else{
391         text = "Exit Game";
392     }
393     g2.drawString(text,textX,textY);
394     g2.setFont(gp.plainFont);
395
396     //BACK
397     textX = getXforCenteredText("Back");
398     textY += 2*gap;
399     if(commandNum == 3){
400         g2.setFont(gp.boldFont);
401         g2.drawString(">",textX-25,textY);
402         if(gp.keyH.spacePressed == true){
403             commandNum = 0;
404             gp.gameState = previousState;
405             gp.optionStateOn = false;
406             gp.keyH.spacePressed = false;
407             gp.playSE(5);
408         }
409     }
410     g2.drawString("Back",textX,textY);
411     g2.setFont(gp.plainFont);
412
413     //MUSIC
414     textX = frameX + gp.tileWidth*5;
415     textY = frameY + 2*gap-24;
416     g2.drawRect(textX,textY,120,24);
417     // 120 divide by 5 section
418     int volumeWidth = 24 * gp.music.volumeScale;
419     g2.fillRect(textX,textY,volumeWidth,24);
420
421     //SE
422     textY += gap;
423     g2.drawRect(textX,textY,120,24);
424     volumeWidth = 24 * gp.se.volumeScale;
425     g2.fillRect(textX,textY,volumeWidth,24);
426 }
427
428 public void drawSubWindow(int x, int y,int width, int height){
429     Color c = new Color(0,0,0,210);
430     g2.setColor(c);
431     g2.fillRoundRect(x,y,width,height,35,35);
432 }
```

```
433         c = new Color(255,255,255);
434         g2.setColor(c);
435         g2.setStroke(new BasicStroke(5));
436         g2.drawRoundRect(x+5,y+5,width-10,height-10,25,25);
437
438     }
439
440     public int getXforCenteredText(String text) {
441         int length = (int)g2.getFontMetrics().getStringBounds(text, g2↵
            ).getWidth();
442         return gp.WIDTH/2 - length/2;
443     }
444
445     private boolean allMessageOnIsFalse(){
446         for(int i = 0; i < 4; i++){
447             if (messageOn[i])
448                 return false;
449         }
450         return true;
451     }
452 }
```

6.4.6 Player Class

- **Initializing Player Attributes:**

The constructor of the Player class is essential for initializing instances of player characters in the game environment. This function executes crucial setup chores. The Player class achieves a simplified approach to instantiating players by enclosing these initialization functions within the constructor. This ensures that all player instances are consistent and coherent.

- **Configuring Default Player Settings:**

The `setDefaultValue()` method allows developers to define default values for player properties, guaranteeing consistency and predictability in player behavior. This method establishes the fundamental aspects of player interactions and enables smooth integration of player characters into the game world by determining default location, movement speed, and other features. By utilizing this strategy, developers have the ability to establish a fundamental set of player attributes, which serves as an initial point for additional customization and improvement.

- **Rendering Player Graphics:**

Central to the visual representation of player characters is the `draw()` method, responsible for rendering player graphics onto the game canvas. Through this method, graphical

assets such as sprites, animations, and visual effects are drawn at the specified position and orientation, bringing player characters to life within the game environment. By leveraging graphics libraries and rendering techniques, the `draw()` method ensures that player characters are presented with fidelity and flair, enhancing player immersion and visual appeal.

Listing 8: Player Class

```
1 package main;
2
3 import javax.imageio.ImageIO;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6 import java.io.IOException;
7
8 public class Player {
9     GamePanel gp;
10    int x;
11    int y;
12    BufferedImage img;
13    public Player(GamePanel gp){
14        this.gp = gp;
15        setDefaultValue();
16    }
17
18    void setDefaultValue(){
19        x = 0;
20        y = gp.tileHeight;
21        try {
22            img = ImageIO.read(getClass().getResourceAsStream("/Menu/↔
                maptit2.png"));
23        } catch (IOException e) {
24            throw new RuntimeException(e);
25        }
26    }
27
28    void draw(Graphics2D g2){
29        g2.drawImage(img,x,y,gp.WIDTH,48* gp.scale,null);
30    }
31 }
```

6.4.7 TileManager Class

- **Initializing Tile-based Environments:**

The TileManager class is centered around its constructor, which is responsible for initializing instances of tile-based settings within the game world. This method performs crucial setup activities, such as loading tile images, specifying tile characteristics, and populating tile maps with appropriate data. The TileManager class simplifies the process of setting up tile-based environments by incorporating these initialization methods into the constructor. This ensures that environmental design is consistent and coherent.

- **Accessing Tile Images:**

The getImage() method is crucial for displaying tile-based environments. It allows for easy access to specific tile pictures using their unique identifiers or coordinates. Using this approach, developers can access the visual depiction of particular tiles from the tileset, allowing for dynamic rendering and interaction with the game world. The getImage() method improves modularity and flexibility in environmental design by offering a user-friendly interface for getting tile images. This empowers developers to construct visually impressive and diverse game worlds.

- **Rendering Tiles onto the Game Canvas:**

The draw() function encapsulates the fundamental process of drawing tiles, converting tile data into visual output that is shown on the game canvas. This approach involves the traversal of tile maps, rendering each tile in its corresponding place within the game world. The draw() method utilizes graphics libraries and rendering techniques to accurately and stylishly depict tile-based settings, creating an immersive experience for players in visually stunning and dynamic landscapes. The draw() method is crucial in molding the visual aesthetics of the game world and improving player immersion, whether it involves drawing terrain, obstacles, or ornamental objects.

Listing 9: TileManager Class

```
1 package tile;
2
3 import main.GamePanel;
4
5 import javax.imageio.ImageIO;
6 import java.awt.*;
7 import java.io.IOException;
8 import java.io.InputStream;
9
10 public class TileManager {
11     GamePanel gp;
12     public Tile[] tile;
13     public int mapTile[];
```

```
14
15     public TileManager(GamePanel gp){
16         this.gp = gp;
17
18         tile = new Tile[14];
19         mapTile = new int[gp.totalRow];
20         getTileImage();
21     }
22
23     public void getTileImage(){
24         try{
25             tile[0] = new Tile();
26             tile[0].image = ImageIO.read(getClass().←
                getResourceAsStream("/Tiles/map0.png"));
27
28             tile[1] = new Tile();
29             tile[1].image = ImageIO.read(getClass().←
                getResourceAsStream("/Tiles/map1.png"));
30
31             tile[2] = new Tile();
32             tile[2].image = ImageIO.read(getClass().←
                getResourceAsStream("/Tiles/map2.png"));
33
34             tile[3] = new Tile();
35             tile[3].image = ImageIO.read(getClass().←
                getResourceAsStream("/Tiles/map3.png"));
36
37             tile[4] = new Tile();
38             tile[4].image = ImageIO.read(getClass().←
                getResourceAsStream("/Tiles/map4.png"));
39
40             tile[5] = new Tile();
41             tile[5].image = ImageIO.read(getClass().←
                getResourceAsStream("/Tiles/map5.png"));
42
43             tile[6] = new Tile();
44             tile[6].image = ImageIO.read(getClass().←
                getResourceAsStream("/Tiles/map6.png"));
45
46             tile[7] = new Tile();
47             tile[7].image = ImageIO.read(getClass().←
                getResourceAsStream("/Tiles/map7.png"));
48
49             tile[8] = new Tile();
50             tile[8].image = ImageIO.read(getClass().←
```

```

        getResourceAsStream("/Tiles/map8.png"));
51
52     tile[9] = new Tile();
53     tile[9].image = ImageIO.read(getClass().←
        getResourceAsStream("/Tiles/map9.png"));
54
55     tile[10] = new Tile();
56     tile[10].image = ImageIO.read(getClass().←
        getResourceAsStream("/Tiles/map10.png"));
57
58     tile[11] = new Tile();
59     tile[11].image = ImageIO.read(getClass().←
        getResourceAsStream("/Tiles/map11.png"));
60
61     tile[12] = new Tile();
62     tile[12].image = ImageIO.read(getClass().←
        getResourceAsStream("/Tiles/map12.png"));
63
64     tile[13] = new Tile();
65     tile[13].image = ImageIO.read(getClass().←
        getResourceAsStream("/Tiles/map13.png"));
66
67     }catch (IOException e){
68         e.printStackTrace();
69     }
70 }
71
72 public void draw(Graphics2D g2){
73
74     int col = 0;
75     int x = 0,y = 0;
76     while(col < gp.totalRow){
77
78         g2.drawImage(tile[col].image,x,y,gp.tileWidth,gp.←
            tileHeight,null);
79
80         col++;
81         x += gp.tileWidth;
82     }
83
84 }
85 }
```

6.4.8 EntityHandler Class

- **Initializing Entity Management:**

The constructor of the EntityHandler class is the core component responsible for initializing instances of entity management within the game environment. This function executes crucial setup activities, such as constructing data structures to hold entities, specifying entity properties, and establishing rules for entity interactions. The EntityHandler class offers a simpler method to handling entities by encapsulating initialization routines within the constructor. This ensures speed and coherence in entity administration.

- **Managing Entity State Updates:**

The update() method encapsulates the core functionality of entity management, which involves modifying the states and actions of entities according to the passage of time and game events. This method involves traversing entities and updating their states based on predetermined rules and reasoning. The update() method is responsible for simulating movement, handling collisions, and processing entity interactions. It guarantees that entities evolve dynamically over time, which adds to the emergent complexity and richness of gameplay experiences.

- **Displaying Objects on the Game Canvas:**

The draw() method is essential for visually representing dynamic things by projecting them onto the game canvas. This method involves traversing entities and rendering their graphical representations at their corresponding points in the game world. The draw() method utilizes graphics libraries and rendering techniques to ensure that entities are shown with accuracy and style, hence boosting player immersion and visual attractiveness. The draw() method is essential for animating characters, objects, and special effects, thereby creating a vivid gaming environment that captivates players' imaginations.

Listing 10: EntityHandler Class

```
1 package entity;
2
3 import main.GamePanel;
4 import main.KeyHandler;
5
6 import java.awt.*;
7
8 public class EntityHandler extends Entity{
9     GamePanel gp;
10    KeyHandler keyH;
11    Boat boat;
12    Dice dice;
13
14    Fishes fishes;
```



```
15     public boolean diceTimer = false;
16     public boolean done = true;
17
18     public EntityHandler(GamePanel gp, KeyHandler keyH){
19         this.gp = gp;
20         this.keyH = keyH;
21         boat = new Boat(gp, this);
22         dice = new Dice(gp, this);
23         fishes = new Fishes(gp, this);
24     }
25     public void update() {
26         // Activate the dice
27         if(keyH.spacePressed && done){
28             diceTimer=true;
29             gp.playSE(6);
30             done = false;
31         }
32         // Action base on the result
33         if(!diceTimer && !done){
34             if(dice.result <4){
35                 //If this fish is caught
36                 if(fishes.getCaught(dice.result)){
37                     boat.run = true;
38                 }
39                 else {
40                     if(fishes.getFinished(dice.result)){
41                         int min = gp.tileWidth*15;
42                         int fishNum = -1;
43                         for(int i=0;i<4;i++){
44                             if(min>fishes.getFishX(i) && !fishes.←
45                                 getCaught(i)){
46                                     min = fishes.getFishX(i);
47                                     fishNum = i;
48                                 }
49                             }
50                             //If the fish is finished -> change to the ←
51                             lowest fish
52                             dice.result = fishNum;
53                         }
54                     }
55                     fishes.run = dice.result;
56                 }
57             }
58             else{
59                 boat.run=true;
```

```
58         }
59     }
60     if(fishes.update()){
61         System.out.println("play SE");
62         fishes.playFishSE();
63     }
64     boat.update();
65 }
66
67 public void draw(Graphics2D g2){
68     fishes.draw(g2);
69     boat.draw(g2);
70     //Check if Boat catch any
71     if(fishes.collision(boat.x+boat.boatWidth)){
72         //Play SE
73         boat.playBoatSE();
74     }
75
76     if (diceTimer) {
77         dice.draw(g2);
78     }
79 }
80
81 public Fishes getFishes() {
82     return fishes;
83 }
84 public Boat getBoat() {return boat;}
85 }
```

6.4.9 Boat Class

- **Charting the Course of Adventure:**

The constructor of the Boat class acts as the guiding force for creating instances of nautical adventure within the game world. This method involves performing crucial setup activities, such as adjusting the boat's characteristics, determining its course, and raising the sails for a virtual journey across the seas. The Boat class offers a simpler method to setting sail by encapsulating these startup routines within the constructor. This ensures consistency and coherence throughout maritime adventures.

- **Anchoring Attributes for Smooth Sailing:**

The setDefaultValue() method establishes a stable foundation by assigning default values to the boat's properties, thus ensuring a consistent trajectory even in tumultuous conditions. This method establishes the basis for nautical adventures by determining the boat's speed, maneuverability, and durability, serving as the foundation for the boat's journey. By

utilizing this technology, developers may guide the boat towards predictable conditions, creating captivating and immersive sailing experiences for gamers.

- **Unveiling the Exquisite Visual Display:**

The `getBoatImage()` method plays a crucial role in visually representing the boat. It showcases the magnificent sails on the gaming canvas. Using this approach, developers can obtain the visual depiction of the boat, displaying its grand outline against the background of blue seas. By utilizing this strategy, the Boat class converts ordinary pixels into a vehicle of excitement, captivating the imagination of players and propelling them into a limitless realm of possibilities.

- **Navigating the Dynamic Seascape:**

The `update()` method exemplifies the essence of marine navigation, guiding the boat through the ever-changing seascape of games. During this process, the boat's path is adjusted by taking into account user input, resulting in a sailing experience that is both interactive and engaging. The `update()` method allows players to navigate through challenging conditions, such as sailing against the wind, riding waves, or avoiding dangerous areas. It enables players to determine their own path and go on exciting adventures and discoveries.

- **Rendering the Boat's Odyssey:**

The `draw()` method is essential in portraying maritime experiences visually. It skillfully displays the boat's journey on the game canvas with creative expertise. This method vividly portrays the boat's journey, capturing the essence of maritime exploration with meticulous attention to detail. The `draw()` approach enhances the boat's journey with a feeling of awe and thrill. It entices players to go on a unique and extraordinary maritime trip.

Listing 11: Boat Class

```
1 package entity;
2
3 import main.GamePanel;
4 import main.KeyHandler;
5
6 import javax.imageio.ImageIO;
7 import java.awt.*;
8 import java.awt.image.BufferedImage;
9 import java.io.IOException;
10
11 public class Boat extends Entity{
12     GamePanel gp;
13     EntityHandler entityH;
14
15     public int x;
```

```
16     int y;
17     int boatWidth ;
18     int boatHeight ;
19     boolean playSE = false;
20     boolean moveSE = false;
21     public boolean run =false;
22     //Set Boat Timer
23     int boatCount = 0;
24     int boatTimer = 20;
25
26     BufferedImage img;
27
28     public Boat(GamePanel gp,EntityHandler entityH){
29         this.gp = gp;
30         this.entityH = entityH;
31         getBoatImage();
32         setDefaultValues();
33     }
34
35     public void setDefaultValues(){
36         //32*16
37         boatWidth = 32 * gp.scale;
38         boatHeight = 16 * gp.scale;
39         x = 0;
40         y = gp.tileHeight/2 - boatHeight/2;
41         speed = 4;
42     }
43
44     public void getBoatImage(){
45         try{
46             img = ImageIO.read(getClass().getResourceAsStream("/↵
Objects/boat.png"));
47
48         } catch (IOException e){
49             e.printStackTrace();
50         }
51     }
52
53     public void update(){
54         if(run){
55             x += speed;
56             boatCount ++;
57             playBoatMoveSE();
58         }
59         if(boatCount >=boatTimer){
```

```
60
61         run=false;
62         entityH.done =true;
63         boatCount = 0;
64         //Boat win
65         if (entityH.fishes.fishRemaining < 2) {
66             gp.ui.gameFinished = true;
67             gp.ui.boatWin = true;
68             if(gp.ui.playerChoice == gp.ui.fishermanChoice){
69                 gp.playSE(3);
70             }
71             else{
72                 gp.playSE(4);
73             }
74         }//Game Tie
75         else if (entityH.fishes.getFishRemaining() == 2 && entityH.fishes.getFishFinished() == 2){
76             gp.ui.gameFinished = true;
77             gp.ui.tie = true;
78             gp.playSE(8);
79         }//Boat catches some fishes
80         else if(playSE){
81             gp.playSE(1);
82             playSE = false;
83         } else if (moveSE) {
84             gp.playSE(7);
85             moveSE = false;
86         }
87     }
88 }
89
90 public void playBoatSE(){
91     playSE = true;
92 }
93 public void playBoatMoveSE(){ moveSE = true;}
94
95 public void draw(Graphics2D g2){
96     g2.drawImage(img,x,y,boatWidth,boatHeight,null);
97 }
98 }
```

6.4.10 Dice Class

- **Rolling the Die:**

The constructor is the most important part of the Dice class. It sets up each instance of rolling dice in the game setting. By using this method, important setup tasks are completed, such as setting up the dice's properties and faces and getting ready for the roll of the dice. By putting these initialization methods inside the constructor, the Dice class makes rolling the dice smooth and consistent, which is important for games that are based on luck.

- **Setting the Stage for Randomness:**

Setting the default values for the dice's properties makes sure that the roll is fair and random. This is what the `setDefaultValue()` method does. Whether setting the number of faces, the seed for creating random numbers, or other parameters, this way makes it possible for chance to be the main character. By using this method, game designers can add the thrill of uncertainty to their works, encouraging players to enjoy the randomness of dice rolls and the thrill of chance meetings.

- **Unveiling the Face of Fate:**

The `getDiceImage()` function plays a crucial role in displaying the visual representation of the dice. It reveals the outcome of the dice roll on the game canvas. Using this approach, developers can obtain the visual depiction of the dice's present side, displaying the result of the roll in a visually striking manner. The Dice class utilizes this strategy to convert numerical probabilities into visual cues, hence intensifying user immersion and involvement in chance-based gameplay dynamics.

- **Rendering the Roll of Fortune:**

The `draw()` function functions as the canvas on which the roll of fortune is displayed, capturing the suspense and anticipation of each dice roll. This method utilizes creative flare to visually represent the outcome of the dice, bringing each face of the dice to life with vivid realism. The `draw()` method adds narrative meaning to each dice roll, whether it results in a key success, a tight escape, or a stroke of tragedy. This enhances random encounters and transforms them into memorable moments in gaming.

Listing 12: Dice Class

```
1 package entity;
2
3 import main.GamePanel;
4
5 import javax.imageio.ImageIO;
6 import java.awt.*;
7 import java.awt.image.BufferedImage;
```

```
8 import java.io.IOException;
9
10 public class Dice extends Entity{
11     GamePanel gp;
12     EntityHandler entityH;
13
14     int x;
15     int y;
16     int diceWidth ;
17     int diceHeight ;
18     //Time for the Dice to exist
19     int diceExist = 150;
20     int diceCount = 0;
21     //Time to end Rolling for player to see
22     int endDiceRoll = 100;
23     public int result=5;
24
25     BufferedImage[] img = new BufferedImage[6];
26
27
28
29     public Dice(GamePanel gp,EntityHandler entityH){
30         this.gp = gp;
31         this.entityH = entityH;
32         getDiceImage();
33         setDefaultValues();
34         //32*16
35         diceWidth = 48 * gp.scale;
36         diceHeight = 48 * gp.scale;
37     }
38
39     public void setDefaultValues(){
40         x = gp.WIDTH/2-24*gp.scale;
41         //y = gp.tileHeight/2-24* gp.scale;
42         y = gp.tileHeight;
43         speed = 4;
44     }
45
46     public void getDiceImage(){
47         try{
48             img[0] = ImageIO.read(getClass().getResourceAsStream("/↵
49                 Dices/blue.png"));
50             img[1] = ImageIO.read(getClass().getResourceAsStream("/↵
51                 Dices/pink.png"));
52             img[2] = ImageIO.read(getClass().getResourceAsStream("/↵
```

```
        Dices/yellow.png"));
51         img[3] = ImageIO.read(getClass().getResourceAsStream("/↵
        Dices/orange.png"));
52         img[4] = ImageIO.read(getClass().getResourceAsStream("/↵
        Dices/red.png"));
53         img[5] = ImageIO.read(getClass().getResourceAsStream("/↵
        Dices/green.png"));
54
55     } catch (IOException e){
56         e.printStackTrace();
57     }
58 }
59
60
61 public void draw(Graphics2D g2){
62     if(diceCount % 10==0 && diceCount<endDiceRoll){
63         result = (int)(Math.random()*6);
64     }
65
66     g2.drawImage(img[result],x,y,diceWidth,diceHeight,null);
67
68     //Count the dice
69     diceCount++;
70     if(diceCount>diceExist){
71         entityH.diceTimer=false;
72         diceCount=0;
73     }
74 }
75 }
```

6.4.11 Fishes Class

- **Schooling in the Sea:**

The constructor of the Fishes class is the fundamental component responsible for initializing instances of aquatic life within the game environment. This method involves doing crucial setup chores, such as establishing the characteristics of fish, determining their behaviors, and populating the ocean with a wide variety of aquatic life. The Fishes class enhances the creation of dynamic underwater ecosystems by incorporating these initialization algorithms within the constructor. This technique guarantees a more efficient and immersive experience in aquatic gameplay scenarios, while also ensuring a high level of realism.

- **A Glimpse of Underwater Beauty:**

The `getFishImage()` method is crucial for visually representing aquatic life. It provides a view of the undersea beauty on the game canvas. Using this approach, developers may

access the visual depiction of each individual fish species, highlighting its vivid hues and elegant motions. By utilizing this function, the Fishes class converts ordinary pixels into a captivating aquatic scene, enticing users to delve into the depths of the ocean and uncover its concealed marvels.

- **Setting the Stage for Aquatic Adventures:**

The `getDefaultValue()` method establishes the foundation for aquatic expeditions, establishing default values for fish characteristics to provide a balanced underwater habitat. This strategy establishes the foundation for dynamic interactions between fish species and the aquatic environment, encompassing swim speed, maneuverability, and spawning patterns. By utilizing this function, programmers have the ability to generate captivating underwater environments abundant with diverse organisms and dynamic events, promoting user involvement and curiosity inside the expansive oceanic realm.

- **Navigating the Ocean Currents:**

The `update()` method encapsulates the fundamental principles of underwater navigation, directing fish through the ocean currents and ever-changing aquatic surroundings. This approach recalibrates fish behaviors by taking into account surrounding inputs, player interactions, and internal emotions, resulting in an underwater experience that is both responsive and immersive. The `update()` function enables fish to travel the ocean depths with elegance and agility, enhancing the complexity and realism of aquatic gameplay scenarios during activities such as schooling, foraging, and dodging predators.

- **Bringing Underwater Scenes to Life:**

The `draw()` function plays a crucial role in depicting underwater experiences by infusing aquatic sceneries with vibrant detail and creative finesse. Using this technique, fish elegantly navigate the game canvas, their motions harmonized with the oscillation of ocean currents. The `draw()` method enhances aquatic sceneries with a feeling of amazement and curiosity, engaging players and stimulating their imagination as they navigate through coral reefs, glide past swaying seaweed, and explore enigmatic underwater passageways.

- **Navigating Obstacles and Predators:**

The `collision()` method functions as an acoustic detection system in the aquatic environment, identifying barriers, predators, and other potential dangers present beneath the water's surface. This technology allows for the dynamic adjustment of fish behaviors in reaction to collisions with environmental objects or interactions with other entities.

- **Reeling in the Big Catch:**

The `getCaught()` method signifies the pinnacle of underwater fishing expeditions, indicating the triumphant acquisition of a coveted fish by the player. During this procedure, fish are recognized and marked as caught when they come into contact with the player's fishing line or lure. By utilizing this function, developers have the ability to replicate the excitement of capturing something.

- **Completing the Aquatic Quest:**

The `getFinished()` method signifies the achievement of aquatic quests and challenges, indicating the fulfillment of particular objectives or tasks associated with fish interactions. This system enables the tracking of player progress and the rewarding of successful completion of underwater activities, such as collecting rare species, fulfilling fish-related quests, or accomplishing milestones in aquatic tournaments.

- **Mapping the Coordinates of Aquatic Life:**

The `getFinished()` method signifies the achievement of aquatic quests and challenges, indicating the fulfillment of particular objectives or tasks associated with fish interactions. This system enables the tracking of player progress and the rewarding of successful completion of underwater activities.

Listing 13: Fishes Class

```
1 package entity;
2
3 import main.GamePanel;
4
5 import javax.imageio.ImageIO;
6 import java.awt.*;
7 import java.io.IOException;
8
9 public class Fishes{
10     GamePanel gp;
11     Fish[] fish;
12     EntityHandler entityH;
13
14     int firstX;
15     int gap;
16
17     int fishWidth,fishHeight;
18     int speed;
19     // Timer for fish to run
20     int fishCount = 0;
21     int fishTimer = 20;
22     public int run = -1;
23
24     String fishColor[] = {"blue", "pink", "yellow", "orange"};
25
26     int fishRemaining = 4;
27     int fishFinished = 0;
28     //Determined when to play SE
29     boolean playSE = false;
30 }
```

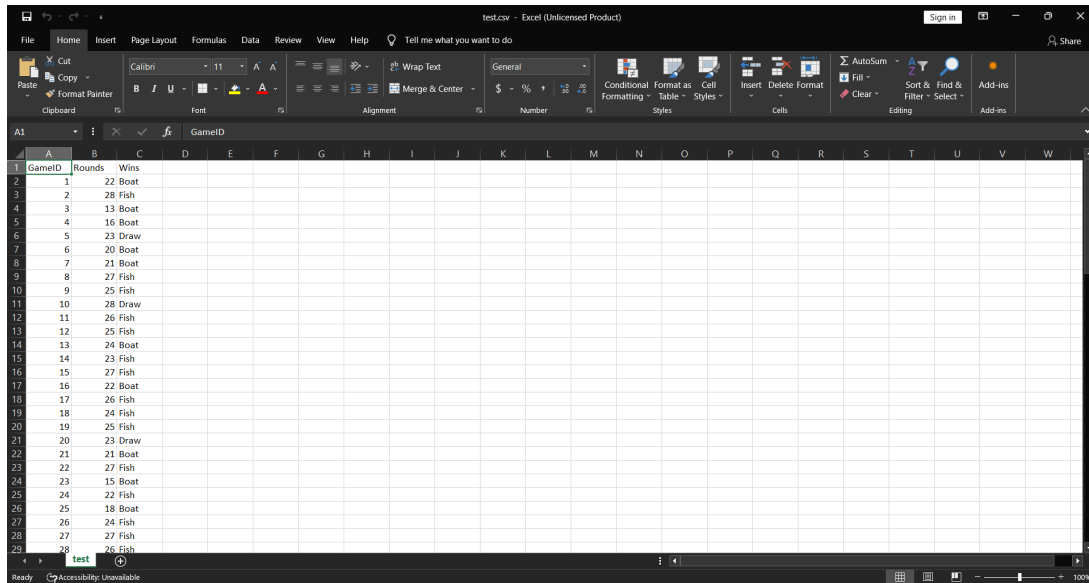
```
31     public Fishes(GamePanel gp,EntityHandler entityH){
32         this.gp = gp;
33         this.entityH = entityH;
34
35         fish = new Fish[4];
36         firstX = gp.tileWidth * 6 + 12;
37         getDefaultValue();
38         getFishImage();
39     }
40
41     private void getFishImage(){
42         try{
43             fish[0].img = ImageIO.read(getClass().getResourceAsStream(↵
44                 "/fish/blue.png"));
45
46             fish[1].img = ImageIO.read(getClass().getResourceAsStream(↵
47                 "/fish/pink.png"));
48
49             fish[2].img = ImageIO.read(getClass().getResourceAsStream(↵
50                 "/fish/yellow.png"));
51
52             fish[3].img = ImageIO.read(getClass().getResourceAsStream(↵
53                 "/fish/orange.png"));
54
55         }catch (IOException e){
56             e.printStackTrace();
57         }
58     }
59
60     private void getDefaultValue(){
61         gap = 12*gp.scale;
62         fishWidth = 8 * gp.scale;
63         fishHeight = 4 * gp.scale;
64         speed = 4;
65
66         for(int i = 0; i < 4; i++){
67             fish[i] = new Fish();
68             if (i == 0){
69                 fish[i].y = 10 * gp.scale;
70             }
71             else{
72                 fish[i].y = fish[i - 1].y + gap;
73             }
74             fish[i].x = firstX;
75         }
76     }
```

```
72     }
73
74     public boolean update(){
75         boolean finishedChanges = false;
76         if(run != -1){
77             playFishSE();
78             fish[run].x += speed;
79             fishCount++;
80             if(fish[run].x >= gp.tileWidth*13+12){
81                 gp.ui.showMessage("The " + fishColor[run] + " fish has↵
82                     escaped!!!");
83                 fish[run].finished = true;
84                 fishFinished++;
85                 finishedChanges = true;
86             }
87         }
88         if(fishCount>=fishTimer){
89             entityH.done = true;
90             fishCount=0;
91             run = -1;
92             if (fishFinished > 2){
93                 gp.ui.gameFinished = true;
94                 gp.ui.fishWin = true;
95                 if(gp.ui.playerChoice == gp.ui.fishChoice){
96                     gp.playSE(3);
97                 }
98                 else {
99                     gp.playSE(4);
100                 }
101             }
102             else if(playSE){
103                 gp.playSE(2);
104                 playSE = false;
105             }
106         }
107         return finishedChanges;
108     }
109
110     public void draw(Graphics2D g2){
111         for(int i=0;i<4;i++){
112             if(!fish[i].caught){g2.drawImage(fish[i].img,fish[i].x,↵
113                 fish[i].y,fishWidth,fishHeight,null);}
114         }
115     }
```

```
115
116     public boolean collision(int boatX){
117         int fishCaughtCounter = 0;
118         for(int i=0;i<4;i++){
119             if(boatX>fish[i].x && !fish[i].caught){
120                 fishCaughtCounter++;
121                 fish[i].caught =true;
122                 fishRemaining--;
123                 gp.ui.showMessage("The " + fishColor[i] + " fish is ←
                                caught!");
124             }
125         }
126         //This is to check whether we should play SE
127         return fishCaughtCounter > 0;
128     }
129     public boolean getCaught(int fishNum){
130         return fish[fishNum].caught;
131     }
132     public boolean getFinished(int fishNum){
133         return fish[fishNum].finished;
134     }
135     public int getFishX(int fishNum){
136         return fish[fishNum].x;
137     }
138     public int getFishRemaining() {return fishRemaining;}
139     public int getFishFinished(){return fishFinished;}
140     public void playFishSE(){
141         playSE = true;
142     }
143
144     //For restarting the game
145     public void setDefaultValue(){
146         getDefaultValue();
147         getFishImage();
148         fishFinished = 0;
149         fishRemaining = 4;
150     }
151 }
```

7 Experimental Results, Statistical Tests, Running Scenarios

7.1 Tables



GameID	Rounds	Wins
1	22	Boat
2	28	Fish
3	13	Boat
4	16	Boat
5	23	Draw
6	20	Boat
7	21	Boat
8	27	Fish
9	25	Fish
10	28	Draw
11	26	Fish
12	25	Fish
13	24	Boat
14	23	Fish
15	27	Fish
16	22	Boat
17	26	Fish
18	24	Fish
19	25	Fish
20	23	Draw
21	21	Boat
22	27	Fish
23	15	Boat
24	22	Fish
25	18	Boat
26	24	Fish
27	27	Fish
28	26	Fish

Figure 12: Statistical Data for Tempo, Kleine Fische

Our comprehensive attempt at doing 10,000 test iterations has yielded vital data, enhancing our comprehension of this game. By conducting a thorough examination of these findings, we have acquired a more profound understanding of its mechanisms, tactics, and possible consequences. The comprehensive information provides a strong basis for additional investigation and improvement, allowing us to make well-informed judgments and optimizations in our future game-related initiatives.

7.2 Charts & Evaluations

By conducting a large number of game simulations, we can obtain intriguing statistical data.

- It is advisable to consistently choose the fishermen team, as they possess a 51.8% probability of winning.
- The probability of fishes winning is 36.8%.
- There is a 22.1% probability of the game ending in a draw.
- The mean number of turns in a game is 24.8.
- Fishes do not possess a winning strategy, even in situations where they have the option to make a choice. For instance, when a fish has already reached the sea and needs to migrate, another fish can be selected to make the move instead. Indeed, relocating the closest fish from the sea will result in a marginal increase of 0.1% in your likelihood of success.

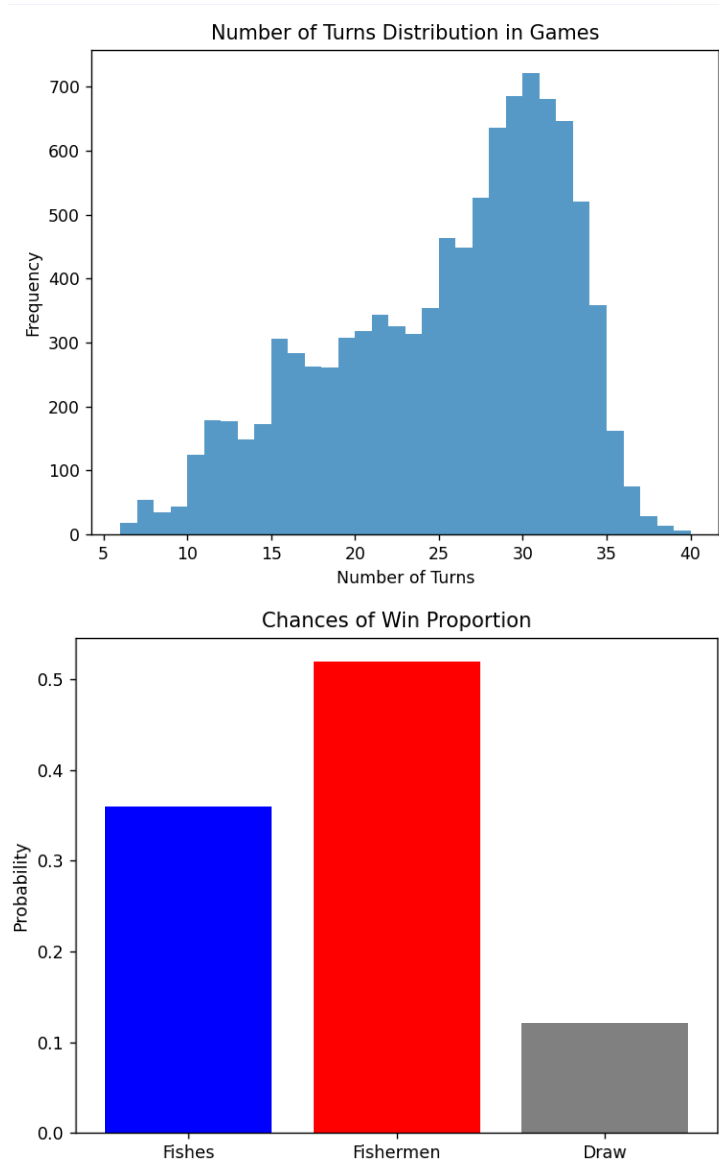


Figure 13: Statistical Data for Tempo, Kleine Fische

8 Conclusions and Future Work

How was the team work:

- This is the first time we code as a team, so a lot of obstacles have happened. We chose Github because many people have used it as a tool for teamwork. At first, we have problems with pull and push and create many conflicts, which lead to many code losses. After this teamwork, we all learned how to use Github professionally, and it can be applied in the future.
- Our path has been marked by significant improvements in communication and interpersonal relationships. We have strengthened our team members' interpersonal interactions and coherence by cultivating open communication. By communicating effectively, we have gained a better knowledge and respect for one another's points of view, successfully addressed differences in a courteous manner, and improved our capacity to work together happily. This better interaction has not only improved our working environment, but has also driven us to greater production and success.
- Our understanding of LaTeX has greatly improved our ability to create documents. Learning LaTeX has given us a powerful tool for quickly and efficiently writing high-quality documents. Using this knowledge, we are better equipped to create aesthetically beautiful and well-organized content, whether for academic papers, reports, or presentations. Our improved understanding of LaTeX illustrates our commitment to continuous learning and progress, allowing us to confidently and accurately address future challenges.

This game absolutely needs some upgrades. As we all see, the win rate of the fish and fishermen is not equal, so there may be some unfair gameplay here. In the future, we may have some adjustments to this game, such as adding a feature where we can choose where the fish stands and where the boat stands at the start of the game (of course it will have some limits for the adjustments) to make the game more fair for the newcomers who know nothing and want to explore the probability of this game. The UI of this game is still not good enough. But we are not artists, so we can do nothing more. However, if we have more time and maybe collaborate with people from other majors, this game can be more interesting.

After simulating a kind of board game, we are looking forward to making another board game imitation. One interesting game that we all love is "The Mind." It is a game where you and your friends have to play cards in a certain order without communicating with each other. This game here is quite simple at first but is still interesting and challenging for us to tackle, like implementing multiplayer over multiple devices, a way to track every achievement and high score, and many other problems that we don't have a chance to take on with "Tempo, kleine Fische." We are excited for many other problems or projects to come, and we can't wait to take them head-on.

9 References

1. Our game.
2. Ideas of our game.
3. Graphic design.
4. Our game.
5. Comparison game.
6. Maybe future project.