NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4240 - SOFTWARE ARCHITECTURE

Fish Miner - Requirements

Quality Attributes

Primary: Modifiability

Secondary: Usability and Performance

Chosen COTS

LibGDX
Scene2d
Ashley ECS

Authors

First Name	Last Name	Email
August Damm	Lindbæk	august.d.lindbak@ntnu.no
Eivind Biti	Holmen	eivinbho@stud.ntnu.no
Emil	Lunde-Bakke	emillu@stud.ntnu.no
Jesper	Seip	jespese@stud.ntnu.no
Salehe	Mortazavi	salehem@stud.ntnu.no
Tale Marie Wille	Stormark	tale.m.w.stormark@ntnu.no

Table of Contents

List of Figures

List of Tables

1 Introduction

In this report, we describe the requirements for a game development project conducted as part of the *TDT4240 Software Architecture* course at NTNU. The goal of this project is to develop a functional multiplayer game for Android, designed based on software architecture principles covered in the course. The game must include a multiplayer feature, an online component, and server-based functionality. Additionally, the architecture will prioritize modifiability as the primary quality attribute and usability and performance as a secondary quality attribute.

The first phase of the project is the Requirement Phase, where we define the game's core mechanics, identify technical constraints, and establish the necessary architectural principles that will guide the implementation. This phase also includes gathering inspiration from similar games, designing initial game sketches, and specifying functional and quality requirements.

1.1 Inspiration

Our selected game concept will extend the classic arcade game Gold Miner into a fishing-themed variant. In the original game, you play as a Gold Miner with the objective of collecting enough valuable items to reach a preset target within 60 seconds, as shown in Figure 1 You achieve this goal by avoiding moles, rocks, and TNT and collecting valuables like gold chunks and diamonds, whose size determines their monetary value. After each level, upgrades can be purchased with the money you collected. The mechanism for collecting the various treasures is implemented as a hook that automatically swings back and forth. When the hook swings to the position you want to aim, you click to fire and whatever the hook hits in its trajectory is retrieved back to the surface. The value of the items you collect are then added to your inventory.

Figure 1: Arcade game Gold Miner.

1.2 Concept Overview

At its core, our game is similar to Gold Miner, but instead of mining for gold, the player will be fishing for fish. Each level has a preset goal, requiring the player to reach a specific score by catching valuable fish before the timer runs out. Beyond simply completing the level, players are encouraged to catch as many fish as possible. As you unlock more levels the difficulty increases. The game saves your best performance: the highest score you reach before losing.

The player sits in a boat at the water's surface, centered horizontally, with the fishing hook swinging back and forth, as shown in Figure 2. The player controls the angle and length of the fishing line by touching the screen. Below, fish of various sizes swim at different speeds and depths. Obstacles like garbage block the hook's path, wasting time if caught but disappearing without taking up space in the boat. In higher levels, we introduce sharks or other sea creatures as obstacles that, if caught, will result in a low or negative score and waste valuable time.

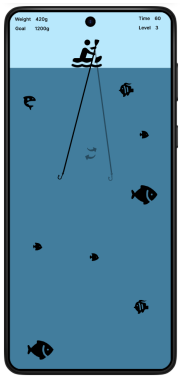

Figure 2: Sketch of the game play in FishMiner.

In another game state, we will introduce a trader/upgrade store, where players can exchange their collected fish for *upgrades / power-ups* that enhance their fishing abilities. Examples of upgrades could be:

- **Long Reel**: an extended reel that will increase the depth the player's hook. This will enable the player to reach fish that are beyond a certain level
- **Heavy Weight**: a weight that increases the speed of the hook's trajectory. Enabling the player to reach the fish faster.

This game concept aligns well with our primary quality attribute, modifiability, as the game will need to be easily extensible. Future updates may introduce new levels, upgrades, fish species, and game elements to keep the experience engaging. Since modifiability is a key focus, we aim to design the game in a way that not only requires this attribute but also actively leverages modifiability patterns during development. By building the core mechanics first and expanding iteratively, we ensure that each phase results in a fully functional version of the game, allowing us to be ambitious while maintaining a stable foundation to fall back on if needed.

2 Functional Requirements

In this project, we have defined the functional requirements necessary to develop a well-structured product. Table 1 presents these requirements along with their priority levels.

2.1 Priority Levels

- **High:** Essential for core gameplay and must be implemented.
- **Medium:** Important for user experience but can be adjusted or postponed.
- **Low:** Nice-to-have features that enhance gameplay but are not critical.

ID	Description	Priority
Core Gameplay Mechanics		
FR1	The hook automatically swings back and forth and is “fired” when the user clicks	High
FR2	The hook must catch a fish when it comes into contact with one	High
FR3	The game must have a countdown timer for each level	High
FR4	If the player does not reach the goal before the timer ends, the game should end	High
FR5	The game must allow progression through levels when the goal is reached	High
FR6	Each new level should increase difficulty by adjusting speed, obstacles, or fish distribution	Medium
FR7	The game must notify the user when time is running out	Low
FR8	The game must show the player’s score and result (win/lose) at the end of each level	Medium
Game World Elements		
FR9	The game must include different types of fish with associated values based on size or rarity	Medium
FR10	There should be obstacles that waste time when hit with the hook	Medium
FR11	Small fish swim near the surface, bigger fish swim in the deep end	Low
Upgrades & Store System		
FR12	The user can earn points based on the fish they catch	High
FR13	The user must be able to access the trader menu	High
FR14	The user must be able to purchase upgrades with acquired currency	Medium
FR15	The game must confirm when an upgrade has been successfully purchased	Medium
Multiplayer & Online Features		
FR16	The user should see the leaderboard	High
FR17	The leaderboard should be updated once a score arrives	Medium
FR18	The user should be able to find themselves on the leaderboard	Low
Platform & User Interaction		
FR19	The user should be greeted with a menu when the app loads	High
FR20	Clicking ”Start Game” should start a new game	High
FR21	The user should be able to pause/resume the game	Medium
FR22	There should be a tutorial available for the user	Medium
FR23	The game must support both portrait and landscape orientations	Low
FR24	The user should be able to mute the game	Low
FR25	The game must support touch input for mobile platforms	High
Data Persistence & Modifiability		
FR26	The game must store user account information	High
FR27	The game must store each user’s highest score (high score)	High

Table 1: Functional Requirements for FishMiner

3 Quality Requirements

Quality requirements define how well our system should perform in key critical areas (Len Bass 2022). In this project, the primary quality attribute is the modifiability, which ensures that the game can be easily extended, maintained, and updated. Usability is a secondary quality attribute that focuses on making the game intuitive and accessible to players. In addition, we have included performance as a key attribute to ensure that the game remains responsive, maintains a stable frame rate, and provides a smooth user experience on a variety of devices.

The following subsections outline specific quality attribute scenarios, detailing expected stimuli, responses, and measurable outcomes for modifiability, usability and performance.

3.1 Modifiability Quality Attributes

Modifiability ensures that developers can efficiently make changes to the system, whether adding new entities, modifying existing functionality, or updating configurations. According to Bass et al., modifiability is achieved by limiting the number of modules affected by a change, controlling the cost of change, and supporting testability (Len Bass 2022). The following scenarios illustrate how modifiability is maintained in our system:

ID	**QA-M1**
Source	Developer
Stimulus	Wants to add a new entity or upgrade
Artifacts	Entities, components
Environment	Runtime, Design time
Response	Make modification, deploy modification, use Component-Based Design for flexibility
Response Measure	The new product or feature is implemented within 3 hours

Table 2: Quality Attribute Scenario: QA-M1

ID	**QA-M2**
Source	Developer
Stimulus	Wants to change existing entity or upgrade
Artifacts	Entities, components
Environment	Runtime
Response	Make modification, deploy modification
Response Measure	The update is implemented with a maximum of 3 affected artifacts

Table 3: Quality Attribute Scenario: QA-M2

ID	QA-M3
Source	Developer
Stimulus	Wants to change game configurations (e.g. screen size, entity speed, score goal)
Artifacts	Game Configuration System
Environment	Design time
Response	Developer makes a change in a single configuration entity and the change is automatically applied to all affected artifacts
Response Measure	Make change in a single location and the change is applied immediately across affected system, without additional manual updates

Table 4: Quality Attribute Scenario: QA-M3

3.2 Usability Quality Requirement

Usability ensures that the game provides an intuitive experience for players, allowing them to learn, interact, and provide feedback without much friction. According to Bass et al., usability is supported by design decisions that make the system easy to learn and use, reduce the risk of user error, and improve user satisfaction (Len Bass 2022). This includes a clear tutorial, intuitive game flow and easy access to settings. The following scenarios define how usability is measured and achieved in our system:

ID	QA-U1
Source	End user
Stimulus	Wants to improve gameplay by providing development team with directive to: - Modify or add features/upgrades, or - Fix bugs or game balance
Artifacts	UI, database
Environment	Runtime
Response	User fills in and sends form to development team. Development team receives and evaluates directive/feedback
Response Measure	User understands the feedback system with no more than one error. 70% of users should be able to fill in the form within 2 minutes

Table 5: Quality Attribute Scenario: QA-U1

ID	QA-U2
Source	User / player
Stimulus	Wants to learn how to play the game
Artifacts	GUI, database
Environment	Runtime
Response	User opens the game, clicks on help/tutorial, and can scroll through tutorial images.
Response Measure	Finishes the tutorial within 1 minute and is able to play the game with no more than 3 errors in the first game session.

Table 6: Quality Attribute Scenario: QA-U2

ID	QA-U3
Source	User / player
Stimulus	Wants to help developers solve bugs and improve the game
Artifacts	GUI, Developer Email Account
Environment	Runtime
Response	The user clicks the feedback button, chooses a subject (bug, feature request, balance, or other). They are then redirected to their email application with the developer team's email pre-filled as the recipient and the selected subject as the email topic.
Response Measure	User knowledge gained per received mail (measured in distinct issues or suggestions and the percentage of feedback mails covering recurring topics)

Table 7: Quality Attribute Scenario: QA-U3

ID	QA-U4
Source	User / player
Stimulus	Wants to customize the game experience based on personal preferences
Artifacts	GUI, Settings
Environment	Runtime
Response	The user-player enters the settings menu and is able to mute the music or all audio
Response Measure	The user is able to find and change preferences within 30 seconds

Table 8: Quality Attribute Scenario: QA-U4

3.3 Performance Quality Requirement

Performance is essential to delivering a smooth and responsive gaming experience. As discussed in the literature, performance refers to the system's ability to meet timing requirements under defined

conditions, including throughput, latency, and response time (Len Bass 2022). This includes maintaining consistent frame rates, fast load times, and minimal input latency. These qualities ensure that the game is playable on a wide range of devices, including mid-range Android smartphones.

ID	**QA-P1**
Source	End user
Stimulus	User is playing the game during a standard gameplay session
Artifacts	Game engine, rendering system
Environment	Runtime
Response	Game continues to run smoothly without frame drops
Response Measure	Maintain at least 30 FPS on mid-range Android devices

Table 9: Quality Attribute Scenario: QA-P1

ID	**QA-P2**
Source	Logged-in user
Stimulus	User opens the leaderboard to view current rankings
Artifacts	Firebase Firestore, Leaderboard UI
Environment	Runtime with internet connection
Response	The system queries Firestore, retrieves the top 10 scores, and displays them along with the current user's rank
Response Measure	Leaderboard data is retrieved and correctly rendered within 2 seconds, including user's own score when logged in

Table 10: Quality Attribute Scenario: QA-P2

ID	**QA-P3**
Source	Player after completing a level
Stimulus	Player purchases an upgrade using earned currency
Artifacts	Upgrade system, in-game currency system, persistent storage
Environment	Runtime
Response	The currency amount decreases, the upgrade is added to the player's inventory, and is available for the next level
Response Measure	Upgrade transaction is completed and reflected in inventory within 1 second; data persists and upgrade is usable in the next session without error

Table 11: Quality Attribute Scenario: QA-P3

4 COTS Components and Technical Constraints

Our game is being developed for the Android platform, an open-source mobile operating system. The game will primarily be written in Java to ensure compatibility with Android development standards.

4.1 LibGDX

To build our multiplayer game, we have selected LibGDX, a Java-based, open-source game development framework that supports cross-platform deployment (libGDX 2025). One key advantage of LibGDX is that it allows us to run and debug the game directly on a computer without requiring an Android device or emulator. Additionally, LibGDX is actively maintained and well documented, making it a reliable choice for our project. LibGDX also includes a Screen management system, allowing different game states—such as menus, gameplay, and settings—to be controlled from a central Game class. This modular approach improves organization and maintainability. Although LibGDX supports multiple platforms, including iOS and web-based deployment, our focus will remain solely on Android throughout development and testing.

4.2 Scene2d

We are planning to use LibGDX Scene2d, which is a 2D scene graph for building applications (libGDX Wiki 2024). The Scene2d framework simplifies UI and actor-based rendering, but it also introduces some constraints in our project. While it provides easy-to-use stage and actor management, it is primarily designed for hierarchical UI elements rather than a physics-driven game. Its actor-based approach may be less flexible for handling complex, real-time game objects compared to using raw LibGDX SpriteBatch rendering. In addition, it might introduce some performance overhead, as the abstraction adds extra processing, which may impact performance on lower-end Android devices.

4.3 Touch-based Input

Given that our game will rely on touch-based input, designing an intuitive control scheme is essential. Unlike traditional PC or console games, which have dedicated input devices, our game must be playable using gestures and taps without overwhelming the user with complex interactions.

4.4 Ashley ECS

As we are planning to use an Entity-Component-System (ECS) approach, we want to use the Ashley framework, which is a tiny entity framework (libGDX Wiki 2022). Using Ashley in our fishing game introduces architectural constraints that impact how we design and structure game objects. Since Ashley enforces an Entity-Component-System approach rather than traditional inheritance-based object-oriented design, all game entities (fish, hook, and objects) must be composed of independent components instead of inheriting behaviors from parent classes.

5 Issues (Optional)

One challenge we faced during the development of the game was determining the balance between usability and modifiability when designing the UI. For instace, implementing both landscape and portrait modes affected layout consistency and required more UI-specific code.

6 Changes

Date	Change history	Comment/Reason
2025-03-04	Added "Performance" as a quality attribute alongside Modifiability and Usability	Recognized that maintaining a smooth and responsive game loop was essential for the player experience, especially on mobile devices.
2025-03-06	Removed support for landscape orientation	Decided that maintaining both orientations was out of scope for the course, where architectural quality was prioritized over UI flexibility.
2025-03-07	Changed the scoring system from time-based per level to a cumulative high-score model	Simplified the logic and focused the scoring mechanism around fish value, making it easier to track progress and integrate with the leaderboard.
2025-04-10	Removed in-game error and improvement reporting feature	User reporting was deprioritized to focus on core features.
2025-04-18	Changed the tutorial format from a video to scrollable images with text	The image-based format is easier to implement, loads faster, and aligns better with the game's modular UI structure.

Table 12: Change History Table

7 Individual Contribution

Member	Contributions
August	COTS components, Technical Constraints and creating game idea
Eivind	Functional Requirements and creating game idea
Jesper	Introduction, Quality Requirements and creating game idea
Tale	Introduction, formatting the document in LaTex, creating game idea, reviewing and refining the text
Salehe	Functional Requirements, creating game idea
Emil	Functional requirements, quality attributes, creating game idea

Table 13: Individual Contributions

Bibliography

Len Bass Paul Clements, Rick Kazman (2022). *Software Architecture in Practice*. Accessed: 24 Feb. 2025. Addison-Wesley.

libGDX (2025). *libGDX*. Accessed: 2025. URL: https://libgdx.com/.

libGDX Wiki (2022). *Ashley*. Accessed: 2022. URL: https://github.com/libgdx/ashley/wiki.

— (2024). *Scene2d*. Accessed: 2024. URL: https://libgdx.com/wiki/graphics/2d/scene2d/scene2d.