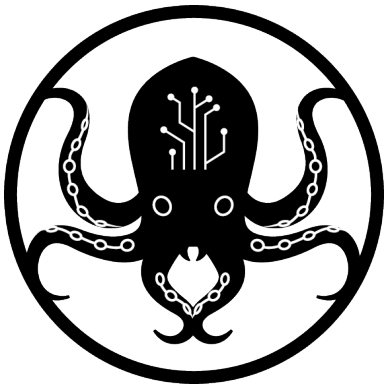
Bachelor Thesis
Documentation

3D-Module for 'Algorithm & Data Structure Visualizer (ADV)'

Semester: Spring 2025



Date: 12.06.2025

Project Team: Christoph Bodschwinna
Philipp Frank

Project Advisor: Thomas Letsch

School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Abstract

Introduction

The Algorithms and Data Structures course uses a tool called Algorithm & Data Structure Visualizer (ADV) to help students better understand the behavior of their code. During coding exercises, it allows students to step through their algorithms and see a visual representation of how the code executes. Since the ADV originally supported only 2D visualizations, a separate standalone tool was developed using JOGL to display 3D landscapes for pathfinding algorithms. The goal of this project was to create a new module for the ADV that can render 3D landscapes and serve as a replacement for the standalone JOGL-based solution.

Problem

A central challenge was the efficient handling of large datasets. Until now, the ADV supported only 2D elements and algorithms that typically generated no more than 20 to 30 snapshots, each with around 10 to 20 elements. In contrast, the new landscape module needed to process matrices as large as 200 by 200, resulting in up to 40,000 data points, and support as many as 150 snapshots per session. A previous semester thesis attempted this before, but the resulting prototype suffered from major performance limitations and was not suitable for practical use.

Result

The newly developed landscape module supports both automatically generated and manually defined matrices of any size. To enhance user interaction, the generated landscapes include mouse and keyboard controls. Like all other ADV modules, it relies on a snapshot-based communication protocol. To handle the significantly larger data volume, the protocol was extended to support partial snapshots. This allows the ADV-UI to construct new snapshots incrementally by applying only the changes to the previous state, which greatly reduces both data transmission and memory usage. Additionally, since the ADV had not been maintained for several years, major version upgrades were performed across the entire application, including updates to Java, JavaFX, and Gradle.

Keywords: Software, 3D Visualization, JavaFX

Acknowledgements

We would like to express our sincere gratitude to everyone who supported us throughout this bachelor thesis.

Above all, we are especially thankful to Thomas Letsch for his invaluable supervision, ongoing guidance and constructive feedback throughout the entire process.

Contents

Part I

Summary

Chapter 1

Management Summary

1.1 Introduction

The Algorithm & Data Structure Visualizer (ADV) is a tool used in the Algorithms and Data Structures course to help students better understand how their code behaves through step-by-step visualizations. Until now, the ADV supported only 2D visualizations. To enable 3D pathfinding exercises, a separate JOGL-based tool had previously been developed. This project aimed to integrate that functionality into the ADV itself by developing a new 3D landscape module.

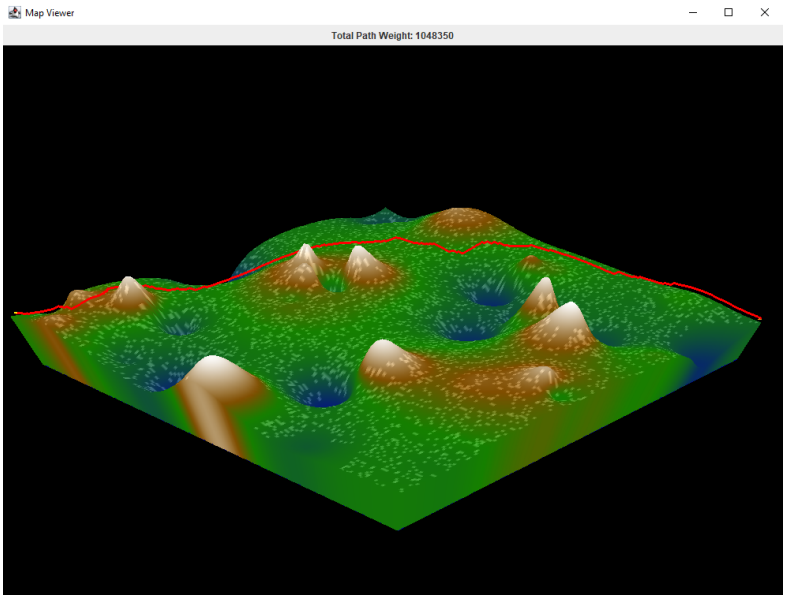


Figure 1.1: 3D Landscape in the JOGL Tool

One of the key challenges was ensuring the new module could efficiently handle much larger datasets than before. Whereas existing ADV modules typically processed no more than 30 snapshots of small 2D structures, the landscape module needed to support matrices of up to 200×200 elements and over 150 snapshots per session.

1.2 Technologies

The ADV is structured as a client-server application and consists of three multi-module repositories: one for shared components, one for the UI, and one for sending data to the UI. Additionally, a separate repository contains example projects that demonstrate how to use the various ADV modules. The application is written in Java with JavaFX for the UI and uses Gradle as its build tool.

Since the application had not been actively maintained for several years, many of its dependencies were outdated or deprecated. As a first step, the entire codebase was upgraded from Java and JavaFX version 11 to version 21, and Gradle from version 4 to 8.13.

Previously, Travis CI was used to build and deploy the JAR files of the different repositories to Bintray, from where they were published to JCenter and Maven Central. However, Bintray was shut down in 2021, followed by JCenter in 2024. As a result, the CI/CD pipeline was migrated to GitLab CI. Going forward, JAR files will no longer be deployed to Maven Central but instead distributed directly within the course environment.

1.3 Result

The newly developed landscape module enables the generation of matrices of any size or the use of manually defined ones. Like other ADV modules, it relies on a snapshot-based communication protocol. To support significantly larger datasets, the protocol was extended to allow partial snapshots. Through this, it is possible to construct new snapshots incrementally by applying only the differences to the previous state, thereby reducing data transmission and memory usage. The complete matrix is now sent only once at the beginning of a session. Additionally, batch updates for drawing points were introduced to further reduce the number of snapshots and memory consumption.

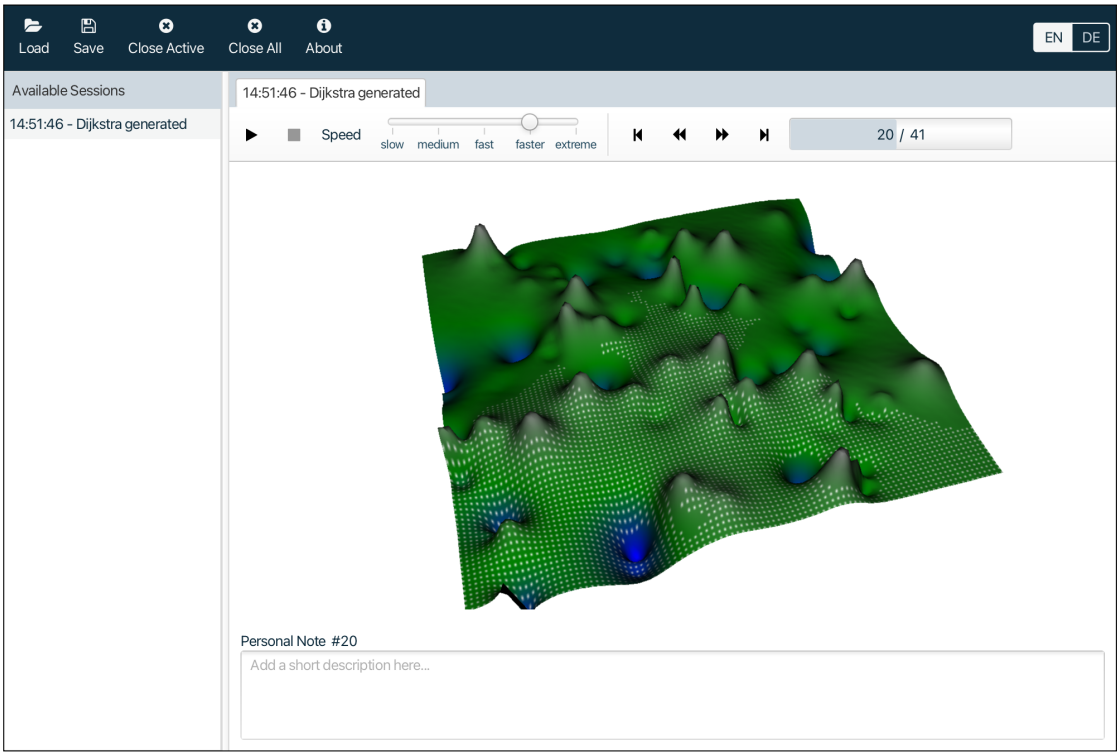


Figure 1.2: New Landscape Module in the ADV

The module supports both keyboard and mouse controls for intuitive 3D navigation and includes utility functions for calculating distances and weights, which are helpful for implementing pathfinding algorithms. These features make it well-suited for use in interactive coding exercises within the ADV environment. Additionally, students and exercise supervisors can configure the initial landscape setup, including the rotation angle, tilt and the camera's starting position.

Since the new landscape module processes significantly larger datasets, memory usage remains an important consideration. This can be controlled by adjusting the matrix size and the number of snapshots created. The number of snapshots depends on how many points are updated in each batch. When more points are updated at once, fewer snapshots are required. Exercise supervisors should keep this in mind when designing new exercises to maintain a good balance between performance and resource consumption.

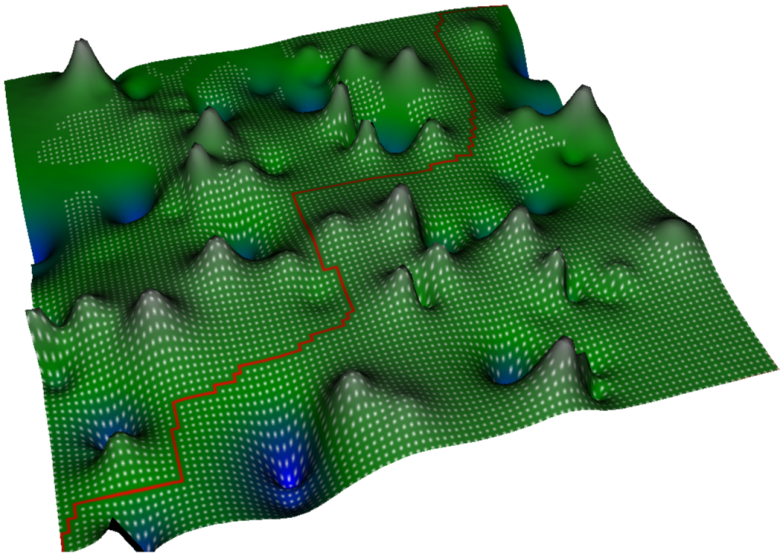


Figure 1.3: Last Snapshot of a Landscape Module

Beyond the module itself, several bugs were fixed, and usability improvements were introduced. This includes two additional replay speeds and an About page that displays student contributions.

1.4 Conclusion

This project successfully developed a new landscape module for the ADV application, adding support for 3D graphics. Additionally, the application was brought up to date through key technology upgrades, resolving several known bugs and implementing existing feature requests.

Future improvements may include major version upgrades, such as updating Guice from version 5 to 7 and migrating JUnit from version 4 to 5. These upgrades would require extensive structural changes, which is why they were not addressed during this project. Another valuable enhancement would be enabling snapshot selection via an input field. Since landscape module sessions can include 100 or more snapshots, allowing users to jump directly to a specific snapshot would improve usability. Additionally, the partial snapshot functionality could be expanded to support relations, enabling graph-related modules to benefit from the same protocol enhancements.

In conclusion, the developed module is a cleaner and improved successor to the prior standalone JOGL solution and is expected to become a key component in future exercises, helping students gain a deeper understanding of complex algorithms.

Part II

Product Documentation

Chapter 2

Requirements

2.1 Functional Requirements

2.1.1 Use Case Diagram

Actors

- **Student**:
  - **Goal**: Implement an algorithm for a provided exercise.
- **Exercise Supervisor**:
  - **Goal**: Create an exercise for the student to learn about algorithms.
- **ADV-UI User**:
  - **Goal**: View a landscape and observe the steps an algorithm takes.

Use Case Diagram

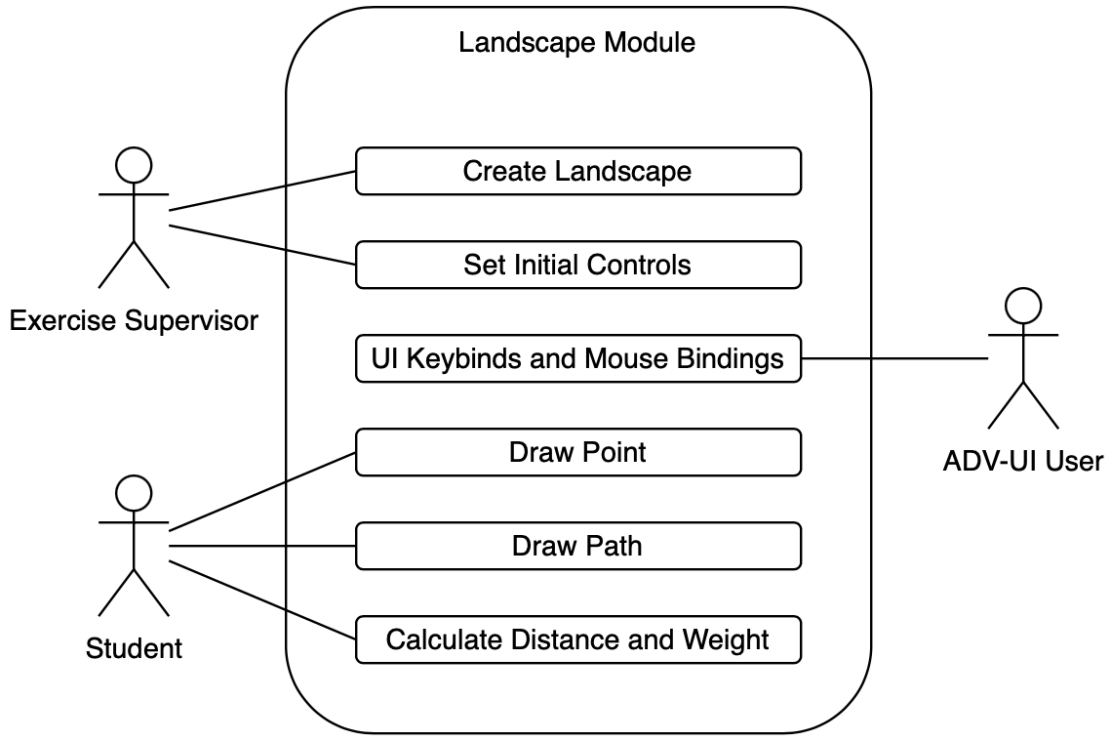


Figure 2.1: Use Case Diagram

2.1.2 Use Case Description

All use cases are written in a casual format.

Use Cases

- **UC-1 Calculate Distance and Weight:** The student can use a function to calculate the distance and another to calculate the weight between two points.
- **UC-2 Create Landscape:** The student and supervisor can create a landscape by generating one with a seed, without a seed, or by directly providing a matrix as the landscape.
- **UC-3 UI Keybinds and Mouse Bindings:** The ADV-UI user can navigate the landscape using keyboard shortcuts (move and zoom) and use the mouse to rotate and zoom the landscape.
- **UC-4 Draw Point:** The supervisor can change the color of a given point in the landscape.
- **UC-5 Draw Path:** The supervisor can provide a list of points in the landscape that are connected, and set the color of the path.
- **UC-6 Set Initial Controls:** The student and supervisor are able to configure the initial landscape settings such as position, viewing angle and zoom.

Optional Improvements

The following list contains bugs that were either already known or encountered during implementation, as well as small feature requests that are not part of the core requirements. The improvements are intended to be addressed either if time permits or in parallel with the main use cases.

- **Bug:** The Unicode infinity symbol cannot be used, as it prevents the session from being saved.
- **Bug:** Flickering occurs when executing a step for the first time in a session.
- **Bug:** Umlauts are not allowed in session names.
- **Bug:** Sessions can be detached via drag and drop, but doing so results in a lost session or a blank (white) screen.
- **Bug:** Changing the language does not update the labels for the replay speed steps.
- **Feature:** Add a new replay speed step to loop rapidly through a session. This can serve as a workaround for the flickering issue.
- **Feature:** Add an `About` page to the application.

Use Case Priority

The following list defines the priority of the previously defined use cases. The date when they were planned to be implemented can be found in the section 8.6 Planned Product Releases.

1. UC-2 Create Landscape
2. UC-3 UI Keybinds and Mouse Bindings
3. UC-4 Draw Point
4. UC-5 Draw Path
5. UC-6 Set Initial Controls
6. UC-1 Calculate Distance and Weight

2.2 Non-Functional Requirements

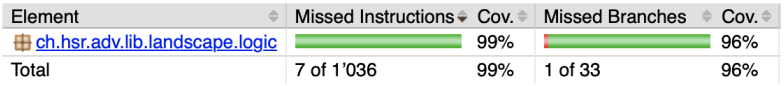
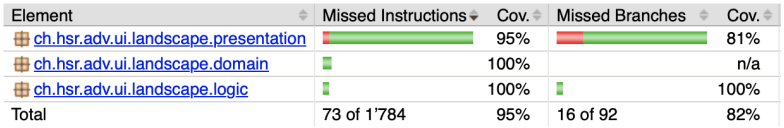
ID	NFR-1
Category	Maintainability
Requirement	The landscape module in the ADV-Lib and ADV-UI should be automatically tested.
Measures	Tests should cover 80% of the code of the new module.
Measuring technique	Tests will automatically run on every Git commit and will fail if the coverage target isn't met.
Priority	High
Result	 Figure 2.2: Landscape Module ADV-Lib Code Coverage  Figure 2.3: Landscape Module ADV-UI Code Coverage

Table 2.1: Non-Functional Requirement: Maintainability Code Coverage

ID	NFR-2
Category	Maintainability
Requirement	Maintain a clean, understandable and modular codebase that adheres to standard coding practices.
Measure	• No new issues reported by SonarQube Cloud • No style violations reported by Checkstyle
Measurement Technique	• Analyze code using SonarQube Cloud to detect duplications, complexity, code smells and maintainability issues • Run Checkstyle using the project's defined ruleset to verify adherence to code formatting and style guidelines
Priority	Medium
Result	 Figure 2.4: ADV-Lib Sonar Overview  Figure 2.5: ADV-UI Sonar Overview Total files checked: 4 / Total violations: 0 / Files with violations: 0 Figure 2.6: ADV-Lib Checkstyle Summary Total files checked: 6 / Total violations: 0 / Files with violations: 0 Figure 2.7: ADV-UI Checkstyle Summary

Table 2.2: Non-Functional Requirement: Maintainability Code

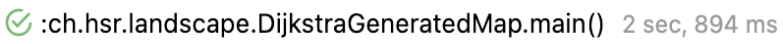
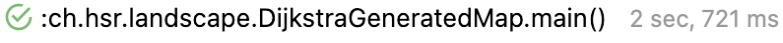
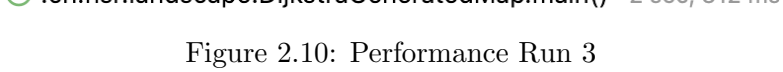
ID	NFR-3
Category	Performance
Requirement	The system must be able to process a 100x100 matrix and update the display at a rate of at least 10 snapshots per second.
Measures	The main function in a sample code (e.g. DijkstraGeneratedMap) should maintain an average of 10 updates per second during execution.
Measuring Technique	Run the DijkstraGeneratedMap in the target codebase and measure execution time of the main function.
Priority	High
Result	**Test Environment** Devise: MacBook Pro Processor: 2.6 GHz 6-Core Intel Core i7 Memory: 16 GB 2400 MHz DDR4 Student: Software Sequoia 15.5 50 Snapshots are sent per run.  Figure 2.8: Performance Run 1  Figure 2.9: Performance Run 2  Figure 2.10: Performance Run 3

Table 2.3: Non-Functional Requirement: Performance – Matrix Processing Speed

2.2.1 Verification of Non-Functional Requirements

The non-functional requirements (NFRs) that can be verified automatically will be assessed throughout the entire implementation phase. The remaining NFRs will be evaluated during week 16.

Chapter 3

Domain Analysis

3.1 Domain Model

The following domain model is limited to the scope of the newly added module.

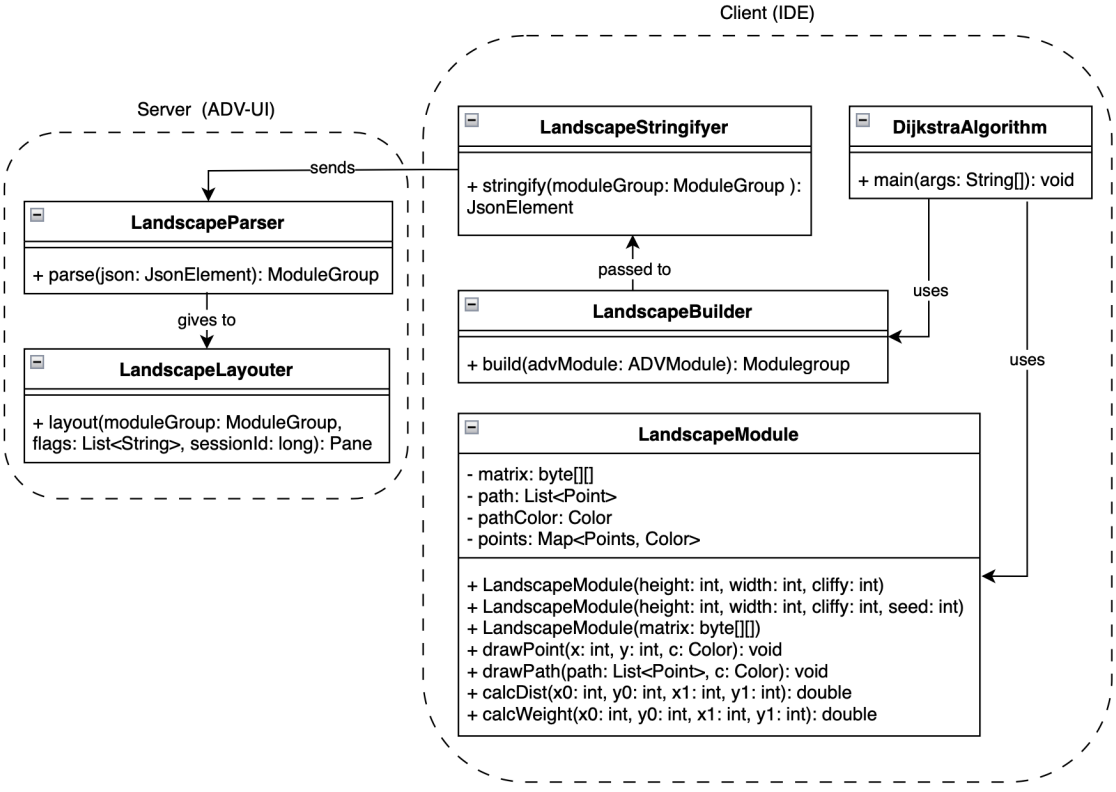


Figure 3.1: Domain model

3.1.1 Explanations

This section provides some clarifications for the domain model.

DijkstraAlgorithm

This class is an example of an algorithm that uses the ADV-UI to display the algorithm. The main function uses the module to create the landscape, set colors for points, or set a path. The builder is then called to create the element that is being sent to the ADV-UI.

3.2 Additional Explanations

The following section provides explanations that are helpful for understanding the solution for the project and cannot be shown or explained inside the domain model.

3.2.1 Server and Client

The term server generally refers to the ADV-UI, while client refers to the ADV-User_Codebase together with the ADV-Lib.

3.2.2 Projects

The ADV consists of four main projects:

- `ADV-Commons`: Contains classes shared by both ADV-Lib and ADV-UI.
- `ADV-UI`: Runs the UI that displays the snapshots sent by the client.
- `ADV-Lib`: A library used in code that sends snapshots to the UI.
- `ADV-User_Codebase`: Holds examples that use the ADV-Lib library.

Both ADV-UI and ADV-Lib depend on ADV-Commons. The ADV-User_Codebase depends on both ADV-UI and ADV-Lib.

Project Structure

All projects, except for the ADV user codebase, follow a structure typical of a multi-module project. This includes a core module that contains classes used by every other module and feature modules such as the array module or graph module, which contain classes specific to their functionality. If a module requires a different parser than the default, it is added within the corresponding feature module. The ADV user codebase differs because it consists of a single module where all example code is defined.

3.2.3 Snapshots

The ADV is built around snapshots, which are saved in a session. Each snapshot always contains all the necessary information to display it in the UI. A snapshot includes an ID, a description and a list of modules referred to as `moduleGroups`. For simplicity, a module group can be considered as a single module. A snapshot may contain multiple modules, either of the same type or of different types. Each module includes a list of `ADVElements` and `ADVRelations`. An element defines the text that is displayed, its position, and its style. This could represent a node in a graph or tree, or an index in an array. A relation defines a connection between two elements, including its label and visual styling. An example of a snapshot is provided in the appendix in chapter 9 Example Snapshot.

Sending a snapshot

A socket is used to send the snapshot from the client to the server. Upon arrival, the server processes the snapshot and adds it to the session to be displayed in the UI.

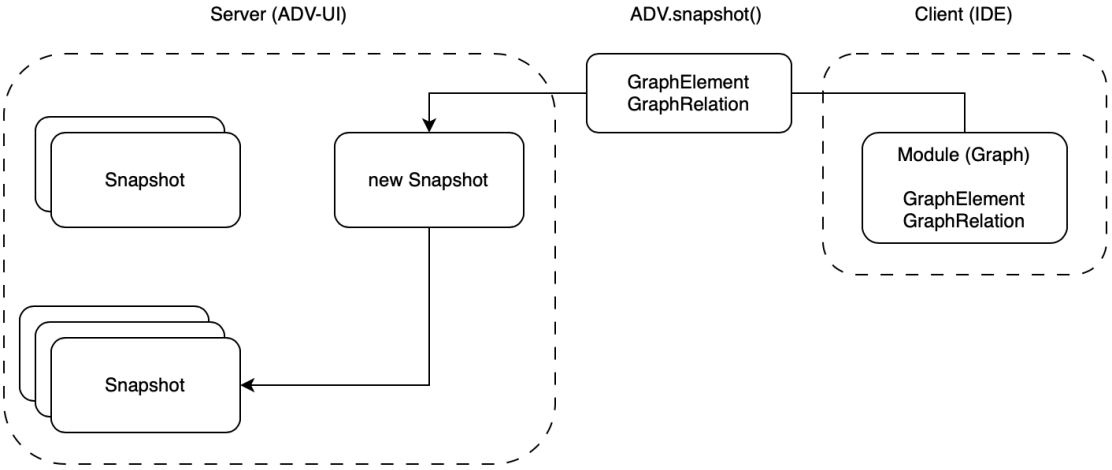


Figure 3.2: Snapshot

Chapter 4

Architecture

The following chapter focuses exclusively on the landscape module and is based on the arc42 method.[9]

4.1 Scope and Context

The application's functionality is described in Section 2.1 Functional Requirements. The scope of this project is limited to adding a new module to handle 3D graphics to the application, while all other changes are considered optional.

4.2 Solution Strategy

This section outlines some of the core decisions, while the architectural decisions regarding technology can be found in section 4.4 Architectural Decisions.

4.2.1 Snapshot vs. Dynamic Landscape Updates

For the implementation of the landscape, two approaches were considered.

The first approach was to use the existing snapshot-based system. Its main advantage is that functionality for replaying, saving and loading sessions is already implemented and tested. Additionally, integrating the new module into this system would preserve consistency with the rest of the application. However, this approach has notable performance drawbacks: each snapshot transmits the entire dataset, which becomes inefficient when working with large matrices (e.g., 200×200 elements). Every snapshot must be fully parsed and stringified, leading to significant overhead. Moreover, when a snapshot is loaded, the user temporarily loses focus of the 3D element, since the landscape is re-rendered for each snapshot.

The second approach considered the development of a new protocol independent of the snapshot mechanism, offering greater flexibility. Instead of sending complete data with every snapshot, the matrix would be transmitted once at the beginning to render the

landscape, followed by incremental updates to draw points and paths. These updates could be rendered dynamically, significantly improving performance and load efficiency. However, this approach would not benefit from the existing features of the application, such as session saving and loading, step-by-step replay, adjustable replay speed, or from preserving the overall look and feel.

Ultimately, the decision was made to retain the snapshot-based system. Its performance limitations can be addressed by enhancing the current implementation to support partial updates, where only the differences relative to the previous snapshot are transmitted. This ensures that the matrix needs to be sent only once initially, while preserving the user experience and integration with the current architecture. This solution combines compatibility with improved efficiency.

4.2.2 Partial Snapshot

Snapshots can include multiple modules simultaneously. To support partial updates to a module, a consistent identifier is required. This identifier consists of the module's position and name. When both match a module in the previous snapshot, elements without IDs are added, and elements with the same ID are updated accordingly. Removing elements is not supported through partial snapshots and requires the creation of a new full snapshot.

The partial snapshot functionality currently supports only the addition of `ADVElements` and new modules. `ADVRelations` are not supported in this context because they do not have identifiers and therefore cannot be reliably matched or updated.

To reduce type-related errors while keeping overhead low, the partial snapshot mechanism uses an enumeration defined in the core of ADV-Lib to control which data is transmitted. This ensures consistent behavior and type safety during snapshot transmission.

Partial snapshots rely on the most recent fully processed snapshot stored in the snapshot store. As a result, they cannot be used as the first snapshot of a session. Attempting to do so results in an exception, which helps ensure that all required elements are present and reduces the likelihood of null pointer exceptions. For the same reason, it is not possible to send multiple partial snapshots at the same time.

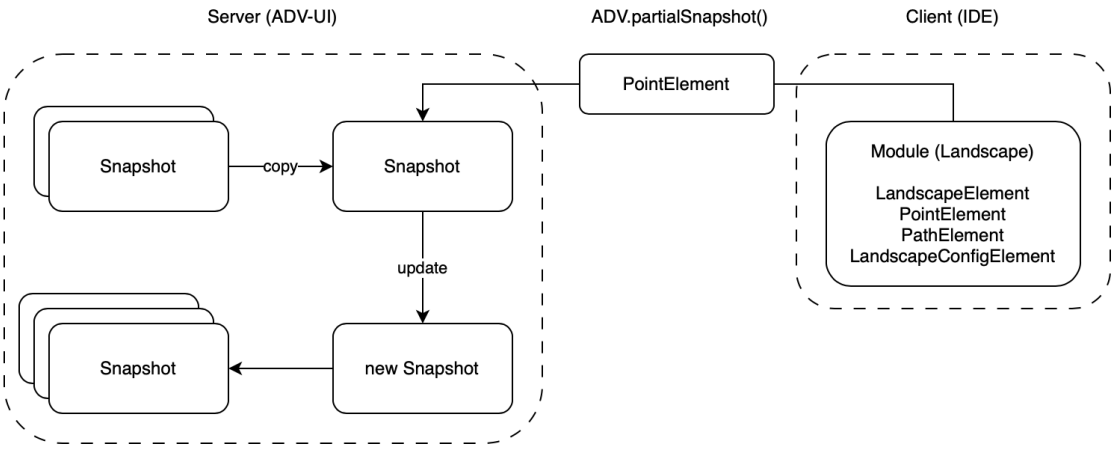


Figure 4.1: Partial Snapshot

4.2.3 Splitting Elements in the LandscapeModule

Typically, a module contains multiple elements of the same type, all handled in a uniform manner. In the case of the landscape module, however, the elements are divided into four distinct types: `point`, `path`, `config` and `landscape`. This separation allows each type to be processed individually and enables selective updates. For example, individual points or paths can be updated without the need to resend the entire matrix or configuration, both of which remain unchanged after initialization.

4.2.4 Generating the Landscape Matrix

The landscape matrix is generated using an improved version of the algorithm from the previous implementation. While the original approach already produced matrices normalized within the range of -128 to 127, the updated version enhances functionality by introducing seed-based reproducibility. It retains core features such as randomness, user-defined terrain complexity, and the influence of surrounding elevation points. With these improvements, the matrix can be generated consistently across sessions, eliminating the need for further refinement within the UI component.

Seeding the Landscape Matrix

The generation process begins by creating a set of seed points, which serve as the basis for the terrain's topography. The number of seed points is determined as a proportion of the total number of cells in the matrix, based on a user-defined parameter that controls terrain ruggedness.

Each seed point has three attributes: a horizontal position, a vertical position (both within matrix bounds), and an elevation value. These attributes are assigned randomly. Elevation values may be either positive or negative, allowing the terrain to include both hills and valleys. This seeded randomness results in a diverse and natural-looking distribution of terrain features across the matrix.

Calculating Cell Values

After seeding, the algorithm calculates an elevation value for each individual cell in the matrix. This is done using a weighted average approach. For each cell, the squared Euclidean distance to every seed point is calculated. Each seed's contribution is weighted inversely proportional to this distance, so that nearby seeds influence the cell more strongly than distant ones.

The final elevation value of a cell is computed by averaging the contributions from all seed points, with each one scaled by its corresponding weight. This produces smooth and continuous elevation transitions across the landscape. While performing these calculations, the algorithm also tracks the minimum and maximum elevation values. These are required for the subsequent normalization step.

Normalizing the Matrix

Once all elevation values are computed, the matrix is normalized to ensure a consistent value range. Positive values are scaled relative to the highest positive elevation, and negative values are scaled relative to the lowest negative elevation. This dual-scaling method preserves the contrast between high and low terrain features, ensuring that all values fall within a defined display range suitable for rendering or further processing.

4.2.5 UI Matrix Refinement using Bicubic Interpolation

The matrix used to represent a landscape can be either automatically generated or manually created. In both cases, the matrix is sent to the user interface for rendering. When the matrix is created manually, it must be refined to ensure that the resulting landscape features smooth hills and valleys. To refine a matrix by a given factor, bicubic interpolation is used to estimate new values between the original data points. This approach increases the resolution while preserving smooth transitions in the data.

Coordinate Mapping

Each point in the refined matrix is mapped back to a fractional coordinate in the original matrix:

$$x = \frac{\text{row}}{\text{factor}}, \quad y = \frac{\text{col}}{\text{factor}}$$

where `factor` is the refinement factor, and `row`, `col` are indices in the refined matrix.

Local Neighborhood

For each interpolated point at position (x, y), a 4×4 patch of neighboring values is extracted from the original matrix, centered around $(\lfloor x \rfloor, \lfloor y \rfloor)$. Border values are clamped to avoid accessing indices outside the matrix bounds.

Interpolation Method

The interpolation is performed in two steps:

1. Cubic interpolation in the y-direction is applied to each of the 4 rows in the patch, using the fractional part of y.
2. Cubic interpolation in the x-direction is applied to the 4 intermediate results, using the fractional part of x.

The cubic interpolation is based on the Catmull–Rom spline, which uses four values v_0, v_1, v_2, v_3 and a parameter $t \in [0, 1]$:

$$\text{interpolate}(v, t) = v_1 + 0.5 \cdot t \cdot (v_2 - v_0 + t \cdot (2v_0 - 5v_1 + 4v_2 - v_3 + t \cdot (3(v_1 - v_2) + v_3 - v_0)))$$

This interpolation is applied first in one dimension, then in the other, resulting in a smooth value for the refined matrix. [12]

4.2.6 Landscape Rendering

To render a landscape, a `TriangleMesh` is used. Triangle meshes consist of connected triangles that approximate the surface of 3D objects using flat facets. Each triangle is defined by three vertices, and each square in the input matrix grid is represented by two such triangles. A `MeshView` is then created from the triangle mesh, which allows a texture to be mapped onto it.

The texture is generated from the same matrix and takes the form of a `WritableImage`, with dimensions matching or scaling proportionally to the matrix. This ensures that the texture resolution remains sharp when applied to the mesh. Pixel colors in the image are set based on matrix values. Higher elevations are rendered in gray tones to represent hills, while lower values are colored blue to depict valleys. Finally, the mesh view is rendered within the scene.

4.2.7 Texture vs. 3D-Objects for Points and Paths

To render points and paths on the surface, two approaches were considered.

The first approach draws them directly onto a texture that is then mapped onto the 3D surface. This method is highly efficient in terms of performance and memory usage, as it avoids adding extra 3D geometry. It also integrates well with JavaFX's rendering system and requires no custom shaders. The main drawback is reduced visual clarity. Since it relies on texture resolution, fine details like narrow paths or closely spaced points can appear blurred, especially on curved or steep areas.

The second approach uses 3D objects: spheres for points and cylinders for path segments, positioned and rotated appropriately in the scene. This results in clearer visuals with consistent appearance regardless of resolution. Lighting and shading further improve visibility. However, this method adds complexity and increases GPU memory usage. Converting path segments into scaled and rotated cylinders is more involved, and performance may suffer with many objects.

Ultimately, the texture-based approach was chosen for its simplicity and performance. To maintain sharpness, a texture scale can be configured through the module. A fixed resolution cannot be set directly, because the resolution must remain a multiple of the matrix dimensions to ensure proper alignment and visual quality. Using a scale factor guarantees that the generated texture fits the matrix structure precisely while still allowing resolution adjustments as needed.

4.2.8 Path Rendering Using Bresenham's Line Algorithm

To visualize a discrete path on a pixel grid, each pair of consecutive points is connected by a straight line. These lines are rasterized using Bresenham's algorithm, a method well-suited for digital environments due to its use of integer arithmetic and efficient decision logic. Let a path consist of a sequence of points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$. For each pair of points (x_i, y_i) and (x_{i+1}, y_{i+1}), a discrete approximation of the straight line is drawn using Bresenham's algorithm.

Bresenham's Algorithm

Bresenham's algorithm computes a line by determining which pixel best approximates the ideal line at each step. Given two points, the algorithm uses the differences

$$dx = |x_1 - x_0|, \quad dy = |y_1 - y_0|$$

and an error term to decide whether to increment the x-coordinate, y-coordinate, or both. This avoids floating-point arithmetic, making it efficient for real-time or low-level rendering tasks. [3]

4.3 Software Structure

The following directory trees define the structure of the newly added landscape module in the ADV-Commons, ADV-Lib and ADV-UI.

4.3.1 ADV-Commons

```
module-landscape/
├── src/
│   └── main/
│       └── java/
│           └── ch/hsr/adv/commons/landscape/logic/
│               ├── domain/
│               │   ├── LandscapeConfigElement.java
│               │   ├── LandscapeElement.java
│               │   ├── LandscapeElementType.java
│               │   ├── PathElement.java
│               │   └── PointElement.java
│               ├── ConstantsLandscape.java
│               └── LandscapeConfig.java
└── build.gradle
```

- `LandscapeConfigElement.java`: Defines the element containing the initial landscape settings.
- `LandscapeElement.java`: Contains the definition of the landscape matrix.
- `PathElement.java`: Represents a path along with its associated color.
- `PointElement.java`: Represents a single point and its color in the landscape.
- `LandscapeElementType.java`: Enum listing the supported landscape element types.
- `ConstantsLandscape.java`: Defines the name of the landscape module.
- `LandscapeConfig.java`: Record used to represent the initial landscape settings such as position, viewing angle and zoom.

4.3.2 ADV-Lib

```
module-landscape/
├── src/
│   ├── main/
│   │   └── java/
│   │       └── ch/hsr/adv/lib/landscape/logic/
│   │           ├── LandscapeBuilder.java
│   │           ├── LandscapeGenerator.java
│   │           ├── LandscapeModule.java
│   │           └── LandscapeStringifyer.java
│   └── test/
└── build.gradle
```

- `LandscapeBuilder.java`: Creates the element that will be sent to the UI.
- `LandscapeGenerator.java`: Generates the matrix used to represent the landscape.
- `LandscapeModule.java`: Provides the module interface for use within coding exercises.
- `LandscapeStringifyer.java`: Converts the element into a JSON-formatted string.

4.3.3 ADV-UI

```
module-landscape/
├── src/
│   ├── main/
│   │   └── java/
│   │       └── ch/hsr/adv/ui/landscape/
│   │           ├── domain/
│   │           │   ├── LandscapeContext.java
│   │           │   └── LandscapePoint.java
│   │           ├── logic/
│   │           │   └── LandscapeParser.java
│   │           └── presentation/
│   │               ├── LandscapeControl.java
│   │               ├── LandscapeLayouter.java
│   │               └── MatrixRefinement.java
│   └── test/
└── build.gradle
```

- `LandscapeContext.java`: Contains all data required for rendering the landscape in the UI.
- `LandscapePoint.java`: Represents a point on the landscape using the appropriate `Color` class expected by JavaFX.
- `LandscapeParser.java`: Parses a JSON-formatted string and reconstructs the corresponding Java classes.
- `LandscapeControl.java`: Configures keybindings and mouse interactions for navigating the landscape.
- `LandscapeLayouter.java`: Generates the landscape and draws points and paths based on the received data.
- `MatrixRefinement.java`: Refines the landscape matrix when it has been created manually.

4.4 Architectural Decisions

4.4.1 Versions Upgrades

The application was last actively developed in 2019. As a result, many of the technologies used at that time are now outdated and no longer supported. This includes Java 11, JavaFX 11 and Gradle 4.10.2.

To ensure long-term compatibility and access to current features, both Java and JavaFX were upgraded to version 21, which is the latest long-term support (LTS) release. Gradle was also updated to version 8.13 to support the newer Java version.

In addition, the dependency injection framework Guice was upgraded to version 5. The testing framework JUnit remains at version 4, as migrating to JUnit 5 would require substantial restructuring of the test suite due to its major architectural changes.

4.4.2 CI/CD Pipeline

The existing application also relied on outdated CI/CD tools. Previously, Travis CI was used to build and test the repositories. If the build was successful, the artifacts were deployed to Bintray, and from there to jCenter and Maven Central. However, Bintray was shut down on May 1, 2021, and jCenter was discontinued on August 15, 2024. [2]

Since deploying artifacts to a central repository is no longer necessary, and the application is used only locally within the context of the algorithms and data structures course, it was decided to discontinue publishing artifacts to a central repository.

The build and deployment process is now handled through GitLab pipelines, which perform testing, generate code coverage reports, run static code analysis and build artifacts. SpotBugs and Codecov were replaced with SonarQube Cloud, which integrates smoothly with GitLab and avoids compatibility issues that had arisen with the updated Gradle version. SonarQube Cloud combines both static analysis and code coverage reporting in a single, well-integrated solution.

4.5 Encountered Problems

4.5.1 Memory Usage

By default, a Java process is allowed to use up to one-quarter of the system memory. Since a 3D element with a landscape of up to 200 by 200 consumes significantly more memory than the previous 2D elements, and because the landscape module requires bindings for controlling the 3D element, there are listeners on the pane that prevent the element from being completely removed from memory after the session is closed. These two factors can cause the ADV-UI to run into an OutOfMemory exception. After discussing the issue with the advisor, it was decided not to invest the time needed to fix the problem with the listeners. Instead, a memory monitor was added to check, before adding or saving a new element, whether the memory usage exceeds 90%. If it exceeds this threshold, an exception is thrown and a notification is displayed in the ADV-UI. This prevents the UI from becoming unresponsive. [5]

Additionally, it was tested and confirmed that the application runs successfully on a laptop with the standard notebook configuration for the OST, which includes 16 GB of system memory. To give an indication of memory usage: storing 100 snapshots of a 200×200 matrix requires approximately 1 GB of RAM, while 100 snapshots of a 100×100 matrix require around 250 MB. The processes of saving and loading can increase memory usage by a factor of two or even three. [6]

The listeners can be easily identified using a tool like VisualVM. The following image shows an example of an object in a heap dump, where a landscape of size 200 by 200 was added and then closed.

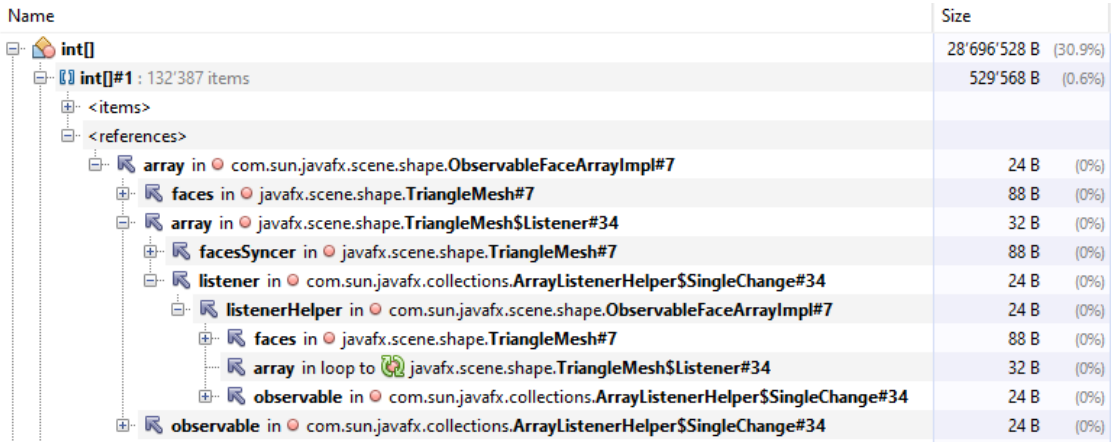


Figure 4.2: Heap dump of object in memory

4.5.2 3D Elements in Virtual Machines

During the testing phase of the Linux build, the application was executed within a virtual machine (VM) running Ubuntu on a Windows 11 host system. It was observed that 3D elements failed to render correctly within this environment, despite 3D graphics acceleration being enabled and all relevant system dependencies installed.

Subsequent troubleshooting indicated that the limitation was inherent to the virtualized environment rather than the operating system itself. This was supported by the fact that in a Windows 11 VM it also did not display the elements correctly. It is important to note that this issue is isolated to 3D functionality, the remainder of the application operates correctly in the virtualized environment.

Following a review with the advisor, it was concluded that no corrective action is necessary, as the software is intended to be deployed on native installations of Linux, Windows, and macOS, platforms on which the 3D rendering operates as expected.

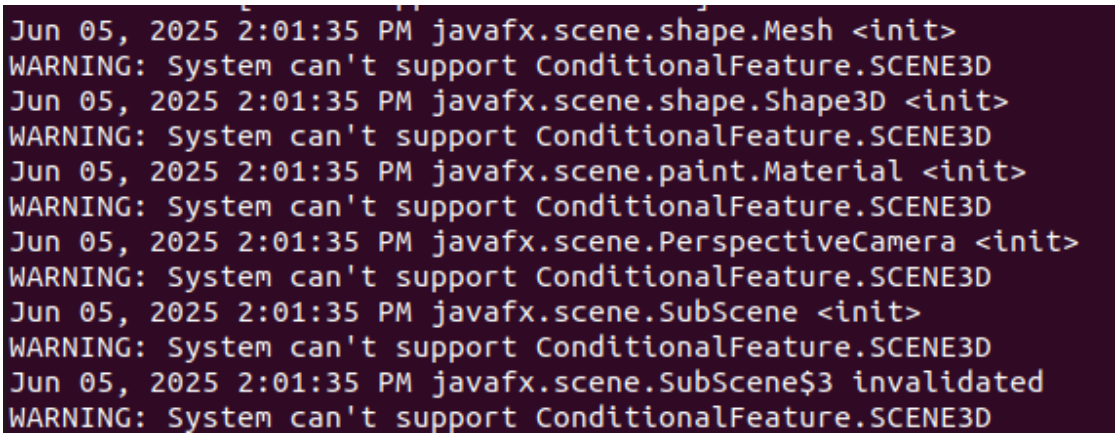


Figure 4.3: Terminal error encountered in a Linux virtual machine

Chapter 5

Quality Measures

5.1 Definition of Done

The criteria below define when a product increment is considered complete, promoting a shared understanding within the team of what constitutes as done [1]:

- The required functionality is fully implemented and working.
- All automated tests have been executed and passed.
- The code meets all quality gate requirements.
- Coding standards and best practices have been followed.
- Documentation is up to date.
- The project plan reflects the latest progress.
- Time tracking records are accurate and complete.

5.2 SonarQube Cloud

SonarQube Cloud is an open-source platform developed by SonarSource that facilitates continuous inspection of code quality. It provides a comprehensive set of static code analysis tools designed to detect bugs, security vulnerabilities and code smells across multiple programming languages. With seamless integration into popular version control systems like GitHub and GitLab, SonarQube Cloud enables automatic analysis and delivers real-time feedback within the development workflow. This capability supports the maintenance of high coding standards, identification of technical debt and enhancement of software maintainability and reliability. Furthermore, SonarQube Cloud generates detailed reports and metrics, enabling the tracking of code quality trends and helping prioritize areas for improvement. [7]

SonarQube Cloud is incorporated into the development workflow for several important reasons.

It facilitates early detection and resolution of code issues during the development cycle, thereby reducing the likelihood of introducing bugs and vulnerabilities into production environments. The platform offers actionable insights and recommendations that guide developers in producing cleaner, more maintainable code.

Integration with the CI/CD pipeline enables automated code analysis and ensures consistent enforcement of quality standards throughout the software delivery process. In addition, SonarQube Cloud provides a variety of metrics, including lines of code, code duplication, complexity and test coverage, which together offer a comprehensive view of the codebase's quality and maintainability.

Analysis of these metrics enables the identification of areas for improvement, facilitates tracking of code quality over time, and supports continuous optimization of development practices.

5.2.1 Quality Gates

The following quality gates have been configured and applied to ADV-Commons, ADV-Lib, and ADV-UI. These gates are enforced only on newly developed modules, as the overall project does not currently meet the required thresholds for code coverage and duplication. These gates are based on default gates from sonar cloud. [8]

ADV-Commons

- Maintainability Rating is A
- Reliability Rating is A
- Security Hotspots Reviewed is 100%
- Security Rating is A

ADV-Lib and ADV-UI

- Coverage is more than 80%
- Duplicated Lines are less than 3%
- Maintainability Rating is A
- Reliability Rating is A
- Security Hotspots Reviewed is 100%
- Security Rating is A

5.3 CI/CD Pipeline

The GitLab CI/CD pipelines consist of the following stages:

- **fetch-jar:** Fetching the ADV-Commons JAR from the package registry (ADV-Lib and ADV-UI)
- **build**: Building the application (ADV-Commons, ADV-Lib and ADV-UI)
- **checkstyle**: Running automated checks to ensure the code adheres to the defined formatting and style guidelines (ADV-Commons, ADV-Lib and ADV-UI)
- **sonarcould**: Performing code quality analysis using SonarQube Cloud (ADV-Commons, ADV-Lib and ADV-UI)
- **publish**: Publishing the JAR and POM files to the package registry (ADV-Commons only)

The pipeline is automatically triggered on every new commit to the main branch and for each merge request, excluding the publish stage in the case of merge requests. It is configured in the `.gitlab-ci.yml` file located at the root of the project directory. The ADV-User_Codebase repository does not have a pipeline, as it contains no test cases, no checkstyle configuration file and does not require SonarQube analysis.

5.4 Test Concept

The test concept outlines the scope, approach, resources and schedule for all testing activities. It is complemented by the test plan, which details the specific components and functionalities to be tested, along with the responsibilities assigned to team members. Together, these elements form the basis for structuring the testing process, efficiently allocating resources and ensuring effective test execution.

5.4.1 Testing Strategy

The testing strategy encompasses the following key steps:

- Identification and definition of test cases
- Execution of manual and automated tests
- Logging and reporting of defects and issues
- Retesting of resolved defects to verify fixes

5.4.2 Test Environment

Mac

- OS: macOS Sequoia Version 15.5
- Java: 21.0.6 temurin
- IDE: IntelliJ IDEA 2025.1.1.1

Linux

- OS: Ubuntu 24.04 LTS
- Java: JDK 21
- IDE: IntelliJ IDEA 2025.1.1.1

Windows

- OS: Windows 11 Version 10.0.26100
- Java: 21.0.7 temurin
- IDE: IntelliJ IDEA 2025.1.1.1

5.4.3 Test Deliverables

The test deliverables are:

- Test Artifacts

5.4.4 Test Schedule

Manual testing will be conducted before each milestone (Release 1, Release 2, Final Product) using the relevant test cases.

Rough Planing

- Week 10: Test Release 1 (no test cases)
- Week 14: Test Release 2 (TC-1 – TC-5)
- Week 16: Test Final Product (all)

Detailed Planning

ADV-UI

The ADV-UI will be tested through unit tests where applicable, as well as through manual testing. The testing strategy includes:

- **Functionality:** Assessed via unit tests where applicable and executed within the CI/CD pipeline.
- **Code Coverage:** Measured using SonarQube Cloud within the CI/CD pipeline.
- **Code Style:** Verified using Checkstyle and SonarQube Cloud checks integrated into the CI/CD pipeline.
- **UI/UX Controls:** Evaluated through manual testing.

The various test cases for the `ADV-UI` are provided in the appendix, see Section 10 Test Cases.

ADV-Lib

The ADV-Lib will be tested primarily through unit tests. The testing strategy includes:

- **Functionality:** Assessed via unit tests executed within the CI/CD pipeline.
- **Code Coverage:** Measured using SonarQube Cloud within the CI/CD pipeline.
- **Code Style:** Enforced through Checkstyle and SonarQube Cloud checks integrated into the CI/CD pipeline.

5.4.5 Test Roles

The following roles were defined as part of the testing process:

- **Lead:** Responsible for designing test cases and verifying the execution results.
- **Tester:** Responsible for executing test cases, reporting bugs, and verifying resolved issues through retesting.

5.4.6 Test Artefacts

The various test artefacts used throughout the project are documented in the appendix, see section 11 Test Artefacts.

Chapter 6

Result

6.1 Functional Requirements

All use cases have been successfully implemented. See section 2.1.2 Use Case Description for detailed descriptions.

UC-1 Calculate Distance and Weight

The landscape module provides default functions to calculate the distance between two points and to determine the corresponding weight. Both are designed to be easily usable in student code. If they do not behave as needed for a specific case, custom implementations can be used instead.

```
double weight = module.calcWeight(p1.x, p1.y, p2.x, p2.y);
double distance = module.calcDist(p1.x, p1.y, p2.x, p2.y);
```

Figure 6.1: Provided Module Functions

UC-2 Create Landscape

When creating a new landscape module, the session name and texture scale are always required. The matrix representing the landscape is also initialized during this process. The texture scale determines the resolution relative to the matrix dimensions. There are two available approaches for initializing the matrix.

The first approach generates the matrix automatically by specifying the width, height, a cliffy value and optionally a seed. The width and height define the matrix dimensions, while the cliffy value controls the frequency and steepness of hills and valleys. Providing a seed ensures that the same landscape is generated every time, which is useful for exercises where all participants should work with an identical landscape.

```
LandscapeModule module1 = new LandscapeModule(
        sessionName: "Genearted with seed",
        textureScale: 3,
        width: 100,
        height: 100,
        cliffy: 0.008,
        seed: 100
);
LandscapeModule module2 = new LandscapeModule(
        sessionName: "Generated without seed",
        textureScale: 3,
        width: 100,
        height: 100,
        cliffy: 0.015
);
```

Figure 6.2: Module Constructors with generated Landscapes

The image below shows the landscape created using the first constructor.

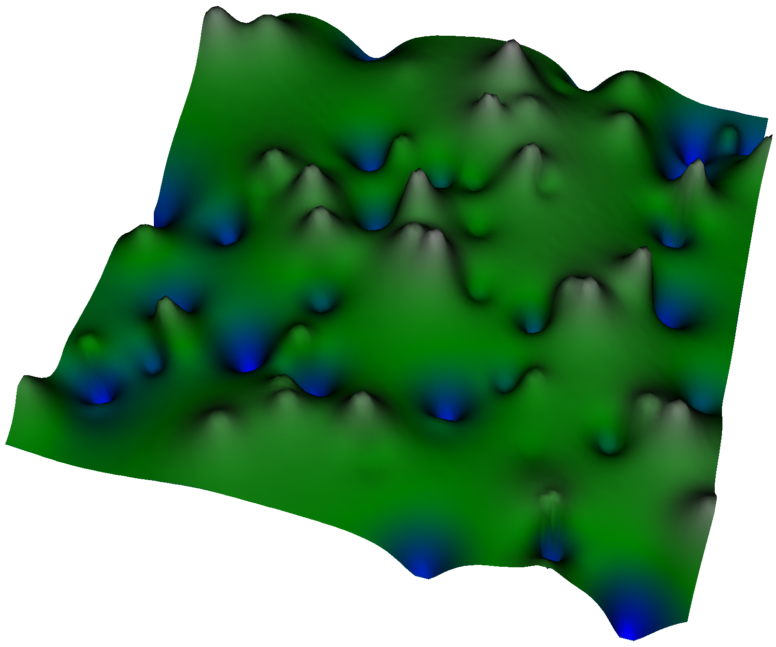


Figure 6.3: Generated Landscape with lower cliffy value

The second constructor results in a generated landscape matrix with a higher cliffy value, producing terrain with steeper slopes and more prominent elevation changes.

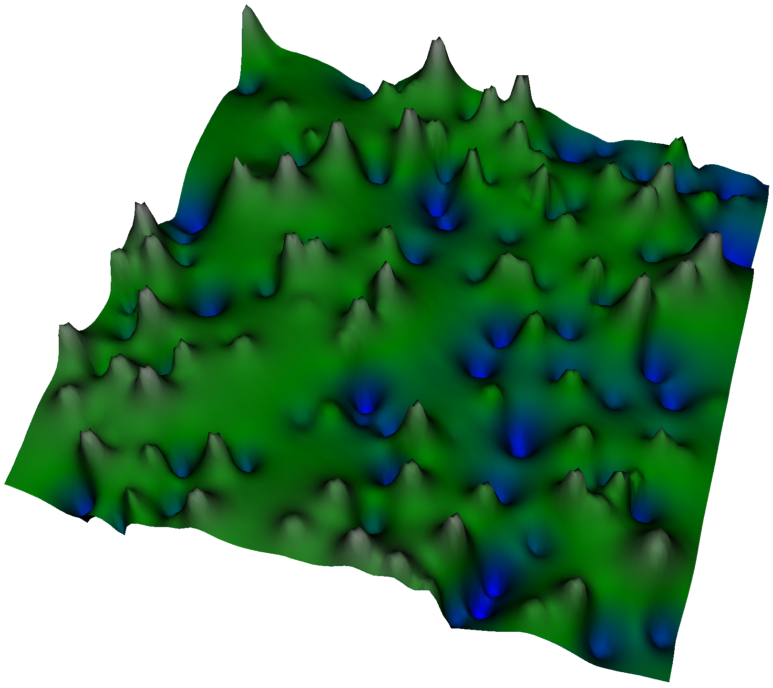


Figure 6.4: Generated Landscape with higher cliffy value

The second approach uses a manually created matrix to generate the landscape.

```
private static final float[][] matrix = {
        {0, 0, 0},          // ---> x / East
        {127, 0, 0},        // |
        {0, 0, 0},          // |
        {0, 127, 127},      // v
        {0, 0, 0},          // y / North
        {0, 127, 0},
};
```

Figure 6.5: Manually created Matrix

```
LandscapeModule module = new LandscapeModule( sessionName: "Manual",  textureScale: 10, matrix);
```

Figure 6.6: Moduel Constructor with provided Matrix

The landscape module keeps track of whether the matrix was generated or provided manually. If it was not generated, the UI automatically applies a refinement step to smooth out hills and valleys for rendering purposes. This refinement affects only the visual representation and has no impact on the algorithms the students implement.

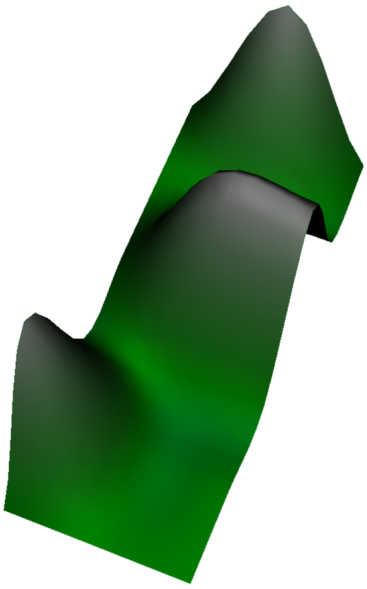


Figure 6.7: Landscape from provided Matrix

UC-3 UI Keybinds and Mouse Bindings

To interact with the generated landscape, various keyboard and mouse controls are available:

- **A:** Move the landscape to the left
- **D:** Move the landscape to the right
- **W:** Move the landscape upward
- **S:** Move the landscape downward
- **I:** Zoom in
- **O:** Zoom out
- **Mouse Wheel:** Zoom in and out
- **Mouse Drag:** Rotate the landscape around its center

UC-4 Draw Point

The landscape module provides a `drawPoint` function that takes the x and y coordinates of a point along with its color When this function is called, the point is added to an internal list that will later be sent to the UI for display.

To trigger the update, the partial snapshot functionality can be used in draw point mode. This sends all collected points from the list to the UI in a single batch. This approach allows efficient batch updates, which is especially helpful for large matrices. For instance, a 100×100 matrix contains 10,000 points. Instead of updating the UI for each individual point, a helper function could update the UI every 50 points visited. This reduces the size of the snapshot and lowers memory usage.

The same approach works with manually created matrices. It is easy to configure how frequently updates are sent, whether for every point, every second point, or based on another rule.

```
private void drawPoint(int x, int y) throws ADVException {
    module.drawPoint(x, y, Color.white);
    if (drawCounter % drawPointBatchSize == 0) {
        ADV.partialSnapshot(module, description: "", PartialSnapshotMode.DRAW_POINT);
    }
    drawCounter++;
}
```

Figure 6.8: Helper Function for Batch Updates

Points are rendered as part of the texture rather than as 3D objects. Below is an example showing how the landscape appears when several points have been drawn.

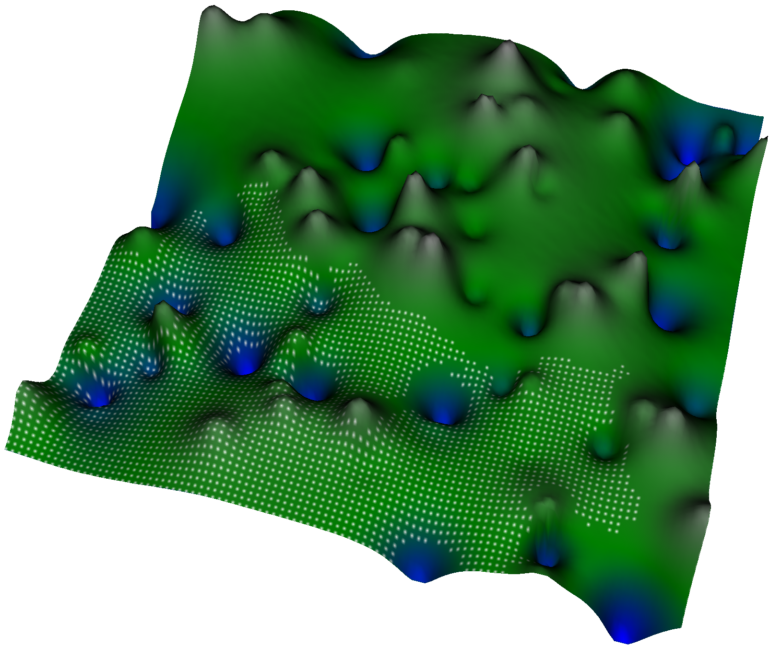


Figure 6.9: Landscape with drawn Points

UC-5 Draw Path

The landscape module includes a `drawPath` function that takes a list of points and a color to represent the path. When this function is called, the path is stored internally. To display it on the landscape, the path must be sent to the UI using the partial snapshot functionality in draw path mode. This sends the list of points to the UI and marks them as a path.

```
List<Point> result = pathFinder.findPath(start, end);
List<Point> path = result != null ? result : Collections.emptyList();

module.drawPath(path, Color.RED);
ADV.partialSnapshot(module,  description: "End", PartialSnapshotMode.DRAW_PATH);
```

Figure 6.10: `drawPath` Usage

The UI then renders the individual points and connects them using the specified color, drawing the path directly onto the texture. Below is an example of how such a path might appear on the landscape.

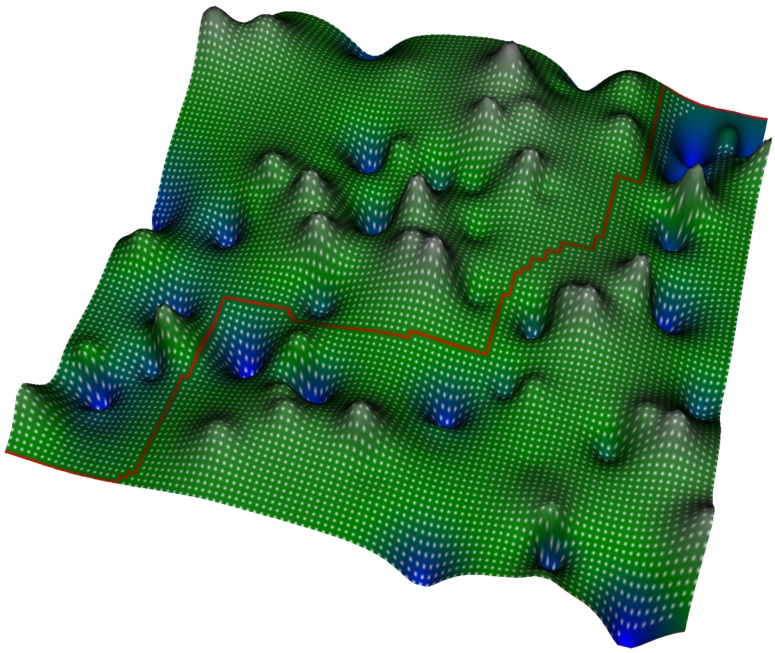


Figure 6.11: Landscape with a drawn Path

UC-6 Set Initial Controls

To configure the initial position, rotation and zoom of the landscape, a `LandscapeConfig` object can be created and passed to the module constructor. This configuration includes five values: the first sets the rotation angle around the x-axis and the second sets the rotation around the y-axis. The third and fourth values control the initial movement along the x-axis (where positive moves the landscape to the right and negative to the left) and the y-axis (positive moves it down and negative moves it up), respectively. The fifth value adjusts the initial zoom by moving along the z-axis, with positive values zooming out and negative values zooming in.

```
LandscapeConfig config = new LandscapeConfig( angleX: 75,  angleY: 20,  moveX: 0,  moveY: 50,  moveZ: 500);
LandscapeModule module = new LandscapeModule( sessionName: "Manual",  textureScale: 10, matrix, config);
```

Figure 6.12: `LandscapeConfig` Usage

These settings only affect the landscape's position when it is first rendered and do not impact the subsequent key or mouse controls used to interact with it.

Optional Improvements

Most of the optional improvements were done as well.

Resolved Issues and Implemented Features

- **Bug:** The Unicode infinity symbol cannot be used.

  This issue could no longer be reproduced after upgrading Java and JavaFX.

- **Bug:** Umlauts are not allowed in session names.

  Like the previous issue, this bug was resolved implicitly through the Java and JavaFX version upgrades.

- **Bug:** Loosing sessions when detached via drag and drop.

  The detachable session feature was disabled to enable proper event forwarding for landscape mouse interaction. This change also naturally resolved the issue of losing sessions when detaching them via drag and drop.

- **Feature:** Add a new replay speed step to loop rapidly through a session.

  Two new replay speed settings were introduced: `faster`, which provides optimal replay speed for landscape module snapshots and `extreme`, which allows for very fast execution as a workaround for the flickering issue.

Figure 6.13: Replay Speed Steps

- **Feature:** Add an `About` page to the application.

  A new button was added to the upper navigation bar in the ADV-UI, opening an `About` page that shows which students contributed to specific features.

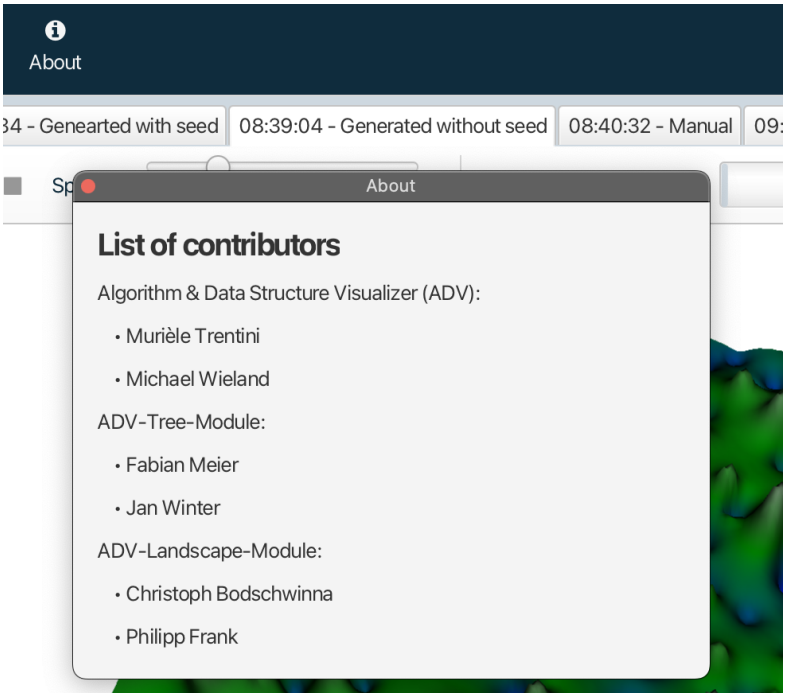


Figure 6.14: About Page

Open Issues

- **Bug:** Flickering occurs when executing a step for the first time in a session.
- **Bug:** Changing the language does not update the labels for the replay speed steps.

6.2 Non-Functional Requirements

The following NFRs were successfully implemented and met:

- **NFR-1**: Tests should cover 80% of the code of the new module.
- **NFR-2**: Maintain a clean, understandable and modular codebase that adheres to standard coding practices.
- **NFR-3**: The system must be able to process a 100x100 matrix and update the display at a rate of at least 10 snapshots per second.

Details of the results can be found in section 2.2 Non-Functional Requirements.

Chapter 7

Conclusion

7.1 Result Reflection

This project aimed to develop a new module for the Algorithm & Data Structure Visualizer (ADV) with support for 3D graphics. It will replace the previous standalone solution built with JOGL and should directly integrate into the existing ADV system. The new module needs to be able to support algorithms such as Dijkstra and A* on 3D terrain.

A key challenge was the need to handle large datasets efficiently. Until now, the ADV had only supported 2D elements and algorithms that typically produced no more than 30 to 40 snapshots. In contrast, the landscape module needed to handle matrices of up to 200 by 200 in size, resulting in 40,000 data points. From the beginning, performance was a central concern.

An earlier semester thesis had already demonstrated that JavaFX could be used to render 3D content within the ADV application. However, the resulting prototype suffered from significant performance issues and was not suitable for practical use. This project aimed to avoid those limitations and deliver a responsive and reliable implementation.

The new landscape module can generate matrices of any size or work with manually defined ones. Like other ADV modules, it uses a snapshot-based communication protocol. To accommodate larger datasets, the protocol was extended to support partial snapshots. This allows the ADV-UI to construct new snapshots incrementally by applying only the differences to the previous state, reducing both data transmission and memory usage. As a result, the complete matrix only needs to be transmitted once at the start. Furthermore, the module also supports batch updates for drawing points, minimizing the number of snapshots and overall memory requirements. These enhancements were essential for achieving a performant and efficient solution.

The module offers both keyboard and mouse controls for intuitive navigation, making it easy for students and instructors to create interactive 3D exercises that integrate seamlessly with the ADV system. Utility functions for calculating distances and weights between points are also included to support algorithm development.

In addition to the new module, the project introduced several general improvements to the ADV application. As the system had not been maintained for several years, many of its technologies were outdated. Java and JavaFX were upgraded from version 11 to version 21, and Gradle was updated to the latest release. Several bugs were fixed, including issues related to special characters such as the infinity symbol and umlauts. The update also added new replay speeds and introduced an About page that displays student contributions to specific features.

7.2 Outlook

Future improvements could include additional major version upgrades. Two key candidates are updating Guice from version 5 to version 7 and migrating JUnit from version 4 to version 5. These upgrades would require structural changes throughout the application, which is why they were not included in this project.

Another useful feature would be enabling snapshot selection via an input field. Currently, users can only navigate snapshots using next and back buttons. With the landscape module, sessions can contain 100 or more snapshots, so being able to jump directly to a specific snapshot by entering its number would improve usability.

The partial snapshot functionality could also be extended to support `ADVRelations`. This would allow other modules that work with graph data to benefit from the same protocol enhancements introduced for the landscape module.

Additionally, the snapshot cleanup process in the landscape module could be improved to ensure all bindings are removed when a snapshot is deleted. This would help fully release the memory used by the module without requiring a complete restart of the ADV application.

Finally, the remaining open issues could be reviewed and resolved in future development efforts.

7.3 Closing Statement

In conclusion, this project achieved its goal and successfully developed a new landscape module for the ADV application, enabling support for 3D graphics. Additionally, the application was upgraded to use modern technologies where possible, several known bugs were resolved and existing feature requests were implemented. The result is a more robust and versatile tool that will hopefully continue to be widely used in future exercises to help students better understand complex algorithms.

Part III

Project Documentation

Chapter 8

Project Plan

8.1 Process

For this project, it was decided to use Scrum+, a hybrid framework that integrates Scrum with the Rational Unified Process (RUP). This approach was chosen for its ability to combine RUP's structured, long-term planning and phased development with Scrum's adaptability and short-term planning. Scrum+ enables teams to maintain a clear focus on overarching project objectives while staying flexible to accommodate evolving requirements.

8.1.1 Iteration

Each iteration will last two weeks, with a new Sprint starting on Tuesday morning. Sprint Planning, Sprint Retrospective, and Sprint Reviews will take place biweekly. Additionally, the refinement meeting will be integrated into the weekly Scrum meeting. Instead of daily Scrum meetings, it has been decided to hold them weekly. The weekly team meeting without the advisor is scheduled for Sundays at 18:00, while the meeting with the advisor will be held on Thursdays at 15:00. Both meetings will generally be conducted online.

8.1.2 Time tracking & Issue management

Time tracking and issue management will be managed using Jira Cloud. All recorded times will be rounded to the closest 15-minute increment.

8.1.3 Roles

The roles and tasks are defined according to Atlassian Scrum roles and the book ‘Clean Agile: Back to Basics‘. [4, 11]

Scrum Master

The Scrum Master is responsible for:

- Organizing Scrum processes and Sprint planning
- Leading Scrum meetings
- Verify meeting protocol
- Assisting the Project Owner with backlog management

This role will be taken on by Philipp Frank.

Project Owner

The Project Owner is responsible for:

- Managing the backlog, including prioritization
- Organizing meetings
- Communicating with the advisor
- Ensuring timely submission of documentation

This role will be taken on by Philipp Frank.

Project Manager

The Project Manager is responsible for:

- Ensuring adherence to established guidelines
- Maintaining a functional development pipeline
- Monitoring and maintaining quality standards

This role will be taken on by Christoph Bodschwinna.

Developer

Developers are responsible for:

- Working on assigned project tasks

Both team members will share this role.

8.2 Guidelines for Code

To ensure consistency with the rest of the application, the coding guidelines are adopted from the previous project and will be enforced using Checkstyle.

8.3 Guidelines for Documentation

The documentation guidelines follow the principles outlined in WRITE THE DOCS. These include using descriptive text for hyperlinks instead of generic phrases like 'Click here'. Examples should be included where relevant. Additionally, the language should be consistent throughout the documentation, maintaining uniform capitalization. Proper indentation should also be applied in LaTeX documentation. [10]

8.4 Phase

The phases are aligned with the milestones.

1. Inception 17.02.2025 - 23.02.2025
2. Elaboration 24.02.2025 - 30.03.2025
3. Construction 31.03.2025 - 01.06.2025
4. Transition 02.06.2025 - 13.06.2025

8.5 Milestones

The milestones are scheduled for Sundays.

M1 Project Setup

Date: 02.03.2025
Objectives:

1. Set up repositories for code and documentation.
2. Define project guidelines.
3. Assign team roles.
4. Identify and evaluate potential risks.
5. Configure Jira Cloud for issue management and time tracking.
6. Develop the initial project plan.

M2 Requirements

Date: 16.03.2025
Objectives:

1. Define the functional requirements through use cases.
2. Establish verifiable non-functional requirements.
3. Conduct the domain analysis.
4. Set up the development environment.

M3 End of Elaboration

Date: 30.03.2025
Objectives:

1. Define the testing strategy.
2. Establish the system architecture.
3. Define quality assurance process.
4. Complete the prototype.

M4 Release 1

Date: 27.04.2025
Objectives:

1. Complete Release 1.
2. Create an Enterprise Architect model of the code for Release 1.

M5 Release 2

Date: 25.05.2025
Objectives:

1. Complete Release 2.
2. Update Enterprise Architect to reflect the current state of the code.

M6 Final Submission

Date: 13.06.2025
Objectives:

1. Complete Final Product.
2. Update Enterprise Architect to reflect the final state of the code.
3. Finalize the project documentation.
4. Complete the A0 poster.
5. Finalize the abstract.

8.6 Planned Product Releases

Prototype

Release Date: 30.03.2025

The prototype aims to help the team become familiar with the application and JavaFX while also serving as a platform for creating smaller proof-of-concept implementations.

Release 1

Release Date: 30.04.2025

Release 1 includes the following features:

- UC-2 Create Landscape

Release 2

Release Date: 25.05.2025

Release 2 builds upon Release 1 and includes:

- UC-3 UI Keybinds and Mouse Bindings
- UC-4 Draw Point
- UC-5 Draw Path

Final Product

Release Date: 13.06.2025

The final product builds upon Release 2 and includes:

- UC-1 Calculate Distance and Weight
- UC-6 Set Initial Controls

8.7 Long-Term Plan / Roadmap

The project timeline spans from February 17 to June 13, 2025, with the final documentation due on the last day.

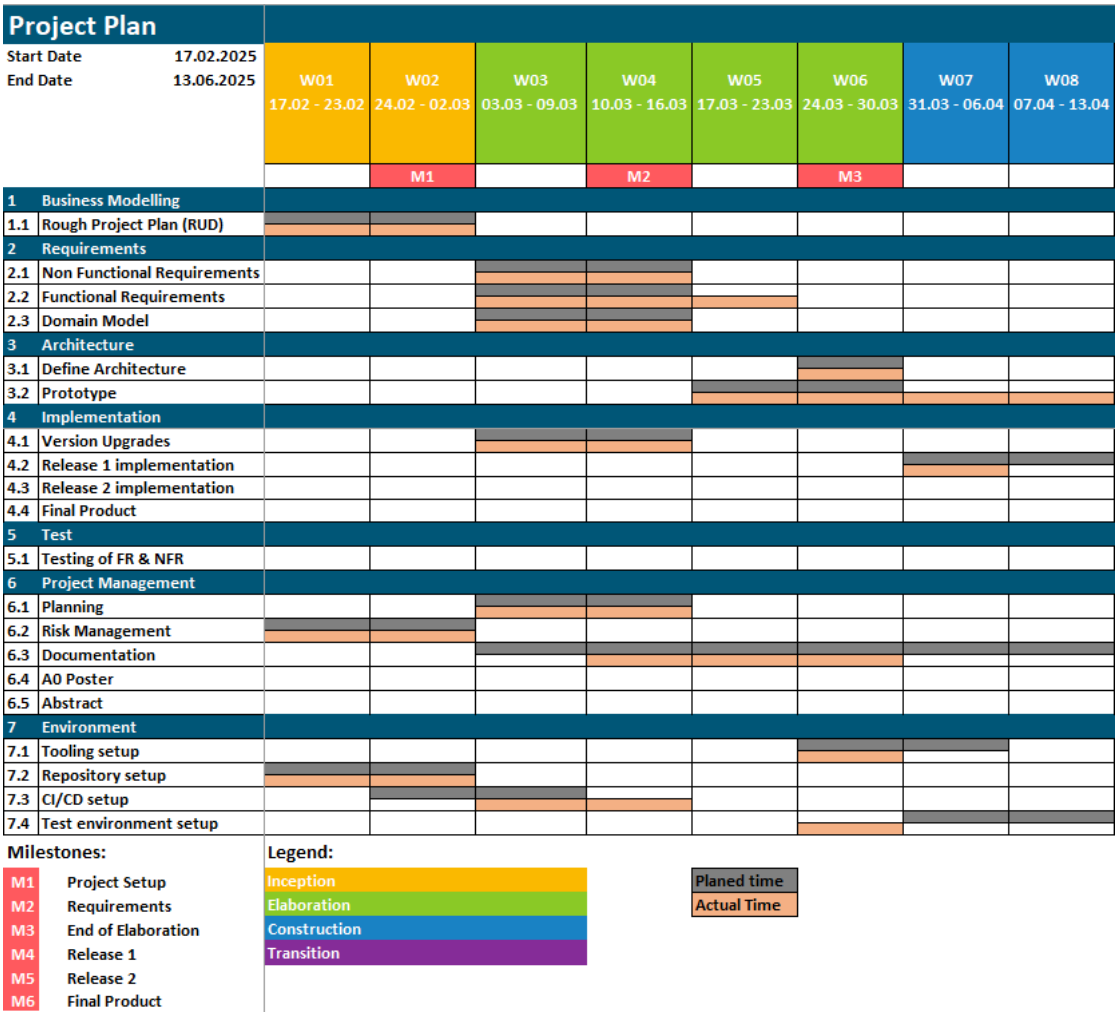


Figure 8.1: Roadmap for W01 to W08

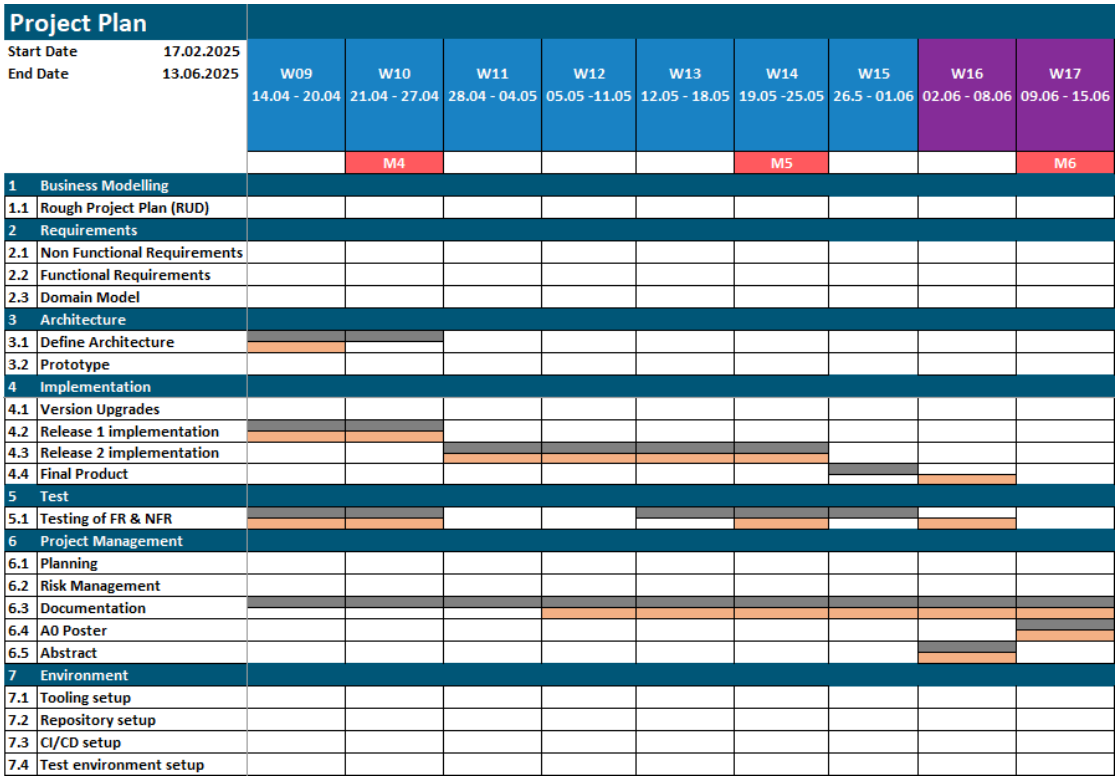


Figure 8.2: Roadmap for W09 to W17

8.8 Risk management

8.8.1 Risk

ID	R1
Risk	Lack of Knowledge
Comment	Limited experience with certain technologies (e.g., JavaFX).
Preventive Action	Begin researching potential challenges early and develop a prototype to gain familiarity.
Corrective Action	Seek assistance from teammates. While tasks are typically handled individually, team members should collaborate and provide support when needed.
Severity	Medium
Probability	High

Table 8.1: Risk Table: Lack of Knowledge

ID	R2
Risk	Unexpected Limitation of Resources
Comment	Unexpected absence of team members (e.g., illness, work emergency) or failure of critical devices (e.g., laptop). The additional effort required should not exceed 8 hours.
Preventive Action	All team members must regularly document and push their progress to ensure continuity.
Corrective Action	Features beyond Release 1 will be dropped or their scope reduced if necessary.
Severity	High
Probability	Low

Table 8.2: Risk Table: Unexpected Limitation of Resources

ID	R3
Risk	Use of Deprecated Functionality
Comment	Deprecated functions or services are being used.
Preventive Action	Develop a prototype to identify deprecated functionality.
Corrective Action	Identify and transition to supported alternatives.
Severity	Medium
Probability	High

Table 8.3: Risk Table: Use of Deprecated Functionality

8.8.2 Risk map

Following are the definitions for the risk map.

Probability

Probability represents the likelihood of the risk occurring during the project, expressed as a percentage.

- Very High: 75%-100%
- High: 50%-75%
- Medium: 30%-50%
- Low: 10%-30%
- Very Low: 1%-10%

Severity

Severity indicates the estimated time required to resolve the damage caused by the occurrence of a risk, measured in hours.

- Very High: 8h+
- High: 4h-8h
- Medium: 2h-4h
- Low: 0.5h-2h
- Very Low: 0h-0.5h

Figure 8.3: Riskmap

Bibliography

[1] Atlassian. What is the Definition of Done? https://www.atlassian.com/agile/project-management/definition-of-done, 2024. Accessed: 2025-03-20.

[2] Stephen Chin. Into the sunset on may 1st: Bintray, gocenter, and chartcenter. https://jfrog.com/blog/into-the-sunset-bintray-jcenter-gocenter-and-chartcenter/. Accessed: 2025-06-06.

[3] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, Kurt Akeley. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2014.

[4] Robert C. Martin. *Clean Agile: Back to Basics*. Pearson, 2019.

[5] Oracle. Memory management. https://docs.oracle.com/en/graalvm/jdk/21/docs/reference-manual/native-image/optimizations-and-performance/MemoryManagement/#overview-1. Accessed: 2025-06-05.

[6] OST. Minimale notebook-konfiguration für standard-anwendungen. https://www.ost.ch/de/die-ost/services/ict/standard-notebook. Accessed: 2025-06-05.

[7] SonarSource. SonarQube Cloud Documentation. https://docs.sonarsource.com/sonarqube-cloud/, 2025. Accessed: 2025-06-07.

[8] SonarSource. SonarQube Cloud Quality gates. https://docs.sonarsource.com/sonarqube-cloud/improving/quality-gates/, 2025. Accessed: 2025-06-07.

[9] Dr. Gernot Starke. arc42 Documentation. https://arc42.org/overview, 2024. Accessed: 2025-03-20.

[10] Write the Docs. Documentation principles. https://www.writethedocs.org/guide/writing/docs-principles/, 2024. Accessed: 2025-03-20.

[11] Dave West. What are the three scrum roles? https://www.atlassian.com/agile/scrum/roles. Accessed: 2025-03-20.

[12] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

List of Figures

List of Tables

List of Resources

Task Area	Tools
Literature Research and Management	ChatGPT, Google
Translation	DeepL, Google Translate, Leo.org
Coding	ChatGPT, IntelliJ IDEA Ultimate, Visual Studio Code
Text Creation, Optimization, Spell and Grammar Check	ChatGPT, DeepL, LanguageTool, IntelliJ TeXiFy-IDEA Plugin
Collaboration and Project Management	GitLab, Jira, Outlook, Teams
DevOps	GitLab, SonarQube Cloud
Code Quality and Static Analysis	Checkstyle, Jacoco, SonarQube Cloud

Table 8.4: List of Tools and Resources

Glossary

A* (A-star) is a pathfinding algorithm that finds the shortest route between two points using a combination of actual distance traveled and estimated distance to the goal.

ADV stands for ‘Algorithm & Data Structure Visualizer‘.

Bintray was a platform for hosting and distributing software packages and binaries. It was commonly used to publish artifacts for Java projects before being sunset by JFrog in 2021.

Dijkstra ’s algorithm is a graph traversal method used to find the shortest path between nodes. Unlike A*, it does not use heuristics and guarantees the shortest path in weighted graphs.

Enterprise Architect is a modeling and design tool used for creating UML diagrams and software architecture documentation.

Gradle is a build automation tool used for managing dependencies, compiling code, and packaging software, primarily in Java-based projects.

Guice is a lightweight dependency injection framework for Java, developed by Google.

JavaFX is a framework for building rich desktop applications in Java. It supports modern UI components, styling and 3D graphics, making it suitable for visually complex applications.

JCenter was a popular artifact repository for Java and Android libraries, hosted by Bintray. It was deprecated in 2021, and developers have since migrated to alternatives like Maven Central.

JOGL (Java OpenGL) is a wrapper library that allows OpenGL functionality to be used in Java applications.

Maven Central is the primary repository for open-source Java libraries and dependencies.

Snapshots are data packages that captures the current state of one or more ADV modules. They are used to transfer relevant data to the UI for visualization.

Sockets are low-level network interfaces used for communication between two systems over a network. They allow data to be sent and received through established connections, such as client-server communication.

Part IV

Appendix

Chapter 9

Example Snapshot

The following figure shows how the JSON snapshot of the graph module is rendered in the ADV-UI.

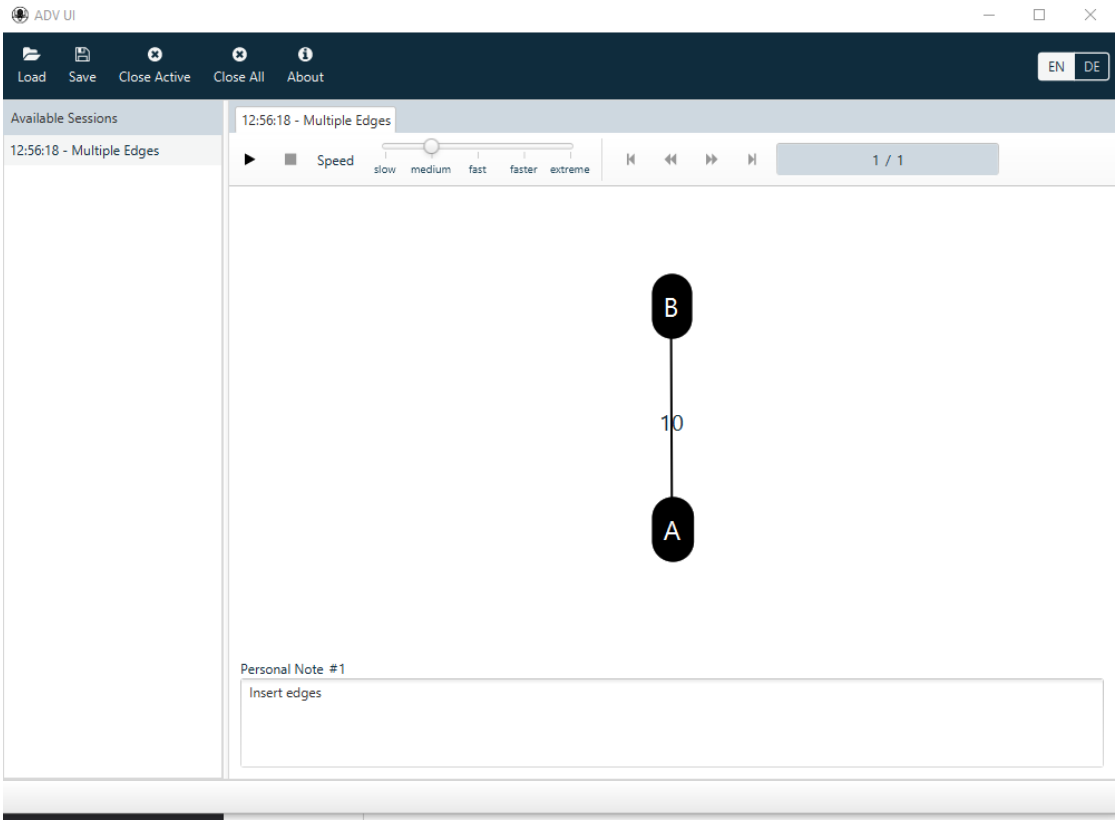


Figure 9.1: Visual representation of the snapshot in the ADV-UI

Below is the JSON data that produces this visual output:

```
{
  "snapshotId": 1,
  "snapshotDescription": "Insert edges",
  "moduleGroups": [
    {
      "moduleName": "graph",
      "elements": [
        {
          "id": 1,
          "fixedPosX": 200,
          "fixedPosY": 200,
          "content": "A"
        },
        {
          "id": 2,
          "fixedPosX": 200,
          "fixedPosY": 100,
          "content": "B"
        }
      ],
      "relations": [
        {
          "isDirected": false,
          "sourceElementId": 1,
          "targetElementId": 2,
          "label": "10"
        }
      ],
      "flags": [],
      "metaData": {},
      "position": "DEFAULT"
    }
  ]
}
```

Chapter 10

Test Cases

ID	TC-1
Lead	Philipp Frank
Title	Keyboard Controls
Precondition	The ADV-UI is running with a landscape module session.
Steps	• Press the A key to move the landscape left • Press the D key to move the landscape right • Press the W key to move the landscape up • Press the S key to move the landscape down • Press the I key to zoom in • Press the O key to zoom out
Expected Result	All key controls work.

ID	TC-2
Lead	Philipp Frank
Title	Mouse Controls
Precondition	The ADV-UI is running with a landscape module session.
Steps	• Scroll the mouse wheel to zoom in and out • Drag with the mouse to rotate the landscape around its center
Expected Result	All mouse controls work.

ID	TC-3
Lead	Philipp Frank
Title	Draw Point
Precondition	The ADV-UI is running with a landscape module session.
Steps	• Click through the first three snapshots
Expected Result	The points are rendered at the correct positions and in the specified color.

ID	TC-4
Lead	Philipp Frank
Title	Draw Path
Precondition	The ADV-UI is running with a landscape module session.
Steps	• Go to the last snapshot
Expected Result	The path is rendered at the correct position and in the specified color.

ID	TC-5
Lead	Philipp Frank
Title	About Page
Precondition	The ADV-UI is running.
Steps	• Click the ‘About‘ button
Expected Result	- The About page opens and displays the contributions.

ID	TC-6
Lead	Philipp Frank
Title	ADV-UI JAR execution
Precondition	The ADV-UI JAR is build.
Steps	• Navigate to the directory where the ADV-UI JAR was built • Start the application using the following command: `java -jar adv-ui-3.0.jar`
Expected Result	The ADV-UI starts successfully.

Chapter 11

Test Artefacts

Test Artefacts for Release 2

Mac

ID	TC-1
Date	25.05.2025
Tester	Philipp
Result	Passed

ID	TC-2
Date	25.05.2025
Tester	Philipp
Result	Passed

ID	TC-3
Date	25.05.2025
Tester	Philipp
Result	Passed

ID	TC-4
Date	25.05.2025
Tester	Philipp
Result	Passed

ID	TC-5
Date	25.05.2025
Tester	Philipp
Result	Passed

Linux

ID	TC-1
Date	25.05.2025
Tester	Christoph
Result	Passed

ID	TC-2
Date	25.05.2025
Tester	Christoph
Result	Passed

ID	TC-3
Date	25.05.2025
Tester	Christoph
Result	Passed

ID	TC-4
Date	25.05.2025
Tester	Christoph
Result	Passed

ID	TC-5
Date	25.05.2025
Tester	Christoph
Result	Passed

Windows

ID	TC-1
Date	25.05.2025
Tester	Christoph
Result	Passed

ID	TC-2
Date	25.05.2025
Tester	Christoph
Result	Passed

ID	TC-3
Date	25.05.2025
Tester	Christoph
Result	Passed

ID	TC-4
Date	25.05.2025
Tester	Christoph
Result	Passed

ID	TC-5
Date	25.05.2025
Tester	Christoph
Result	Passed

Test Artefacts for Final Product

Mac

ID	TC-1
Date	08.06.2025
Tester	Philipp
Result	Passed

ID	TC-2
Date	08.06.2025
Tester	Philipp
Result	Passed

ID	TC-3
Date	08.06.2025
Tester	Philipp
Result	Passed

ID	TC-4
Date	08.06.2025
Tester	Philipp
Result	Passed

ID	TC-5
Date	08.06.2025
Tester	Philipp
Result	Passed

ID	TC-6
Date	08.06.2025
Tester	Philipp
Result	Passed

Linux

ID	TC-1
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-2
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-3
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-4
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-5
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-6
Date	08.06.2025
Tester	Christoph
Result	Passed

Windows

ID	TC-1
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-2
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-3
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-4
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-5
Date	08.06.2025
Tester	Christoph
Result	Passed

ID	TC-6
Date	08.06.2025
Tester	Christoph
Result	Passed