Quality Assurance Concept – *Think Outside the Room*

Team Collaboration

Our team organizes the project through a combination of regular in-person meetings and online coordination via our Discord channel. We meet every week to discuss progress, resolve open issues, and plan the next development steps. Task assignments are not fixed from the beginning—instead, we distribute responsibilities after each milestone, based on the current project state and the needs of the next phase.
This flexible approach helps us adapt quickly to changes while ensuring balanced participation from all team members. Continuous communication is key to our workflow: we stay in touch throughout the week using Discord, where we hold online meetings, share updates, ask questions, and collaborate in real time. During both in-person and online sessions, we also present our completed work to receive feedback and improve the overall quality of the code.

Version Control

For version control, we use GitHub. Our team follows a clear branching strategy to ensure the stability of the main branch and the traceability of all development efforts. The `main` branch contains only stable and tested code. Each team member creates their own feature branch based on the latest state of `main`, and commits changes frequently with descriptive commit messages.
Merging into the main branch is only permitted after the following criteria are fulfilled:

- The code has been tested locally.
- A code review has been completed by at least one other team member.
- The merge is discussed and approved by the group.

In order to maintain traceability and transparency, we follow a commit message convention that includes meaningful summaries of the changes.

Code Review

All changes to the main branch go through a peer review process. Each pull request must be reviewed and approved by at least one team member. The reviewer focuses on:

- Code readability and structure,
- Correct use of exception handling,
- Logical consistency and test coverage,
- Adherence to project coding standards.

In some cases, pair programming is used especially for complex parts of the project such as the GUI or networking logic. This improves code quality and ensures that more than one team member is familiar with key components of the codebase.

Coding Standards

To ensure readability, maintainability, and consistency in our codebase, we follow standard Java conventions throughout the project. We have agreed on a common style guide within

the team, including rules for naming, indentation, and structure. Our IDEs are configured to automatically format code on save, which helps us maintain clean code with minimal manual formatting effort.
We use JavaDoc for method and class documentation to clearly explain functionality.
We also maintain a comprehensive `README.md` file at the root of the repository.

Tools

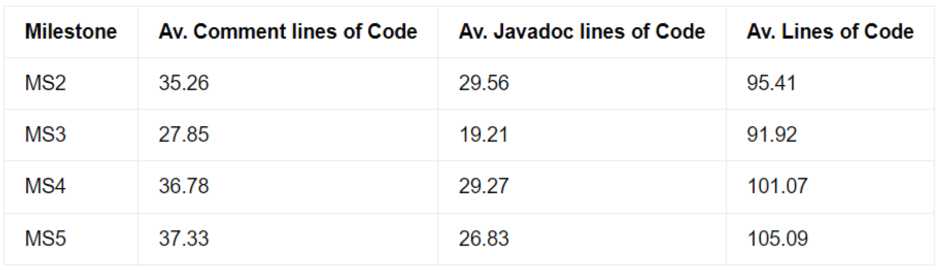
The analysis was conducted using the MetricsReloaded plugin integrated into IntelliJ IDEA. This tool provides a comprehensive set of static code analysis metrics, including cognitive complexity, cyclomatic complexity, design complexity, and code documentation metrics. In addition, JaCoCo was used to measure the code coverage of our test suite. These tools allowed us to evaluate both the evolution of code quality and the effectiveness of our test coverage across project milestones.

Measurements

For our project, we measured the following metrics:

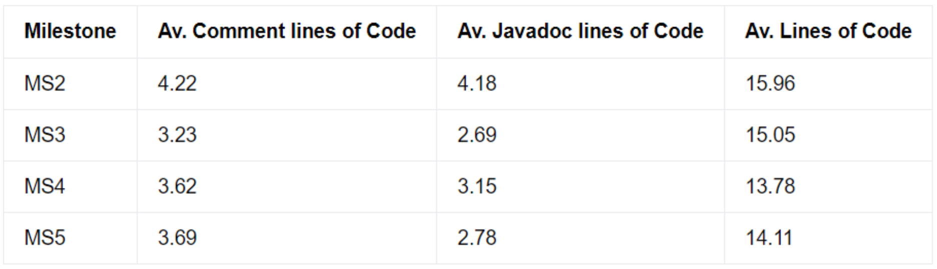
Number of code lines, JavaDoc lines, and comments per class/method

Class:



Milestone	Av. Comment lines of Code	Av. Javadoc lines of Code	Av. Lines of Code
MS2	35.26	29.56	95.41
MS3	27.85	19.21	91.92
MS4	36.78	29.27	101.07
MS5	37.33	26.83	105.09

Method:



Milestone	Av. Comment lines of Code	Av. Javadoc lines of Code	Av. Lines of Code
MS2	4.22	4.18	15.96
MS3	3.23	2.69	15.05
MS4	3.62	3.15	13.78
MS5	3.69	2.78	14.11

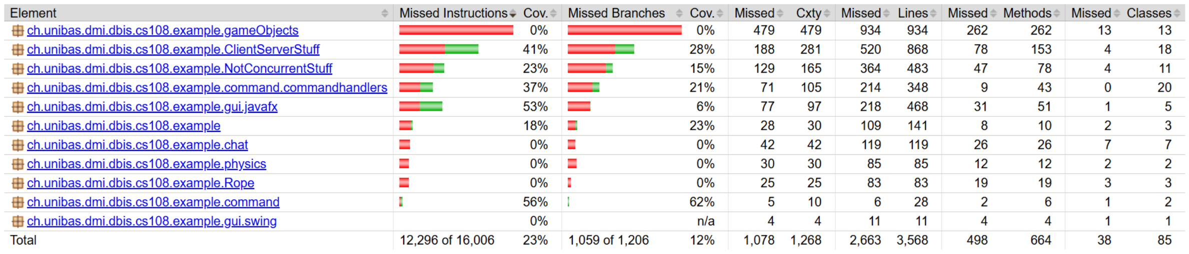
Complexity

Milestone	Cognitive (Total / Avg.)	Cyclomatic (Total / Avg.)	Design (Total / Avg.)
MS2	417 / 2.37	424 / 2.48	352 / 2.06
MS3	1,185 / 2.82	1,046 / 2.52	903 / 2.18
MS4	1,631 / 2.22	1497 / 2.06	1,324 / 1.82
MS5	1,736 / 2.25	1,602 / 2.09	1,372 / 1.79

Code Coverage

Milestone 3:

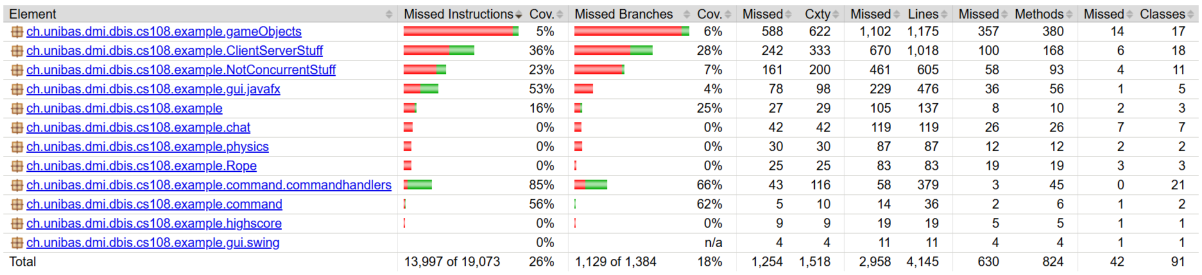
Think_Outside_The_Room



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ch.unibas.dmi.dbis.cs108.example.gameObjects		0%		0%	479	479	934	934	262	262	13	13
ch.unibas.dmi.dbis.cs108.example.ClientServerStuff		41%		28%	188	281	520	868	78	153	4	18
ch.unibas.dmi.dbis.cs108.example.NotConcurrentStuff		23%		15%	129	165	364	483	47	78	4	11
ch.unibas.dmi.dbis.cs108.example.command.commandhandlers		37%		21%	71	105	214	348	9	43	0	20
ch.unibas.dmi.dbis.cs108.example.gui.javafx		53%		6%	77	97	218	468	31	51	1	5
ch.unibas.dmi.dbis.cs108.example		18%		23%	28	30	109	141	8	10	2	3
ch.unibas.dmi.dbis.cs108.example.chat		0%		0%	42	42	119	119	26	26	7	7
ch.unibas.dmi.dbis.cs108.example.physics		0%		0%	30	30	85	85	12	12	2	2
ch.unibas.dmi.dbis.cs108.example.Rope		0%		0%	25	25	83	83	19	19	3	3
ch.unibas.dmi.dbis.cs108.example.command		56%		62%	5	10	6	28	2	6	1	2
ch.unibas.dmi.dbis.cs108.example.gui.swing		0%		n/a	4	4	11	11	4	4	1	1
Total	12,296 of 16,006	23%	1,059 of 1,206	12%	1,078	1,268	2,663	3,568	498	664	38	85

Milestone 4:

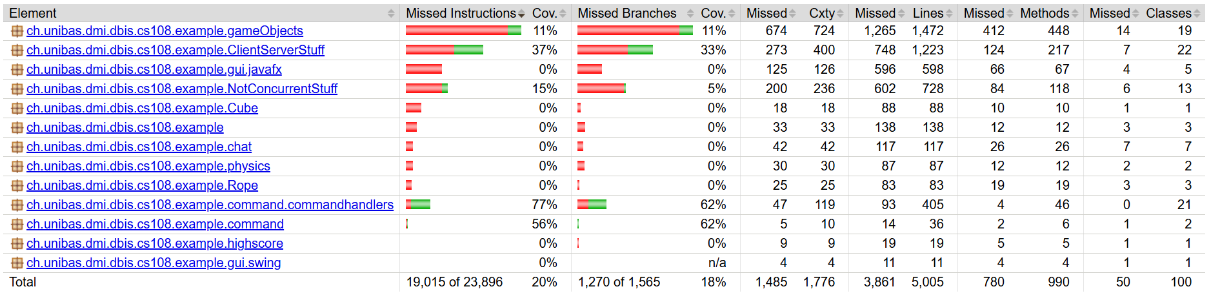
Think_Outside_The_Room



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ch.unibas.dmi.dbis.cs108.example.gameObjects		5%		6%	588	622	1,102	1,175	357	380	14	17
ch.unibas.dmi.dbis.cs108.example.ClientServerStuff		36%		28%	242	333	670	1,018	100	168	6	18
ch.unibas.dmi.dbis.cs108.example.NotConcurrentStuff		23%		7%	161	200	461	605	58	93	4	11
ch.unibas.dmi.dbis.cs108.example.gui.javafx		53%		4%	78	98	229	476	36	56	1	5
ch.unibas.dmi.dbis.cs108.example		16%		25%	27	29	105	137	8	10	2	3
ch.unibas.dmi.dbis.cs108.example.chat		0%		0%	42	42	119	119	26	26	7	7
ch.unibas.dmi.dbis.cs108.example.physics		0%		0%	30	30	87	87	12	12	2	2
ch.unibas.dmi.dbis.cs108.example.Rope		0%		0%	25	25	83	83	19	19	3	3
ch.unibas.dmi.dbis.cs108.example.command.commandhandlers		85%		66%	43	116	58	379	3	45	0	21
ch.unibas.dmi.dbis.cs108.example.command		56%		62%	5	10	14	36	2	6	1	2
ch.unibas.dmi.dbis.cs108.example.highscore		0%		0%	9	9	19	19	5	5	1	1
ch.unibas.dmi.dbis.cs108.example.gui.swing		0%		n/a	4	4	11	11	4	4	1	1
Total	13,997 of 19,073	26%	1,129 of 1,384	18%	1,254	1,518	2,958	4,145	630	824	42	91

Milestone 5:

Think_Outside_The_Room



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ch.unibas.dmi.dbis.cs108.example.gameObjects		11%		11%	674	724	1,265	1,472	412	448	14	19
ch.unibas.dmi.dbis.cs108.example.ClientServerStuff		37%		33%	273	400	748	1,223	124	217	7	22
ch.unibas.dmi.dbis.cs108.example.gui.javafx		0%		0%	125	126	596	598	66	67	4	5
ch.unibas.dmi.dbis.cs108.example.NotConcurrentStuff		15%		5%	200	236	602	728	84	118	6	13
ch.unibas.dmi.dbis.cs108.example.Cube		0%		0%	18	18	88	88	10	10	1	1
ch.unibas.dmi.dbis.cs108.example		0%		0%	33	33	138	138	12	12	3	3
ch.unibas.dmi.dbis.cs108.example.chat		0%		0%	42	42	117	117	26	26	7	7
ch.unibas.dmi.dbis.cs108.example.physics		0%		0%	30	30	87	87	12	12	2	2
ch.unibas.dmi.dbis.cs108.example.Rope		0%		0%	25	25	83	83	19	19	3	3
ch.unibas.dmi.dbis.cs108.example.command.commandhandlers		77%		62%	47	119	93	405	4	46	0	21
ch.unibas.dmi.dbis.cs108.example.command		56%		62%	5	10	14	36	2	6	1	2
ch.unibas.dmi.dbis.cs108.example.highscore		0%		0%	9	9	19	19	5	5	1	1
ch.unibas.dmi.dbis.cs108.example.gui.swing		0%		n/a	4	4	11	11	4	4	1	1
Total	19,015 of 23,896	20%	1,270 of 1,565	18%	1,485	1,776	3,861	5,005	780	990	50	100

Discussion

Analysis of Code Size

Our review of the code base across all milestones shows that the balance between implementation and documentation was generally well maintained. The number of lines per class gradually increased from 95.41 in Milestone 2 to 105.09 in Milestone 5. This steady growth suggests that more functionality was added over time, leading to larger classes.

Interestingly, the size of individual methods remained relatively small and stable, with only slight variations—starting from 15.96 lines on average in MS2 to 14.11 lines in MS5. This indicates that we managed to keep methods compact and focused, which is a positive aspect of our coding style.

However, the comment and JavaDoc coverage at the method level decreased slightly as the project progressed. While class-level documentation remained relatively strong, we observed that some methods lacked detailed explanations, especially in later milestones. This might make future maintenance and onboarding of new team members more challenging.

Lesson Learned

While our architectural decisions provided a solid base and supported feature development effectively, we noticed that some classes, particularly in later stages, became too large and took on too many responsibilities. This could reduce maintainability and increase complexity in the long run.

In future projects, we will focus on distributing functionality more evenly across smaller, dedicated classes. Additionally, we aim to ensure consistent documentation at both class and method levels throughout the entire development process. This should help us maintain high code quality and improve long-term maintainability.

Analysis of Complexity

The complexity evaluation shows that while the total cognitive, cyclomatic, and design complexity increased as the project grew, the average complexity per class and method remained stable. This suggests that despite the increasing size and functionality, the code stayed manageable and understandable. Especially in later stages, the design complexity slightly decreased, showing improvement in structuring the architecture.

Lesson Learned

We learned that keeping methods small and focused helped us control complexity effectively. For future projects, we aim to continue improving by splitting large classes earlier and regularly reviewing complexity to keep the system simple, maintainable, and easy to extend.

Analysis of Code Coverage

Code coverage analysis shows minor changes across the last three milestones. While coverage slightly improved from 23% to 26% in Milestone 4, it dropped back to 20% in Milestone 5, with branch coverage staying around 18%. Some parts like the command handlers and JavaFX were well covered, but core areas such as gameObjects and ClientServerStuff remained mostly untested. This shows that our testing was not consistently applied to all important parts of the system.

Lesson Learned

We learned that increasing coverage requires not just writing more tests, but also ensuring that they target the most critical parts of the system. In future projects, we plan to define clearer testing priorities early on, extend coverage to neglected areas, and integrate regular coverage reviews into our development process to ensure that key features are properly tested and validated.