**Oblig 1**

**Game concept**

**General gameplay**

The game is based on the Pokémon games. The player goes around meeting monsters called Kaijumon. Upon meeting a Kaijumon, the player may either defeat it using their own Kaijumon, or they may become friends with it. After befriending the player, the Kaijumon will follow them and aid them in battle.

Wild Kaijumon are encountered in the wilderness. The player may also battle other human characters who have Kaijumon of their own.

**Progression**

Multiple human characters with unusually strong Kaijumon are spread across the map. Defeat them all to become the best trainer.

Kaijumon may level up upon defeating many enemies. This increases stats such as strength, defence, etc. Most Kaijumon evolve at certain levels, meaning that their appearance changes.

**Online**

A player may connect to another player and fight them. Multiple players can play together on the same map, either aiding or battling each other.

**Development methodology**

**Continuous deployment**

The team will use agile methodologies as described in the course literature, focusing on releasing small features as they are implemented.

The team will use GitLab Issues as a Kanban board, where board items are connected to GitLab Milestones with more detailed user stories.

A sprint is usually 1-2 weeks, depending on the complexity of the tasks.

**Pair programming**

The team will usually assign tasks to groups of 2-3 people, such that no developer is left alone to solve a task. These groups will regularly practice pair programming.

Tasks and groups are assigned at the start of every sprint.

**Communication**

The team communicates via a Discord server. All merge requests are posted there.

**Git workflow**

A git branch is created for every task. A merge request is created upon completion. It is recommended to merge the main branch into your branch before submitting a merge request, as it is easier to deal with merge conflicts in your IDE.

**Meetings**

A meeting is held at the start of every sprint. New features are presented by their creators, and new tasks are assigned to new groups.

**Minimum viable product**

A minimum viable product consists of:

* An area where the player can battle and befriend Kaijumons.
* NPCs that the player can battle.
* One more challenging NPC that functions like a boss battle.

**Summary**

Teams of 3 people sometimes led to the third person being passive, as it was hard to find a time when all team members were available for pair programming.

We will experiment with doing more independent work instead of always pair-programming. Always ask for help if that's needed. We need to make more descriptive commit-messages when you commit something to git.

We need to avoid having code that is stuck locally. Remember to commit and push often. It is OK to have commits where a feature is not fully implemented.

**Oblig 2**

**Project report**

**Team roles**

All team members currently are developers, with Benjamin Godø Mulelid as team lead, and Fredrik Limi Ballestad as secretary. We think that adding more roles at this point adds needless complexity to the project, and we do not have enough capacity to have different departments such as "development", "QA" and "customer service". The team chose to not have a designated customer contact role. The team lead will function as customer contact if that becomes relevant.

We thought about having separate QA people, but this is difficult as the course requires all team members to write code. Instead, every developer is responsible for testing their own code, and the code is also reviewed before being merged into the main branch.

Our current roles are the absolute minimum required to make the team work. The team lead is responsible for setting up meetings, and the secretary records important information from them. Other than that, all members are developers, meaning that every team member writes, tests, and commits code.

We will not switch team lead or secretary yet, but we may switch at a later stage if the roles are not properly fulfilled, or if we want to switch things up out of curiosity.

In summary, we like being flexible rather than having fixed roles for everyone. For the moment, we do not intend to change anything about this.

**Team dynamics and communication**

We have had good experiences with creating groups of two people that work together on solving a task. Some groups have gotten tasks that were noticeably larger than those of other groups, and we will try to break those larger tasks down going forward. There are no disagreements in the team yet. Discussions regarding both team coordination and development have been friendly and productive.

Communication via Discord works well. We have sometimes forgotten to update the "Issues" board in GitLab, which has left some confusion regarding who solved which task. User stories have not been sufficiently descriptive, which makes requested features ambiguous and hard to implement.

**Retrospective**

The team has implemented new features consistently, and everyone has had something to work on at all times. We want to shorten our sprints to one week rather than the two weeks we have been used to. This is because the release candidate will have to be ready in about a month.

Some features were harder to implement, which is why some features (like saving and drawing the UI) have fewer commits in a longer timespan. This explains some unevenness in the commits. Team members also have different approaches to committing, as some only commits after finishing a feature while others commit more frequently during development. Lastly, pair programming leads to commits being unbalanced, as the work of two people are committed by one of them. We have included a list of participants in affected merge requests to mitigate this issue.

Areas that we will try to improve are:

* Making more and smaller issues on the "Issues" board.
* Keep the "Issues" board updated and create more descriptive user stories.
* Announce clearly who is working on what for every sprint.

**Requirements and specification**

**Updated user stories**

See milestones on GitLab for thorough descriptions of core game features: <https://git.app.uib.no/malicious-malware/kaijumon/-/milestones>

**Changes**

Since oblig 1, we have made the following changes:

* Added a proper map using a tile set.
* Added collisions and boundaries to the map.

Other features that are under development are

* Creating a battle system with a separate battle screen.
* Creating NPCs that walk around the map.
* Switching between world view and battle view.

Our sprints start and end on Tuesdays, which is why most features are still in development. We have also had great difficulty with implementing the Kaijumon model and battle system in a simple manner, which is why this feature is still under development.

See <https://git.app.uib.no/malicious-malware/kaijumon/-/boards> for an overview of all features in development.

**Prioritizing tasks**

We have prioritized the core game mechanics and visuals:

* Rendering a tile map and a player.
* NPCs (no interaction for now)
* Player movement and interaction with the world.
* Battle system.
* Mechanics for switching between world view and battle view.
* Start and pause menu.

Combining these features results in a simple game without any progression, but we think that adding progression is easy after the core features are done. Features are implemented in a logical order. Rendering the player and tile map is implemented before implementing player movement, for instance.

After finishing the battle system and world view, we will start adding NPCs which can be battled. These battles will have different difficulties, and they will force the player to become better. All of these mechanics depend on the core mechanics, which is why we will wait until the base game is completed.

**Current state and MVP**

The MVP was described as follows:

* An area where the player can battle and befriend Kaijumons.
* NPCs that the player can battle.
* One more challenging NPC that functions like a boss battle.

The game now consists of an area where the player can encounter Kaijumons and NPCs. It is not yet possible to befriend Kaijumons. The player does not have an inventory to track its Kaijumons.

A boss battle has not yet been added, but that is easy to add after implementing the base game mechanics.

We will try to keep the MVP as it is.

**Product and code**

**Known bugs**

* the camera can be partially outside the map if a player spawns close enough to the border of the map.
* the movement of the player sprite can somtimes be a bit stuttery.
* some directional input keys are prioritized over others.
* there is a tile in front of the stairs north of the spawn which blocks the player from entering even though it should be clear. it is not blocked in the tiled program.

**Fixes**

* fixed a camera bug where the camera could move outside the map.
  + still need to fix the bug where the spawns partially outside the map
* fixed a bug where the player could walk outside the map

**Class diagram**

See the separate file in the "doc" folder.

**Manual tests**

* Save game test:
  + start the game and click new game.
  + move away from the spawn location.
  + Press the "s" key to save the game.
  + exit the game, and then start it again.
  + click "load game" and verify that the spawn is where you were when you saved the game.
* New game test:
  + after the save game test, make sure that starting a new game will spawn you at the original spawn (by the beach) and not the saved game spawn.
* Test that you can walk behind objects.
  + walk to any building or tree and see if you can walk behind it such that the graphics appear in front of the player sprite.
* Test that the "quit game" button in the main menu works.
* Camera test:
  + check that the camera will not move beyond the game map.
  + first move towards the middle of the map such that the camera only sees the map, then walk towards any edge and check that the camera stops. (this test does not include the case where the camera spawned with some of it outside the map)

**Oblig 3**

**Project report**

**Team roles**

We will continue with the same roles as before, which is:

* Team lead: Benjamin Godø Mulelid
* Secretary: Fredrik Limi Ballestad
* Developers: Everyone

The team is happy with the current roles, and we find that our roles enable us to develop efficiently. Role descriptions are the same as before, with the team lead setting up meetings as well as functioning as customer contact, while the secretary records important information from meetings.

**Team dynamics and communication**

We do no longer strictly follow the rule of having two people work on each task. Some smaller tasks or tasks that required a lot of research were impractical to cooperate on, such as configuring Gradle settings or setting up the initial battle system. In cases where only one developer has implemented a feature, that developer has been asked to present their work on the following meeting, keeping the team up to date.

We have started tagging bugs with the "bug" label in GitLab, and we try to handle them first. Discord works fine for coordinating meetings and discussing implementation details.

**Retrospective**

We have implemented many new features, and we are now closing in on the MVP. Our new features are listed under "changes" below.

We have tried to fix the unevenness in commits by letting the people with fewer total commits be the ones that commit when pair programming. This will hopefully result in an even distribution of commits for oblig 4.

Areas that we will try to improve are:

* Distributing commits more evenly.
* Prioritize fixing bugs before developing new features.
* Adding documentation for all public methods, interfaces and classes.
* Keeping the documentation updated when modifying the code.

**Requirements and specification**

**User stories**

See milestones on GitLab for thorough descriptions of core game features: <https://git.app.uib.no/malicious-malware/kaijumon/-/milestones>

**Changes**

Since oblig 2, we have made the following changes:

* Added a battle system, with the ability to attack, use items, switch Kaijumon or flee.
* Added an inventory system (currently without GUI).
* Added a proper player sprite with animations.
* Added areas inside buildings.
* Added a "how to play" screen.
* Added NPCs and a dialogue system.
* Cleaned up gradle build files.

Features that are under development are:

* Befriending Kaijumons, and making them join your crew.
* Leveling up Kaijumons, such that their stats increase.
* GUI for managing inventory.

Our priority for the next sprint is to write tests to ensure that the already implemented functionality is working according to the specifications. We have encountered some bugs that are mentioned under "known bugs" below.

See <https://git.app.uib.no/malicious-malware/kaijumon/-/boards> for an overview of all features in development.

**Prioritizing tasks**

We have tried to prioritize tasks that meet the requirements to our project, such as:

* Having a help screen which tells the user how to play the game.
* Having at least one power-up.
* Compiling the project with Java SE 19 on Windows, Mac and Linux.

We have done this while implementing new features to meet the MVP. Our goal now is to prioritize bugs so that we can present a more polished product at the end of the semester.

**Current state and MVP**

The MVP was described as follows:

* An area where the player can battle and befriend Kaijumons.
* NPCs that the player can battle.
* One more challenging NPC that functions like a boss battle.

The base game is almost complete, and we can soon add progression to our game. What's missing is the ability to befriend Kaijumons.

We find the MVP to be a good basis for further expansion, but we think that beyond the scope of this project.

**Product and code**

**Known bugs**

* The opposing Kaijumon sometimes appears with lower health during wild encounters.
* Resizing the game window crashes the game on macOS.
* Saving inside the Kaijucenter and then loading spawns the player in the ocean.

**Fixes**

* Fixed JUnit tests not working with newer versions of Java.
* Fixed camera inconsistencies when walking along the edges of the map.

**Class diagram**

See the separate file "diagram3.pdf" in the "doc" folder. This diagram broadly illustrates how the MVC architecture makes up both the battle and world sections of the game.

**Manual tests**

* Save game test:
  + start the game and click new game.
  + move away from the spawn location.
  + Press the "s" key to save the game.
  + exit the game, and then start it again.
  + click "load game" and verify that the spawn is where you were when you saved the game.
* New game test:
  + after the save game test, make sure that starting a new game will spawn you at the original spawn (by the beach) and not the saved game spawn.
* Test that you can walk behind objects.
  + walk to any building or tree and see if you can walk behind it such that the graphics appear in front of the player sprite.
* Test that the "quit game" button in the main menu works.
* Verify that pressing "how to play" shows the help screen.
* Camera test:
  + check that the camera will not move beyond the game map.
  + first move towards the middle of the map such that the camera only sees the map, then walk towards any edge and check that the camera stops. (this test does not include the case where the camera spawned with some of it outside the map)
* Battle tests:
  + attack the opposing Kaijumon and verify that its health declines.
  + use an item and verify that it has been used (use health potion to see the effect).
  + switch Kaijumon and verify that your active Kaijumon has been switched.
  + flee and verify that the battle ends immediately.

**Oblig 4**

**Project report**

**Team roles**

The team roles are still the same as in oblig 2 and 3:

* Team lead: Benjamin Godø Mulelid
* Secretary: Fredrik Limi Ballestad
* Developers: Everyone

We have had no issues with these roles, and it would be too late to change anything now. The role descriptions are still the same:

* Team lead: Sets up meetings, and functions as customer contact.
* Secretary: Records important information from meetings.
* Developers: Develops and tests the features that are agreed upon in meetings.

**Development methodology**

Our initial attempt at doing test driven development failed upon realizing that LibGDX is notoriously difficult to test. We were recently able to create a test application that makes it possible to run headless tests with an OpenGL context. Until now, every developer has had to test their code manually, and this has not always been done thoroughly.

We found the issues board to be lacking in that it was impossible to add a small description to each issue. This was solved by creating a separate channel in Discord with explanations for every feature or bug.

**Meetings**

See "doc/minutes" for the minutes from our meetings.

**Team dynamics and communication**

The communication has been good, and all team members have known what to do. We have done some pair programming, but we still found that some features are best to work on alone. Examples of this are setting up the test application and adding interior to the houses.

**Retrospective**

We should have figured out how to test LibGDX properly during the earlier stages of the project, but this was hard as none of the team members had any experience with the library.

Gradle turned out to be a powerful build tool, but Maven would have been easier to set up, as Gradle was new for all of us. We took a long time to set up gradle tasks properly.

Most of us have about the same number of commits, with some outliers. The team agrees that everyone was involved in the project, and no members were left behind during the development process. All members got the help they needed.

Things that we would have done differently:

* We personally do not like LibGDX, since it mixes rendering and update logic. The difficulties with testing it properly is also a drawback.
* Acceptance criteria should have been translated into tests before implementing features.
* Making a Pokémon clone was unnecessarily ambitious, as we could have gotten the same grade with a simple platformer game.
* GitLab requires us to set up runners on our local computers, and this is an enormous drawback. We chose to avoid pipelines entirely because of this restriction.

Things that worked well and that we would do again:

* Gradle works really well for multi-project builds, and we now know how to set it up properly.
* The MVC architecture worked well.
* Our team structure yielded a good process and result.

**Requirements and specification**

**User stories**

See milestones on GitLab for thorough descriptions of core game features: <https://git.app.uib.no/malicious-malware/kaijumon/-/milestones>

**Changes**

Since oblig 3, we have made the following changes:

* Added different messages to the battle screen.
* Fixed visual bugs in the battle screen, such as Kaijumons not updating upon switching.
* Added a custom test backend and more tests.
* Added interiors to all houses.
* Updated the help screen.
* Added licenses for all assets.

We found that the game was almost done after oblig 3, so the last few sprints have been dedicated to polishing the game and meeting the formal requirements. All base mechanics are now implemented.

See <https://git.app.uib.no/malicious-malware/kaijumon/-/boards> for an overview of all completed features.

**Prioritizing tasks**

As mentioned above, we have prioritized meeting the formal requirements, as well as polishing the game a little. Most requirements were met after oblig 3, but we had not yet figured out how to write tests for LibGDX. We have also written documentation for all public methods, and cleaned up our abstractions such that all model, view and controller classes implement their respective interfaces.

**Current state and MVP**

The MVP was described as follows:

* An area where the player can battle and befriend Kaijumons.
* NPCs that the player can battle.
* One more challenging NPC that functions like a boss battle.

We chose to abandon the idea of befriending Kaijumons, and we resorted to just trapping them by using a Kaijuball item. We also have only one NPC that the player can battle. All the base functionality for adding more fights is in place, but we focused on ensuring that what we had was working properly rather than adding more content.

**Product and code**

**SpotBugs**

* Some of the spotbugs that remain are false positives where it says that a variable never gets a value due to the fact that we load the value in from a json file via Gson.
* Some of the spotbugs that remain are due to the way libgdx is implemented.

**Known bugs**

* Resizing the game window crashes the game on macOS. This issue has persisted throughout the whole process, and we think it might be an issue with LibGDX or LWJGL.
* Tests may fail if they are executed in a wrong order. This might be the cause of one test initializing the Dialogue System, and the other tests being stuck waiting for the dialogue to end.

**Fixes**

* Fixed a bug where every battle action was performed twice.
* Fixed inconsistent asset paths that caused tests to fail.
* Fixed a bug where saving inside the Kaijucenter would make the player spawn in the ocean when loading.

**Class diagram**

See the separate file "diagram4.pdf" in the "doc" folder. This diagram broadly illustrates how the MVC architecture makes up both the battle and world sections of the game.

**Formal requirements**

All formal requirements have been met, except for the abstract factory pattern. We found that we had no related groups of items which made it necessary to implement this pattern. Implementing the abstract factory pattern introduces a lot of extra abstraction, and it would be awkward to try to adjust our project to fit this pattern rather than implementing the correct patterns for our problems.

**Manual tests**

* Save game test:
  + start the game and click new game.
  + move away from the spawn location.
  + Press the "s" key to save the game.
  + exit the game, and then start it again.
  + click "load game" and verify that the spawn is where you were when you saved the game.
* New game test:
  + after the save game test, make sure that starting a new game will spawn you at the original spawn (by the beach) and not the saved game spawn.
* Test that you can walk behind objects.
  + walk to any building or tree and see if you can walk behind it such that the graphics appear in front of the player sprite.
* Test that the "quit game" button in the main menu works.
* Verify that pressing "how to play" shows the help screen.
* Camera test:
  + check that the camera will not move beyond the game map.
  + first move towards the middle of the map such that the camera only sees the map, then walk towards any edge and check that the camera stops. (this test does not include the case where the camera spawned with some of it outside the map)
* Battle tests:
  + attack the opposing Kaijumon and verify that its health declines.
  + use an item and verify that it has been used (use health potion to see the effect).
  + switch Kaijumon and verify that your active Kaijumon has been switched.
  + flee and verify that the battle ends immediately.