PRIMAL

Operating systems and process-oriented programming (1DT096)

Project proposal for group Zeta: ***Axel Hultin*** *(940913-7958),* ***Daniel Degirmen*** *(920626-2033)* ***David Håkansson*** *(971113),* ***Fredrik Andersson*** *(930804),* ***Johannes Almroth*** *(951129-2931),* ***Love Nordling*** *(951106-3258),* ***Viktor Enzell*** *(960113-2658),*

Version 1, 2018-04-11

Table of contents

1. Introduction

When discussing what kind of program would be suitable for implementing concurrency and multi-threading, a majority of the group gravitated towards some form of simulation. Popular ideas included simulating an anthill, cities or an ecosystem. Discussion ensued, and many compelling points were laid out for each suggestion. Simulating cities would be fun, and an added twist in form of an omnivorous “blob” that would consume the city made the idea very compelling. An anthill would be relatively simplistic when it came to graphics and A.I, but could allow for further complexity and development if time was deemed sufficient. An ecosystem would provide a challenge in designing different behaviours for animals, as well as getting them to interact with each other and the environment around them.

After some voting we decided that the simulation of an ecosystem would fun. This would be some form of sandbox simulation where we could play around with variables such as animal behaviour, environment and weather. Since it also has a foundation in reality, it felt like an interesting project from a scientific point of view.

The reason this would be a fitting match for the project requirements is that our animals could be constructed and controlled by different threads and processes. The same could be applied for the environment and weather, making it easy to modularize and keep concurrent. With this, we hope to apply the principles that we have learned in the course so far, and implement threads and concurrency on a larger scale.

We think our main challenge will be implementing a realistic A.I for the animals, as we only have a shallow understanding of the subject, and nearly no practical experience of it. Making multiple A.I concurrent will certainly prove a challenge as well. As such, we will probably construct the A.I according to a simple design, and only strive to emulate basic behaviour.

We hope that this project will provide ample opportunity to improve our skills in developing concurrent programs, as well as expand our current knowledge within object oriented programming. Another skill we hope to develop is to coordinate via git effectively.

2. Concurrency models and programming languages

We chose Java and Kotlin as the languages we will use in the project. The reason for this was that they are both object oriented, runs on the JVM, and we are all familiar with Java. The familiarity with Java ensures that we will not have to learn a new language from scratch, but can instead focus on learning more about concurrency. Object oriented languages also makes it easy to reason about hierarchies such as animals being animals but they may also be herbivores or carnivores. The concurrency model in Kotlin also interested us which we wanted to learn more about.

JVM threads will be used since both Kotlin and Java runs on the JVM, allowing us to use the full Java API. Kotlin also provides its own lightweight threads called “coroutines" . Threads are interesting to learn more about since there are many problems associated with the use of threads, such as race conditions. We believe that getting a better understanding about threads will make us better programmers, as well as prepare us better for further studies in concurrency.

3. System architecture

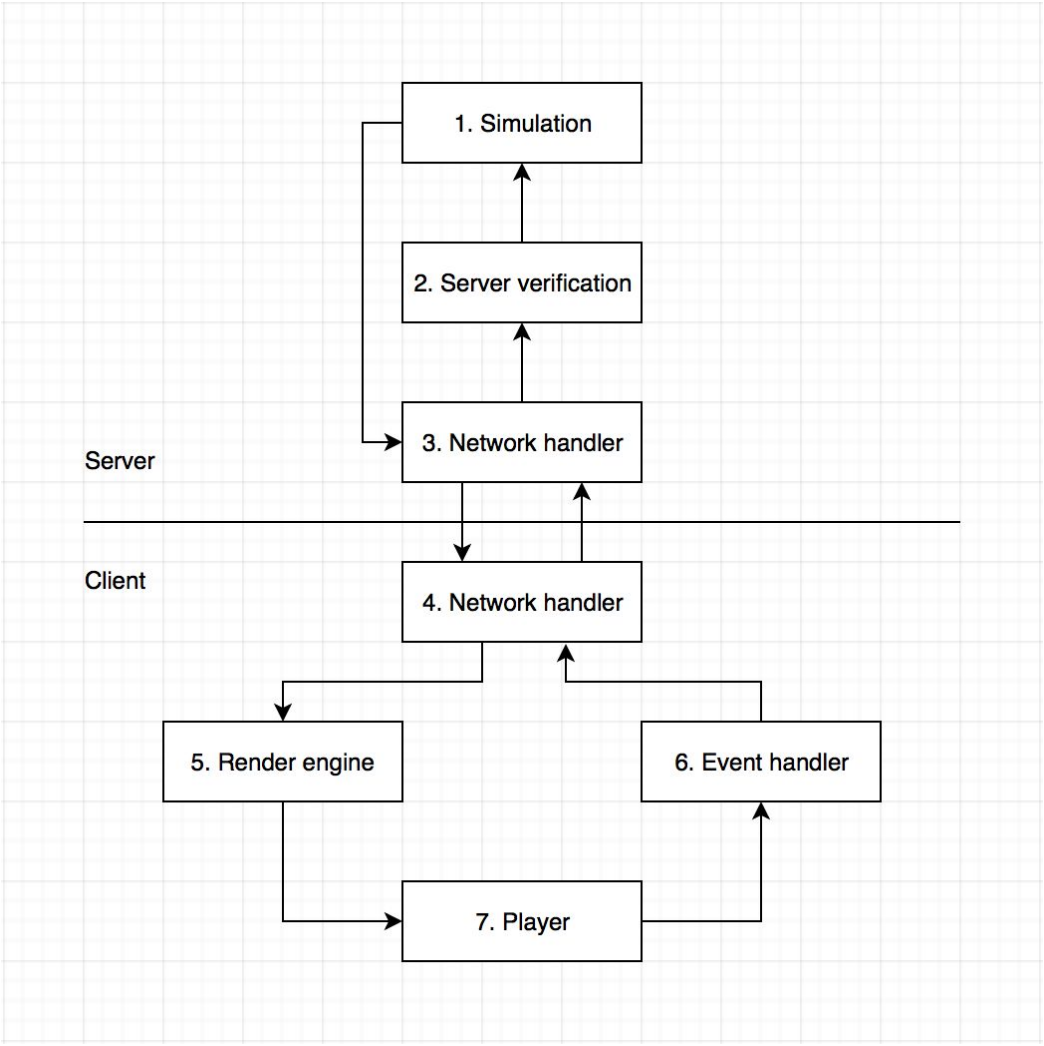

Figure 1. PRIMAL system architecture

1. Simulation - The main component of the system responsible for simulating different entities and environments. Takes commands passed by server verification (2) and passes changes to server side network handler (3).

2. Server verification - Takes input from server side network handler (3) and passes the correct commands to the server simulation (1).
3. Server network handler - Receives and sends information to and from the client.
4. Client network handler - Receives and sends information to and from the server.
5. Render engine - Displays the current state of the simulation.
6. Event handler - Parses input from the player and transforms them to commands and passes them to client side network handler (4).
7. Player - Gives commands to the simulation via the event handler (6).

We decided to go with this architecture because it divides each component into standalone subsystems that can easily be implemented without needing to go and interfere with other subsystems. We decided to divide the client and server because it will be easier to implement multiplayer later.

4. Development tools

We plan to use the following tools in our project:

- **Source code editor:** We have not chosen one single editor that everyone has to use as long as the editor does not affect the code in a bad way.
- **Version control:** We will use Git and Github for version control. Git because we have all worked with it, and it is a powerful tool for safely handling code. When using Github we have established the following rules for submitting code: everyone must work in a branch, and may not push directly to master (not possible anyway since it's protected). Commit messages must be clear about what has been done, and the code follows Google Java style. A minimum of two people must approve changes before the they can be committed to master.
- **Build tool:** Gradle, since we want to learn how to use it (and it's the industry standard).
- **Unit testing:** For unit testing we will use Junit since we have worked with it before and it is a well renowned tool for testing Java code. Findbugs will also be used to assist with finding faulty code or bad practices.
- **Documentation generation:** We will use Javadoc since it is the standard tool for documenting Java and Kotlin. We will adapt the Javadoc standard for writing documentation.
- **Communication:** We will communicate using the chat application Discord because it integrates well with Github and we can have a well organised workflow, using different channels.

5. Conclusion

Ecosystems are fragile, and changes on one end may have disastrous consequences on another. Simulating an ecosystem provides us with a sandbox where we can play around and

observe what happens when we change different variables. The project allows us to learn more about concurrency and how to apply it to something with a foundation in the real world. What will happen if all herbivores dies, or if all carnivores dies? Questions like these will be answered by our project. Using Java and Kotlin, we aim to learn more about threads and the lightweight threads that Kotlin uses. This ensures that we will have a better understanding of threads and concurrency and hopefully how to avoid problems that can arise in concurrent programming. We also see the project as a great foundation for further studies in concurrency.