A Bug's Life
Operating systems and process-oriented programming (1DT096), project proposal , group 5

Oliver Eriksson Edholm (930615-5210)
Aleksander Lundqvist (900728-0317)
Henrik Sommerland (890618-4950)
Edvin Wahlberg (910721-3176)
Oscar Wallster(910615-1096)

April 20, 2015

1 Introduction

We have decided that we are going to simulate an *ant ant*. We want to do this in order to learn more about both *swarm intelligence* and the actor model.
We chose ants as inspiration because of their ability to cooperate in a very large scale and solve complex problems while every ant follows a very simple set of rules.

Our goal is not to simulate the behavior of real world ants or to mimic the details of any biological systems. We are more interested in the theoretical concepts of how complex behaviors can arise from the interaction of a large group of agents each possessing only limited cognitive abilities.

One of the later goals is also to have two or more ant hives interact with each other in a world where food is scarce, and battles are inevitable.

1.1 Challenges

During the course of this project we are aware that there may be many challenges ahead.

1.1.1 Technical Challenges

Making heavy use of the actor model using thousands of actors will create a lot of complexity and there are many possibilities for problems to occur. The most obvious danger is the possibility of deadlocks to occur. With many actors communicating together one is almost certain that one will get some form of circular dependency. So great care needs to be taken to ensure that no deadlocks will occur.

There is also a risk of severe performance degradation in regions with many interacting actors. Although this is something which is intrinsic to the actor model and it may be hard for us to control.

1.1.2 Rules For Interaction

The other challenge will be to set up rules for how the ants interact with the world and how the world gets updated. The parameters for these rules will need to be fine-tuned and it may be hard to find rules which result in complex behaviors. It is also very hard to analytically determine these parameters so the only feasible alternative is to trough experimentation find rules which yields satisfactory behaviors.

1.1.3 Social Challenges

In order to get this project moving forward in a pace that is required, we need to have every member of the group working regularly certain hours of the weeks. In order to achieve this every member needs to have the same mental image over the finished project, therefore we need to be active with communication in the group so that no one is left behind and wonders what he could/should be doing.

2 Concurrency Models and Programming Languages

This section will contain our thoughts on what form of concurrency we want to use and what languages will be appropriate.

2.1 Concurrency Goals

As previously stated we want to create a system which relies heavily on the actor model. Our project idea is well suited for this since the ants are by nature individual agents who and make decisions without any direct knowledge of the global state.

We want to make our simulation free of so called *stop the world* scenarios. These are when we for some reason have to block all of the actors in the simulation in order to perform some kind of action.

2.2 Language Requirement

Our main requirement for the programming language we choose is that it have to meet our concurrency goals. We have discarded c/c++ because it's to low-level and we would have to implement the actor model ourselves or make us of a library we know nothing of. Also java because it's not the proper concurrency model. We are considering python for front-end(graphics), but it's not suited for our back-end needs.

2.3 Rust

Rust is a very compelling language for this project. Built for concurrency, fully realized actor model and really fast. Sadly, it is quite difficult to do very basic things and the need of understanding the ownership model used in Rust will mean that a lot of time will be wasted on things that do not bring the project forward.

2.4 Nim

Nim has a very nice and simple Python-esque syntax, and has surprising performance due to it compiling to C. It has a actor model like take on threads, with message passing between them, and also a proper actor model in its standard library. It is however a very young language (v 0.1 at time of writing) with a large amount of compiler bugs. It is unknown how it would handle the massive concurrency that is needed for this project.

2.5 Erlang

Built for massive concurrency and is the text book example of the actor model. Also extremely low thread-by-thread performance, and blocking receives for messages means extra work in trying to get around some problems that will be relevant in this project.

2.6 Encore

Encore would be a great language for this project since encore is completely built on the actor model. The problem with encore is that it is in its pre-alpha stage and thus its not a viable option for this project.

3 System Architecture

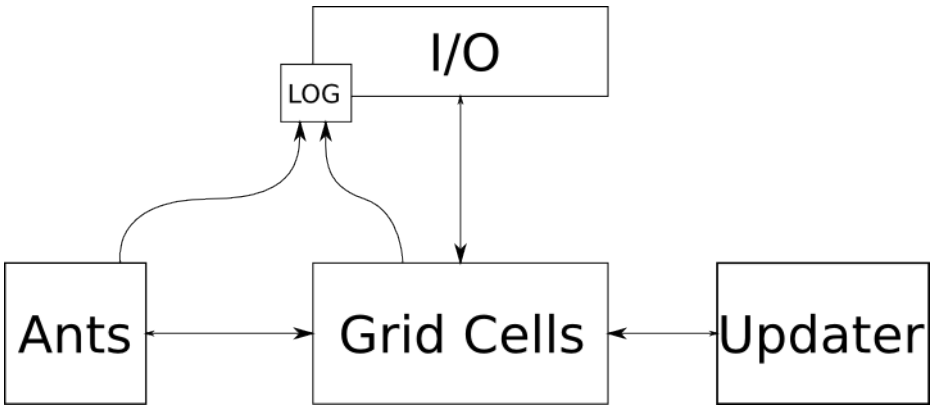

Figure 1: A draft of a system architecture

Above you can see a rough draft of the architecture of the system. Each box represents an agent, note that this is just a conceptual map of the communication between the different agents. The arrows between the a means that they communicate via message passing. The large arrow indicates that a message will be sent between those actors in that direction. The smaller arrows on the lines indicates that the sending actor will block until a reply message is received.

3.1 Ants

In figure 1 the box labeled **Ants** corresponds to the ant object. The ant will be an independent actor holding very little information about its own state but containing the faculties to make decisions based on received input. The ant actor will send a message to the cell querying about the state of its neighboring cells. The ant will then wait for the reply message from its cell and then take some action based on the reply from the cell. the actor then sends a message to its cell requesting to do whatever it is that the ant wanted to do and then awaits a reply whether or not the move was successful.

3.2 Grid Cell

In figure 1 the box labeled **Grid Cells** corresponds to one cell on the "map". Each of these cells contains all the information about the state of that cell. It also contains references to its neighboring cells in order to retrieve the information about its neighborhood. The cell will wait for queries from ants to receive the state of the neighborhood and the cell will also process requests from ants to perform actions such as moving or picking up food.

3.3 Updater

In figure 1 the box labeled **Updater** is something that will handle the background work for the cells. Such as the dissipation of the pheromones. We are uncertain whether or not to use this approach with a separate actor sweeping trough all of the cells and updating them. We could let each of the cells handle their own "passive" updates but that might result in performance degradation since the cells would either always be doing work or we might need to introduce some form of delay.

3.4 I/O

In figure 1 the box labeled **I/O** is the actor that be responsible for the handling of the input and output. This actor will sweep through the cells and query them for their state and then

process output the information in some suitable fashion. We may add support for some form of interactivity at a later state.

3.5 LOG

In figure 1 the box labeled **LOG** will be a simple actor which is just waiting for logging messages from the ants and the grid cells. This will be used as a form of rudimentary output and for debugging purposes.

4 Development Tools

Below there will follow a short discussion about what tools we decided to use.

4.1 Communication and planning

To plan meetings and set up to-do tasks for the project, we have chosen to use Trello. Trello is a very user friendly and will give us a great overview of the progress of the project. It will also be a good tool to keep track of who is doing what. For communication we're going to use Slack to ensure that we have all communication of the project in one place. If we should have chosen to use Facebook it's very easy to get distracted and talk about other things that isn't related to the project. If we really need to discuss something urgent we're going to use Skype.Working in a group of six fully active members it is very important that everyone in the group keeps himself up-to-date with the progress. We were therefore very careful when choosing the communication medias so that everyone would feel that they would check the media every day. Again, the problem with Facebook is that although everyone would check it daily, there would be a lot of other things going on. The communication environment needs to be strictly for work related things to keep everyone focused on what's important.

4.2 Development

We will be using GitHub for source control. We will try to keep track of all issues using GitHubs issue tracking system. We will make no restrictions or requirements of what IDE/editor to be used.
Since we have not chosen our language yet we are not sure what build tools we will use. That decision is based upon what languages we will use.

5 Conclusion

Digital ant colonization is perfect project for developing deeper understanding of the actor model of concurrency, since each ant is an independent actor and only interact with its immediate surroundings. This project can be divided into different milestones which can be developed further as time goes on. We all have a lot and different ideas how we want the final product to work, but we all agree on the basics so as time elapse we will hopefully be able to add a lot more complexity depending how much work will be needed for each milestone. Ergo we start with a stable core and build outwards in different directions.