Parallel Design Patterns II

B138813

February 2019

1 Introduction

We present our report for the second submission of the Parallel Design Patterns Coursework. We first discuss our implementation in Section 2, and demonstrate its correctness in Section 3. Although our implementation has some features similar to a framework, we were not able to succesfully seperate policy and mechanism to an adaquete extend. We will discuss this issue further in Section 4.

2 Implementation

We implemented our squirrel simulation in C++. We present a UML diagram of our implementation in Figure 1. To maximise code re-use, we abstracted as much common functionality into the parent actor class as possible. The actor class holds methods such as `send_msg()` and `msg_recv()` which act as wrappers to MPI functions, whilst also providing extra functionality. For example, `msg_recv()` returns a three item tuple, which contains:

- A boolean, which indicates if the message was succesfully recieved.
- An integer, which indicates where the message was recived from.
- An integer, which is the message itself.

The semantics of the message integer are encoded into the `MSG` enum, which is shared by all actors. This standardisation makes it easy for the programmer to reason about messages, allowing them to dictate how a message if handled in each child class's main event loop. The function `send_msg()` is simple convenience wrapper around `MPI_Bsend()`. Each class inherits directly from the actor class, except for the controller, which is a child of the master class. We made this decision because the master and controller share most of their functionality and variables. Both classes need to know what the simulation's initial values, and share a function to implement this. If C++'s inheritance model did not allow both child and grandchild classes to implement their own versions their parent's virtual functions, this pattern would not be possible.

3 Correctness

4 Further Work

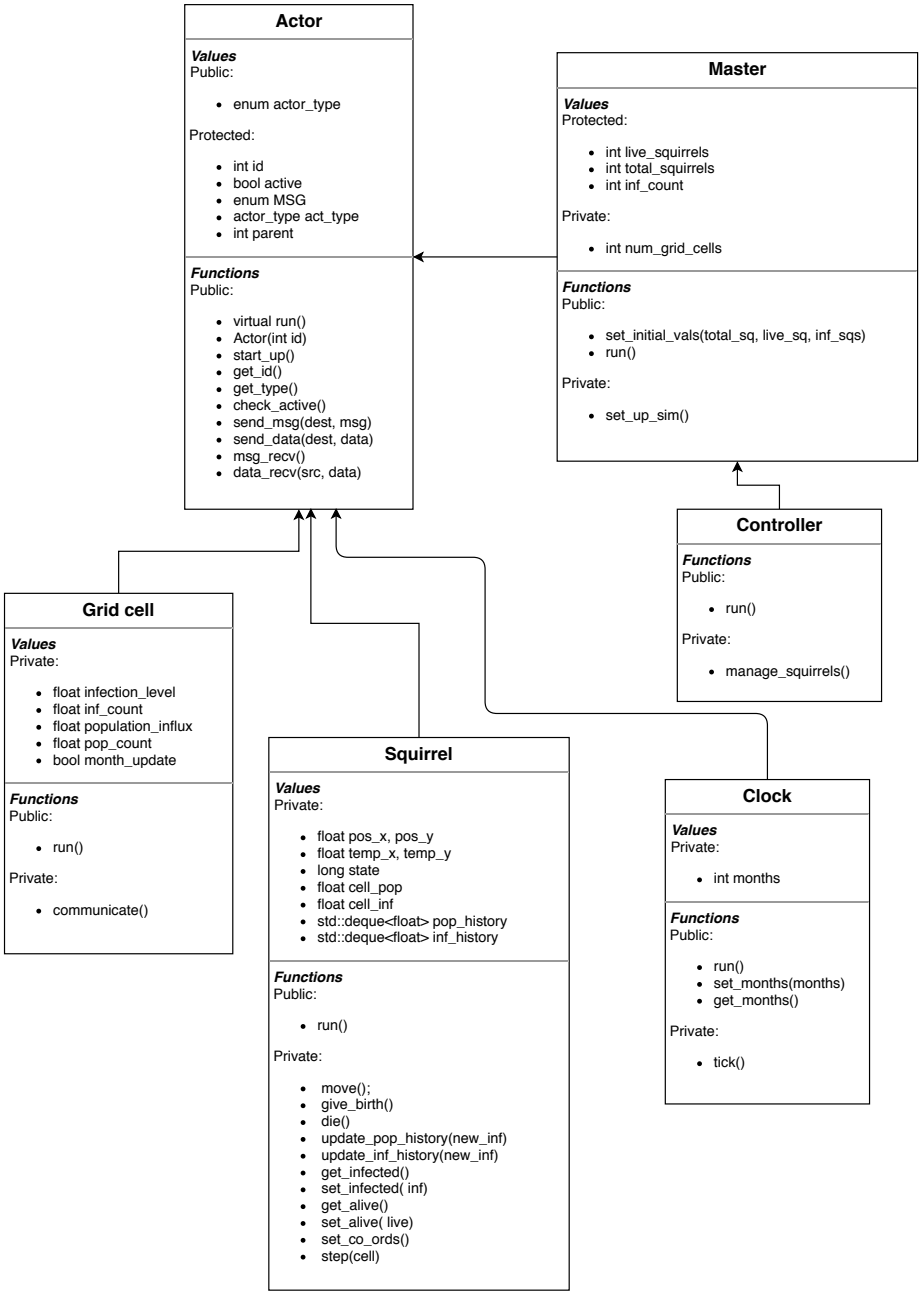

Figure 1: UML Diagram of the Program