University of Copenhagen

Assignment 4: Swarm Simulation using Quadtrees
- Resubmission

Jens Fredskov (chw752)

October 21, 2013

1 Introduction

The following report describes the implementation and testing of a quadtree library and fish farm simulation in Erlang.

2 Implementation

2.1 The Quadtree

The quadtree library has been implemented in the file `quadtree.erl`. The functions used for the coordinator, nodes and leaves all communicate with asynchronous messaging. This e.g. means that we do not wait around for an element to be inserted, by just send the message and move on.

The children of a node is contained in a list of tuples, where each child is matched with its respective corner, allowing us to easier decide where messages should be sent.

When a leaf reaches its limit, it converts to a node by spawning four new empty leaves, and then afterwards sending adds to these with the correct elements.

When the tree applies a function (with `MapFunction/3`) the function propagates down through the tree to all leaves which have elements within the specified bounds. After applying the function if all elements in a leaf which are out of bounds, will be sent to its parent as an add. In this way the elements move up through the tree and down to the new leaf where it should be added.

If the function which is mapped fails and throws and error, the leaf recovers by starting itself with its old data.

2.1.1 The Fish Farm

The fish farm has been implemented in `swarm.erl`. The simulation is started using the function `run/0`, which initializes a farm with 10 predefined fish and then enters the simulation loop with the two phases.

Note that the loop also contains an if-clause which prints an SVG with the fish. If this is unwanted it can be commented out. It however helps in visualizing the movement of the fish.

To update the fish a function `updateFish1/2` is mapped over the entire universe of fish. This function maps a function to return fish that are within one of the zones, for each fish in the tree. Lastly it calls `updateFish2/1` which accepts the messages back from the before mapped function and updates the fish respectively. Since we cannot know how many messages we should expect back `updateFish2/1` returns the updated fish if it receives no messages for 250ms.

The change counter has been changed so that it is instantiated (and reset) to 1 instead of 0, as a counter of 0 gives an arithmetic error in `moveFish/1` which in turn means that all fish very quickly comes to a halt and never moves again.

3 Testing

The quadtree has been tested using the handed out printer example, and of course using the simulation. In the tested cases, the tree has behaved correctly, but these tests are not exhaustive.

The swarm has been tested by running it and checking the generated SVG files. The swarm in general behaves somewhat odd, and from the tests it is still no clear wheter this behaviour is the desired. In figure 1 we can see iteration 1, 10 and 20 from the simulation.

Figure 1: Iterations 1, 10, 20 of the simulation.

As seen it seems like the fish are repelled by each other but might not attract that well. Around iteration 100 or so however the two middle fish will be in a formation with one of the other fish (many of the others will have swam out of bounds).

We can therefore not from these tests conclude wheter the simulation works correctly.

4 Conclusion

We have now described the implementation of the quadtree library and fish farm simulation. We have also described how the implementation has been tested and have concluded that we cannot conclude whether the implementation is correct, but that all tried instances of at least the quadtree worked as expected.