What Program language should this be written in?

I have come across a range of different languages used for IBM’s. There is netlogo <https://ccl.northwestern.edu/netlogo/>, parcel <https://github.com/OceanParcels/parcels> and the C++ <https://github.com/trophia/sna1>.

So which language should we go with?

I feel C++ would be the most commonly known language to fishery scientists so could be the most useful for future adaptation and collaboration.

What behaviour should we apply to each agent?

**Growth**

-growth explicitly (von bertalanffy) length is determined by some function of time or age

-Model growth as a consequence of its environment (bio-energetics model), food/energy availability, life stage

**Movement**

-Box-transfer movement – Markovian\natal homing

-Advection-diffusion movement

-Ontogenetic movement spawning migrations

**Reproduction**

-Find a mate near by?

-Population level sum up the Mature biomass over all individuals in an area and calculate number of eggs based on some Stock recruitment function

**Mortality**

-Natural Mortality

-Fishing Mortality

What Flexible should the spatial and temporal resolution of the model be?

I say completely user defined so people could model daily temporal scales or others could look at seasonal temporal scales

Temporal component

The model consists of

1. Annual steps
   1. User defined time steps within a year e.g. day, season, annual timestep

Defining processes in a time step, each time step is responsible for sorting the processes that occur in each time step. This makes processes the key dynamics. An example of this is say we have an annual time step with the following processes

* M
* F
* Growth
* Recruitment

We would write process code for all of these processes and each year they would get called and iterate over all individuals, or individuals of interest and execute there functionality (individual dies, individual grows, individual moves). The problem with this it is not as fast as hard-coded examples which usually put processes in the individual class iterate over the partition and call a pre-defined set of processes.

So I can think of 2 methods for dealing with this

1. Each process is written out as a separate class and gets passed the partition by reference to alter as the process algorithm is written. This could be threaded, you could block up the partition and send each block to a different thread. This would make it nicely modular-so others can add their own processes. Negative is you would need to give a lot of access from agents to the processes classes goes against encapsulation (a pillar of C++ programming)
2. Individuals have processes within their own class and are executed in a pre-defined order. The time step manager would then be essentially passing bool variables whether that process is being executed in this time step.

I like the 1) option but both should be explored, the first will have tidier code but may come at a speed cost.

Spatial component

We want a user defined environment, I think something like what SPM (https://github.com/NIWAFisheriesModelling/SPM) does, where the user defines a grid of cells and defines reporting definitions but can also be queried by a process to ask where an individual is and where it wants to go based on the individual’s characteristics and the surrounding environment. If we discretise space it will make spatial processes a lot faster, for example applying spatially varying fishing process, instead of querying over all agents if they are in a cell that we want to fish in, having spatial constraints means we know the agents in a cell are ‘potentially’ vulnerable to removal via fishing.

**The coding challenge** with a discrete spatial world. Within each cell what is the best way to manage agents in memory. The three processes that will need to be nailed down,

**deleting agents from a cell (mortality)**

1-remove from memory (so we only model alive fish), C++ containers would have to be non-contagious in memory, otherwise this could be a computationally expensive exercise to reset all agents in memory.

2- put a flag on an agent to say whether it is alive or dead, if dead never enter agents functionality, it can just sit in memory, waiting to get overwritten by another process (recruitment or movement)

**Addition agents to a cell (recruitment)**

-this will depend on the above choice if,

1- just append a new agent to the end

2- search for dead agents overwrite, if non-dead append agent to end

**moving agents among cells (movement)**

This has a slightly different issue, which relates to order of operations. If we iterate over the spatial world, we cannot move agents instantaneously as we may not have run the movement process in the destination cell, and so could end up moving individual’s multiple times. We need to either have mirror world where we make the movement store it in the mirror world and at the end of the process world = mirror world. An alternative is to move agents in continuous space and then migrate them in memory after you have updated their latitude and longitude attributes.

**Random Access**

Another important attribute of the container memory management is allowing random access that is, we can access the ith element efficiently. Unfortunately, this is usually best for containers that have contagious memory storage vector, array, deque etc

**TODO** - Bench mark a range of containers for the above issues, make the bench marks as realistic as possible, i.e 1 million agents in each agent 1000’s of random access, multiple years.