- More annotations in the code needed. What do variables represent?

- Why are males, females and juveniles different breeds?

(Juveniles do not perform any procedure so far; almost all state variables seem to be the same for males and females, so they could go to turtles)

But even if you decide to have different breeds, most of the code in the set up is the same for both breeds, so you can just use “ask turtles” for setting up common variables with the same initialization and use “ask males” or “ask females” only for variables with different values. Although it would be easier if you just have a variable called “sex”, then you can initialize variables with different value between males and females using “ask turtles with [sex = male]”

- mu\_thresh and h2 are not used so far.

- Some state variables should be global parameters:

(They’re the same for all individuals of the same sex, or even the same for all individuals, so there’s no point in model them as state variables if there’s no individual variability. If the difference is between sexes, then you can just have one value for each sex.)

\* my-month, my-day

\* prob-death could be probably defined as a local variable, but it’s ok

\* Vp Va Ve mu\_cond V\_cond

(Note that the values of Vp Va Ve are the same when there’s conflict than when there’s no conflict.)

- Parameters defining quality (the parameters of the normal distribution, mean and sd) should be defined as global parameters, not hard coded. Just in case you want to change their values more easily or include them in SAs.

That happens also with L, K, mass0 in the fecundity equation

- Variables as anadromous can use True/False instead of binary 1/0 conditions. Then you can use primitives that need a true or false condition: e.g., “ if anadromous [migrate]” Apart from the fact that’s more intuitive. But it’s not relevant.

**Mortality procedure:**

- Why using “xcor <= 50” instead of pcolor as done before? Everything should be the same for consistency.

- With the current formulation, trout only die at sea. To make them die also at freshwaters, “if random-float 1 < prob-death [die]” must be out of the “ifelse xcor <= 50”

**Carrying capacity:**

- Shouldn’t be different capacities for each environment (freshwater vs. marine)? Indeed, I’d say the carrying capacity should only affect trout in the freshwaters.

**Reproduction procedure:**

- What’s the difference between the procedures for resident an anadromous trout? The looks like exactly the same. Will it be the value of the parameters of the fecundity equation?

**Choose mates:**

- With the current formulation,, you’re caught in a loop. There’s a “ask females[choose]” in the “go” procedure and then a “ask females…” within the “choose” procedure, so spawner females are asking all females to do what’s inside the brackets…

- Within the “while” loop, in “let prop\_B “ you’re not calculating the proportion of residents in remaining available males but in the set of available males at the beginning, because you’re dividing by nmax , which was set before the loop started, and it’s not being updated.

- In the “set mates…” command, it might be that there are no resident spawners or no anadromous spawners left within available males, so no spawner would be added to the set, so you’ve to include the command to pick up one spawner of the other migratory behavior in that case.

**Genetic architecture:**

The model so far is quite slow. You should profile it to see which parts are slowing it down but I have the feeling it’s the reproduction procedure. I think this procedure could be faster if you use matrices instead of using lists and a random choice of the locus values.

My question here is: is there a different if a gene is (1 0) or (0 1). Based on the current genetic architecture, the answer is no, because G is calculated as weight1\*(locus1.1+locus1.2) + … , so in the end, the value of the locus are just summed up.

If there’s not a difference, then the genetic map of an individual can be stored as a matrix of 1 row and 21 columns (each gene) [genes matrix, GM]. At initialization, each gene value can be drawn from a uniform distribution “random 3” so it can have values of 0 (0 0), 1 [either (1 0) or (0 1)], and 2 (1 1). You have to create a global parameter, which is also a matrix, with the weight values (again, 1 row, and 21 values) [weights matrix, WM]. G for an individual would result from the multiplication of the two matrices: G=GM\*WMT, WMT being the transpose of WM. From this multiplication of a 1x21 and 21x1 matrices, you get a scalar, which is G. So you get the G value for each individual at initialization.

During transmission of genes, you just have to sum the matrix of the mother and the matrix of the father and divide by 2. Then you have to check each element; if the element of the resulting matrix is 0 (0 + 0), 1 (either 2 + 0 or 0 + 2 or 1 + 1), or 2 (1 + 1) it remains the same; if it’s 0.5 (either 1 + 0 or 0 + 1) then you get a random draw “random 2” and decide randomly whether is 0 or 1; it happens the same as with 1.5 (either 2 + 1 or 1 + 2). You calculate then G in the same ways as at initialization.

I think that in this way you would have the same genetic architecture but the calculation should be much faster and the implementation is quite straightforward with the Matrix extension, which is designed to optimize this kind of algebraic calculations. I can help you with implementation in the code, if needed.