SLiM Workshop Series #2: The Chromosome Hierarchy

1. Launch SLiMgui

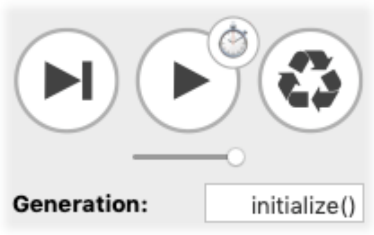
Locate the SLiMgui application, named `SLiMgui.app` or just `SLiMgui`. There may be an alias to it on your desktop, or perhaps in the Dock; in any case it should be installed in `/Applications` on macOS, while on Linux it will probably be in /usr/bin or/usr/local/bin.

Launch it by double-clicking it. You should see a modeling window with a default model in it, almost identical to the one we have just been looking at:

```
// set up a simple neutral simulation
initialize() {
   initializeMutationRate(1e-7);

   // m1 mutation type: neutral
   initializeMutationType("m1", 0.5, "f", 0.0);

   // g1 genomic element type: uses m1 for all mutations
   initializeGenomicElementType("g1", m1, 1.0);

   // uniform chromosome of length 100 kb with uniform recombination
   initializeGenomicElement(g1, 0, 99999);
   initializeRecombinationRate(1e-8);
}

// create a population of 500 individuals
1 {
   sim.addSubpop("p1", 500);
}

// output samples of 10 genomes periodically, all fixed mutations at end
1000 late() { p1.outputSample(10); }
2000 late() { p1.outputSample(10); }
2000 late() { sim.outputFixedMutations(); }
```

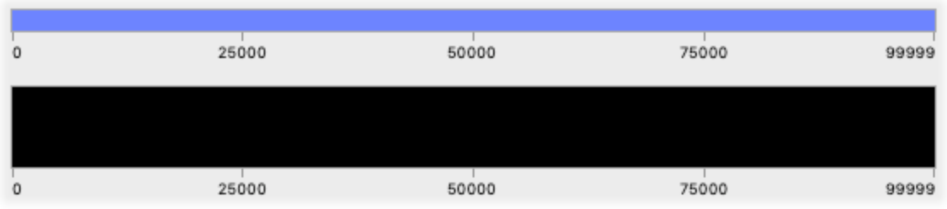
2. Run the model

Locate the play controls at the upper right of the window:



The leftmost button is the Step button. Press it once, and the simulation will step over the initialization phase, during which `initialize()` callbacks are called. The generation counter shown above will switch from "initialize()", indicating that the initialization phase is about to execute, to

"1", indicating that generation 1 is about to execute – it always displays the generation that is *about to happen*, not the one that has just happened. Notice that the chromosome view also changes to show tick labels, since the length of the chromosome and the genomic structure is now known:

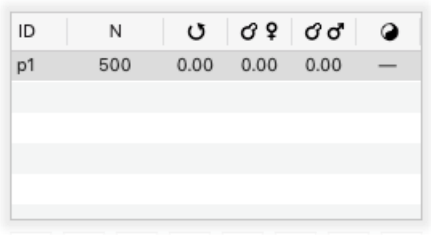


The top chromosome view always shows the full chromosome, from beginning to end, and is color-coded to indicate the genetic structure (all blue, here, because there is just one genomic element of a single type). The bottom chromosome view can show a zoomed-in view; we'll see that below.

Finally, a little output appears in the output pane, showing the initialization that has occurred:

```
// Initial random seed:
2205191794193

// RunInitializeCallbacks():
initializeMutationRate(1e-07);
initializeMutationType(1, 0.5, "f", 0);
initializeGenomicElementType(1, m1, 1);
initializeGenomicElement(g1, 0, 99999);
initializeRecombinationRate(1e-08);

// Starting run at generation <start>:
1
```

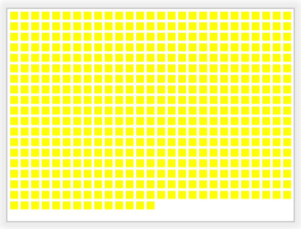
Press the Step button again, to step over generation 1. The `sim.addSubpop()` call will execute during generation 1, in the `early()` event (declared without the "`early()`" specifier this time – remember, `early()` is the default callback type). This call will add a new subpopulation named `p1`. This can be seen in the population table at the upper left:



ID	N	↺	♂♀	♂♂	◕
p1	500	0.00	0.00	0.00	—

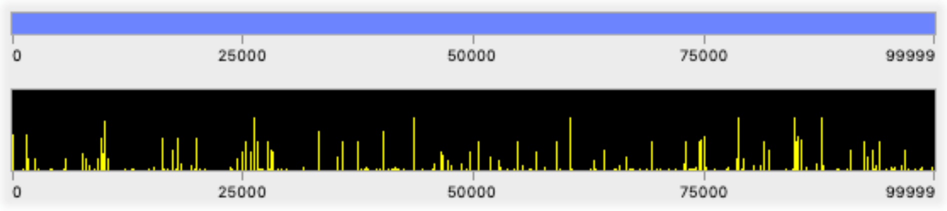
This indicates that `p1` has 500 individuals in it. The other columns show things like the selfing rate, cloning rate, and sex ratio; you can hover the mouse over the column headings to get tooltips.

Also, the individuals in the model are now shown in the individuals view:



Each yellow square represents one individual; there are 500 squares. Each is colored yellow, because each individual has the same (neutral) fitness, since this is a neutral model; other colors would represent fitness values above or below neutral.

Now press the center button in the play controls, the Play button. You will see a quick flurry of activity, and then the run will finish and SLiMgui will stop. At that point, the chromosome view will show something like this:



Each yellow bar represents one mutation. The position of the bar on the *x*-axis indicates the mutation's position on the chromosome, and the height of the bar indicates the mutation's frequency in the population; when a bar reaches the full display height it is fixed across the whole population.

The generation counter should show "2001", since generation 2000 has finished and the model has completed. A bunch of model output should have been generated, too: a population sample of size 10 at generations 1000 and 2000, and then a dump of all fixed mutations (if any). We will discuss the output format in a later exercise.

Now click the rightmost button in the play controls, the Recycle button. The model will reset to its initial state: an undefined chromosome, and a generation counter showing "initialize()". Press Play again and watch the model run forward. Recycle and Play a couple more times. Notice that the chromosome view updates continuously as the model runs; if you watch closely you can even see particular haplotypes, composed of sets of associated mutations, varying together in frequency.

3. Modify the model

Let's try making some simple modifications to this model. For starters, find the line:

```
initializeMutationRate(1e-7);
```

and change it to:

```
initializeMutationRate(1e-8);
```

Notice that as soon as you changed the model, the Recycle button turned green. This indicates that the model script no longer matches the running model, and encourages you to recycle so that you are running the current version of the code. Let's follow that suggestion, and click Recycle followed by Play. The model will run again, but now with much less mutational activity; there will be just a handful of segregating mutations at any given time because of the low mutation rate.

It would be nice to be able to watch these dynamics for a longer period of time. Change:

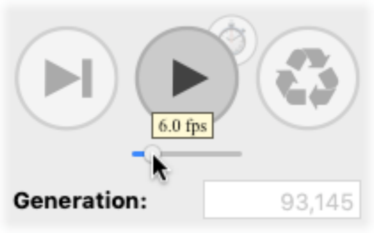
```
2000 late() { sim.outputFixedMutations(); }
```

to:

```
200000 late() { sim.outputFixedMutations(); }
```

Recycle and Play. Now it will take a little while for the run to finish. Note that while the model is running the Play button is highlighted and other controls are disabled. You can click the Play button again to pause the simulation at any point, and then you can Step one generation at a time, or click Play again to resume full-speed playing, or Recycle to start again from the beginning.

One final user interface detail is worth mentioning. While the model is playing, you can use the small slider below the play button to adjust the playing speed:



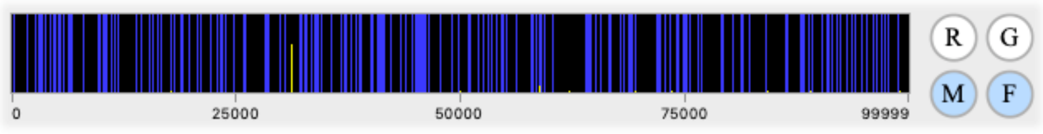
While dragging the slider, a tooltip appears that shows you the frame rate (in frames per second, or fps) that you are choosing. Dragging the slider all the way to the right resumes full-speed play.

EXERCISE: Try modifying other things about the simulation, such as the recombination rate (initially `1e-8`), the chromosome's length (initially specified with a final base position of `99999`), the population size (initially `500`), or the distribution of fitness effects for `m1` mutations (initially specified as fixed, `"f"`, with a value of `0.0`; try a normal DFE, type `"n"`, with a mean of `0.0` and a standard deviation of `0.01`, specified as: `"n", 0.0, 0.01` instead of `"f", 0.0`).

4. View fixed mutations

If you run this model for a while (not necessarily all the way to the end), you may notice a curious thing: you never actually see any fixed mutations. Yellow bars rise toward the top of the chromosome view, and then disappear. What's going on?

When mutations in SLiM reach fixation, by default they are turned into *substitutions*, removing them from active simulation. SLiM no longer tracks their presence in each genome; being fixed, they are in every genome by definition. They are no longer used in fitness calculations, either, since Wright–Fisher models (such as the model we are working with now) use only *relative* fitness; a mutation possessed by every individual will generally produce the same fitness effect in every individual (unless epistasis is present), and will therefore produce no relative fitness effect at all – and can thus be neglected by SLiM, for speed. They are still tracked, though. All the fixed mutations will be output by the `sim.outputFixedMutations()` method call at the end of the run. They can also be seen in SLiMgui by clicking the **F** button to the right of the chromosome view:



This toggles on display of fixed (**F**) mutations, shown by default with blue bars. The automatic substitution of fixed mutations can be turned off; we will explore that in a later lesson.

5. Make the genetic structure more complex

To start with a fresh version of the default model, close the current window and then press command-N to make a new SLiMgui window. Change the final generation event to run in generation 200000, as before, so that it's easier to see the dynamics of the model over time.

Now we're going to make the genetic structure of this model more complex. We'd like to have beneficial and deleterious mutations, in addition to neutral, and we'd like to have a somewhat structured chromosome with a couple of non-coding regions that produce only neutral mutations, and then a couple of genes that frequently produce deleterious mutations and infrequently produce beneficial mutations (as well as neutral mutations).

Let's start by defining some additional mutation types. Modify the `initialize()` callback code to contain the following definition, just below the definition of `m1`:

```
initializeMutationType("m2", 0.1, "e", 0.1);
```

As an aside: it is not necessary to type out long symbols like "initializeMutationType". One could copy/paste them, of course, if they already occur in the script; but one can also leverage the power of *auto-completion* in SLiMgui. On a new line in the `initialize()` callback, where you want to insert the line shown above, try typing `ini` and then pressing the escape key (usually labeled "esc"). A pop-up appears showing all the symbols SLiMgui knows that start with, or otherwise resemble, the base string `ini`. Partway down the list is `initializeMutationType()`; you can double-click it, or move down to it with the arrow keys and then press return to accept it. This saves some typing, but you can do even better by typing `iMT` and then pressing escape; there are only three symbols that match this pattern, and `initializeMutationType()` is the first one, so you can then simply press return to accept it.

In any case, once the line above is added, a mutation type named `m2` is defined, but not yet actually used in the model. It uses a dominance coefficient of `0.1`, just for the heck of it; more interestingly, it draws its selection coefficients from an exponential distribution (`"e"`) with a mean of `0.1`, rather than using a fixed selection coefficient (`"f"`). Each new `m2` mutation will draw its own selection coefficient from this exponential distribution of fitness effects (DFE).

Let's define one more mutation type (try using auto-completion with `iMT`):

```
initializeMutationType("m3", 0.2, "g", -0.03, 0.2);
```

This defines `m3` as having a dominance coefficient of `0.2` and a DFE that is a gamma distribution (`"g"`) with a mean of `-0.03` and a shape parameter of `0.2`. (Gamma distributions cannot have a negative mean, formally; SLiM interprets this as meaning that the distribution has a mean of `0.03`, but the sign of each drawn value is flipped.)

OK, we have the mutation types we want. We will use `g1` for non-coding regions, since it undergoes only `m1` mutations. Let's define a new genomic element type, `g2`, that we will use to represent genic regions, by adding this line after the definition of `g1` (try auto-completing `iGT`):

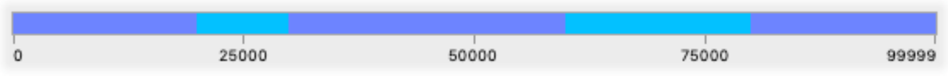
```
initializeGenomicElementType("g2", c(m1, m2, m3), c(100, 1, 10));
```

This definition says that `g2` will undergo all three types of mutations that we have defined (`m1`, `m2`, `m3`), in relative proportions of `100`, `1`, and `10`; in other words, for every hundred neutral (`m1`) mutations there will be, on average, one beneficial (`m2`) and ten deleterious (`m3`) mutations. SLiM computes the actual fractions for you from the relative proportions.

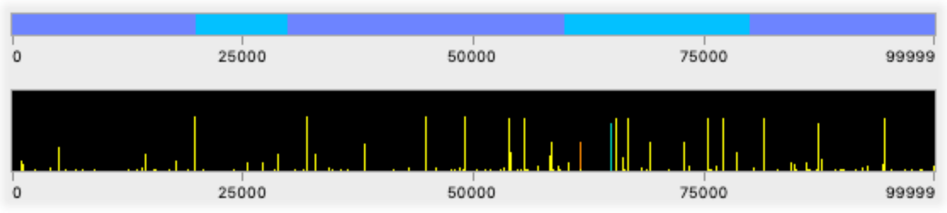
Last, we need to define a chromosome structure that uses these genomic element types to alternate non-coding and genic regions, as represented by genomic elements, by replacing the script's existing `initializeGenomicElement()` call with this code (`iGE` will auto-complete):

```
initializeGenomicElement(g1, 0, 19999);
initializeGenomicElement(g2, 20000, 29999);
initializeGenomicElement(g1, 30000, 59999);
initializeGenomicElement(g2, 60000, 79999);
initializeGenomicElement(g1, 80000, 99999);
```

That's all of the modifications we need to do. Now you can Recycle and then Step once, to see how the model is set up in SLiMgui. The top chromosome view shows the chromosome structure:

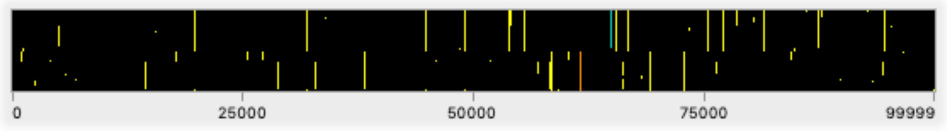


This shows graphically the structure that we just defined with calls to `initializeGenomicElement()`. SLiMgui has chosen blue to represent non-coding regions (`g1`) and turquoise to represent genic regions (`g2`). The non-coding regions, following the `g1` definition, will undergo only neutral mutations; the coding regions will undergo all three mutations types, following the `g2` definition. The resulting evolutionary dynamics can be observed by running the model (i.e., clicking Play):



In this screenshot, the second genic region has a deleterious mutation (orange) and a beneficial mutation (cyan) both at intermediate frequency. These mutations are not linked, however, and as the model runs forward (most easily seen with Step or with the play speed slider), the beneficial mutation will fix while the deleterious mutation will be lost.

There is an alternative visualization for the chromosome view that reveals that the two mutations are not associated. Right-click on the chromosome view (i.e., click with the right mouse button, if you have one, or press the right edge of your laptop trackpad, or try a two-fingered trackpad click, depending on your System Preferences settings – ask if you have trouble), and a pop-up menu will appear. Select "Display Haplotypes" and the view will switch to something like this:



This is a haplotype structure plot of the population, rather than a frequency barplot. Here it can be seen that the subset of individuals with the deleterious mutation is disjoint from the subset with the beneficial mutation, making it easy to predict how the population will evolve.

EXERCISE: Select "Display Haplotypes" yourself, and run the model forward in that mode. Try pausing at various points and switching between displaying frequencies and displaying haplotypes, to get used to the difference in the information conveyed. Now, note that the context menu you see on the chromosome view has some other display options as well; experiment with those to figure out how they work and what they do.

EXERCISE: Define a new genomic element type, `g3`, that you will use to represent introns, which should undergo 90% neutral mutations and 10% deleterious mutations (no beneficial mutations). Split up the second genic region defined with `g2`, to now use `g2` and `g3` to define a structure of alternating exons (`g2`) and introns (`g3`). Recycle and run to see the results.

As a final capstone, open SLiMgui's model drawer by clicking this button on the right-hand side:

☞

You can see the mutation types and genomic element types that you have defined in the tables shown in the drawer. Try hovering the cursor over `m2` or `m3` to see a preview of the DFE associated with that mutation type (the cursor needs to sit for more than a second without moving at all).

6. BEEP! With extra time, SLiM manual chapter 7 is worth skimming for added depth.