Model
Much of our model was unchanged from Project 4. Our only interesting implementation was for handling timesteps; instead of trying to keep a secondary data structure (e.g., a grid) updated, we used a couple HashSets and HashMaps to keep track of Critters during timestep execution. We made this work by mathematically deriving a simple two-way hash for coordinates, which allowed us to uniquely identify every spot in the world with an integer, which we used as the key for these sets/maps.

As a timestep would be processed, we would maintain a mapping from these hashed coordinates to linked lists of Critters at those spots. Our new look() method functions by just checking this map.

View
The view component was designed to use javafx. We created a side panel on the left for the user to control the environment by setting the seed, adding Critters, steping through the world, animating the environment, and viewing stats about each type of Critter. See the Controller section for more details. As a visual representation of the world the right side of the component is a grid which is a 2D representation of the torus the critters reside on. Each Critter can define its CritterShape and the fill and outline colors it would like to be displayed as. We use javafx's native graphics library to display these shapes and colors in the grid for a presentation that is informative and pleasing.

Controller
The controls really just involved repackaging most of the commands from Project 4, except we also had to update the view after many of the "commands" (e.g. spawning new Critters). The main new functionality was animation, which relies on javafx's Timeline class, which is very similar to java.util.Timer. Unfortunately, Timer doesn't work with javafx UI components since the spawned thread doesn't have access to them, which is why we needed to use Timeline.

Notes
- There's a corner case where look() can be called on a location where more than one Critter resides (before any encounter is processed). We arbitrarily chose to return the string of the first Critter in that spot. (This is dependent on the order in which Critters were added to the world.)
- Because we were developing in IntelliJ instead of Eclipse/Linux, the reflection code that loads the possible Critter classes looks for .java files in the src folder instead of .class files in the out folder. If we were to distribute this, we would fix this to look for the .class files instead, which is more "correct." To this end, we made our code very easy to change—we just need to set the directory to look in and the target file extension.