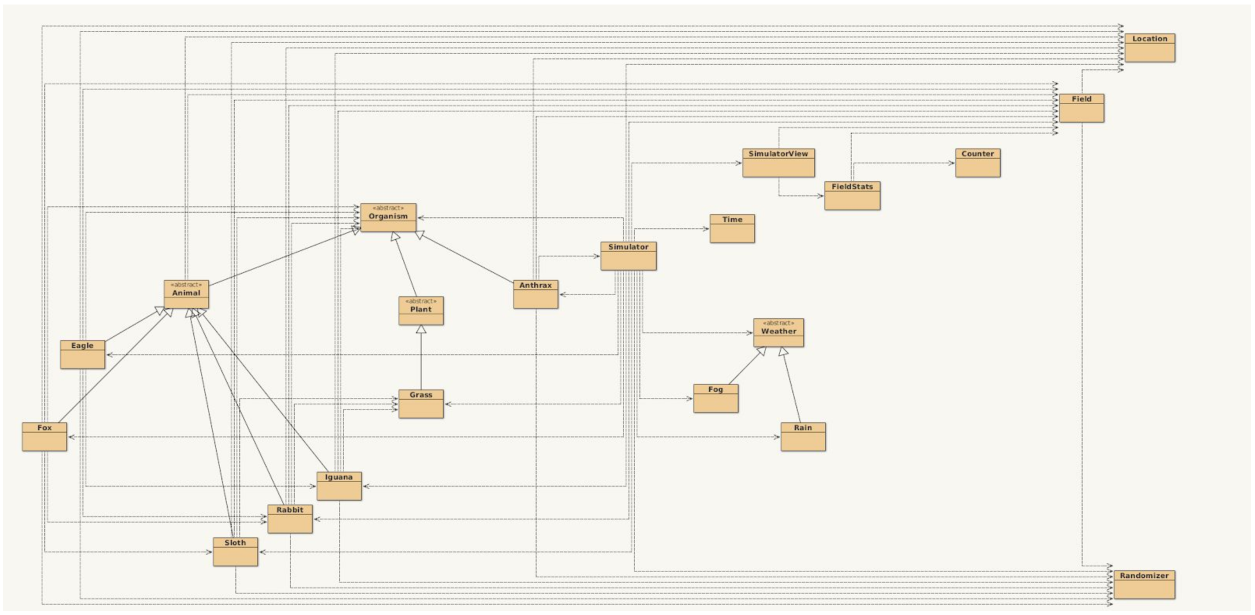
Assignment 3: Predator/Prey Simulation

Sebastian Tranaeus(K1762177) and Peter Xu(K1764015)



Simulation Description

Our simulation extends the base functionality of the starter code given to us. Here is a breakdown of our classes:

- *Organism* - Abstract class that implements the core, shared functionality of any living thing.

- *Animal* - Abstract class that extends *Organism*. Adds sex functionality, allowing for subclasses to be male or female.
- *Eagle* - Concrete class that extends *Animal*. A predator that eats *Rabbits* and *Sloths*.
- *Fox* - Concrete class that extends *Animal*. A predator that eats *Iguanas* and *Sloths*.
- *Sloth* - Concrete class that extends *Animal*. Eats *Grass*, prey of *Fox*.
- *Rabbit* - Concrete class that extends *Animal*. Eats *Grass*, prey of *Eagle and Fox*.
- *Iguana* - Concrete class that extends *Animal*. Eats *Grass*, prey of *Eagle*.

Animal has two informal groups of subclasses, predators and preys. Their interactions with each other are defined in Animal's subclasses. The defining characteristics of a subclass of Animal is what classes it consumes(e.g. Fox eats Sloth, Sloth eats Grass), and the probabilities set for various behaviours such as hunting, procreating, escaping etc.

We implemented the notion of night and day in the application by creating a class called *Time*. *Time* is instantiated when the simulator starts running. Using the *Time* class allowed us to implement a day and night period, defined by when in day the sun rise and sets. *Time.isNight()*

is used throughout the application to influence organism behaviour when it's night time. Internally, we represent time in minutes. So 1 means one minute past midnight and 1440 means midnight. This gave us precision, without getting too tedious. The simulation starts at 8AM, which is 480 minutes.

Distinguishing between male and female organisms is handled in the *Animal* class. It is assumed that plants in this application are asexual. When an animal's *procreate()* method is running, if it finds an organism of the same animal type, and of the opposite sex, then there is a probability that they procreate.

Predators compete for the same food source whenever they share a prey animal. In the scenario where two different predators are adjacent to an animal which they both hunt, it is up to the probabilities set in these predators' classes to see which one of these is victorious in hunting that prey.

Extension Tasks

Plants

We decided to only have one type of plant, *Grass*, to keep the application simple. Rabbits, Sloths, and Iguanas all eat *Grass*. When first addressing this extension task we constructed a *Plant* abstract class to accommodate for if we wanted to add more plant types that would have shared characteristics. Plants are assumed to be asexual.

- *Plant -* Abstract class that extends *Organism*.
- *Grass -* Concrete class that extends *Plant.*

Weather

When a weather phenomena is instantiated, it changes the relevant probabilities of Organisms its affecting. Each weather phenomena has a duration, which is how many steps it lasts. When that duration is exceeded, the changed probabilities are reset. Good weather(e.g. sun, little wind etc) is implicit in the application. For every step in the simulator, there is a given probability that a weather phenomena will occur, assuming the same weather phenomena is not already occurring.

- *Weather* - Abstract class that implements functionality shared across different types of weather phenomena.
- *Rain* - Concrete class that extends *Weather*. Makes it harder for an E*agle* to hunt, for a *Sloth* to escape, and makes it easier for *Anthrax* to spread.
- *Fog* - Concrete class that extends Weather. Makes it harder for an *Eagle* to hunt, and easier for a *Fox* to hunt, and for a *Rabbit* to escape.

Disease

- *Anthrax* - Concrete class that extends *Organism.* Represents a particular case of Anthrax disease. Behaves differently than other children of *Organism*, in that there is only ever one instance of Anthrax. It keeps track of the spread of its infection using an

ArrayList of all the Organism it’s infected. When it acts, it iterates over this list of Organisms, and for each Organism tries to infect the Organisms around it. If successful in infecting those adjacent organisms, they get added to this list.

Known Problems
No known bugs that lead to the simulation crashing at runtime.

General issues with the application we would have fixed with more time:

- Animals have a tendency to die quicker at night. This could be to do with how much the probabilities change at night, or an actual design flaw in the logic that changes behaviour at night.
- A default value in the Simulator class for when the simulation starts would be a good idea. Right now, we have hardcoded the value 480, which represents 480 minutes passed midnight(8AM). A default value would reduce the issues associated with hardcoding the same value in multiple parts of your application, and make it easier to update that value.
- One issue is that male Animals have the same waiting time as females when it comes to how soon they can procreate again. This is flawed in terms of representing a real life situation for some animals. This could be rectified by abstracting the PROACTIVE_INTERVAL variable to the Animal abstract class, where its value is set in the logic for sex in the Animal constructor.
- The Anthrax disease is only created at the populate method in Simulator. So it can not randomly appear in the simulation. This was not our intention, but something we were unable to address on time.

In terms of having the simulation run a full 4000 steps, we’ve seen this happen in our simulation in a number of scenarios. To mention one, when a prey, predator, and Anthrax are left, the game can run indefinitely.