***Table of Contents***

*1 Introduction p. 3*

* 1. *Scope p. 3*
  2. *Related Documents p. 3*
  3. *External Requirements p. 3*
  4. *Internal Requirements p. 4*
  5. *Design Issues p. 4*

1. *Internal Design p. 4*
   1. *Data Flows p. 5*
2. *Unit Test Strategy p. 7*
   1. *User Defined Stats and Program Flow p. 7*
   2. *Printing of Map, Stats, and Tile Data p. 7*
   3. *World Generation p. 7*
   4. *Food Chain p. 7*

*4 Additional Information p. 8*

*4.1 Design Methodology p. 8*

*4.2 Improvements p. 8*

1. **Introduction**

The Sask. Wildlife Federation wishes to study the life cycle of animals in a certain habitat in Saskatchewan. The simulation, which is named Animals, is a model that simulates the food chain of animals in that habitat. Each animal in the simulation will behave like simplified versions of their real-life counterparts.

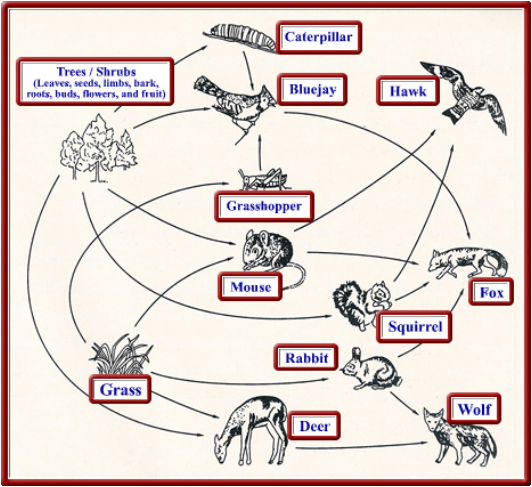
The animals will be generated in a world and move around each day. Along with animals, vegetation will also be generated in this virtual world for some of the animals to eat. Animals takes all the requirements given by the lab instructor, and creates an easy to use simulation for any user to use.

* 1. **Scope**

Animals is a food chain simulation of a specific habitat in Saskatchewan. Given specifications from the customer (or lab instructor) the simulation will interact with the user to get specifications for the simulation. The simulation will consist of flora and fauna, which is defined in the food chain diagram provided (see Related Documents Figure 1). This document will outline the requirements for this simulation to run as realistic as possible, how it was designed, and how testing was done and how successful it was.

* 1. **Related Documents**

Food chain diagram as shown in figure 1:



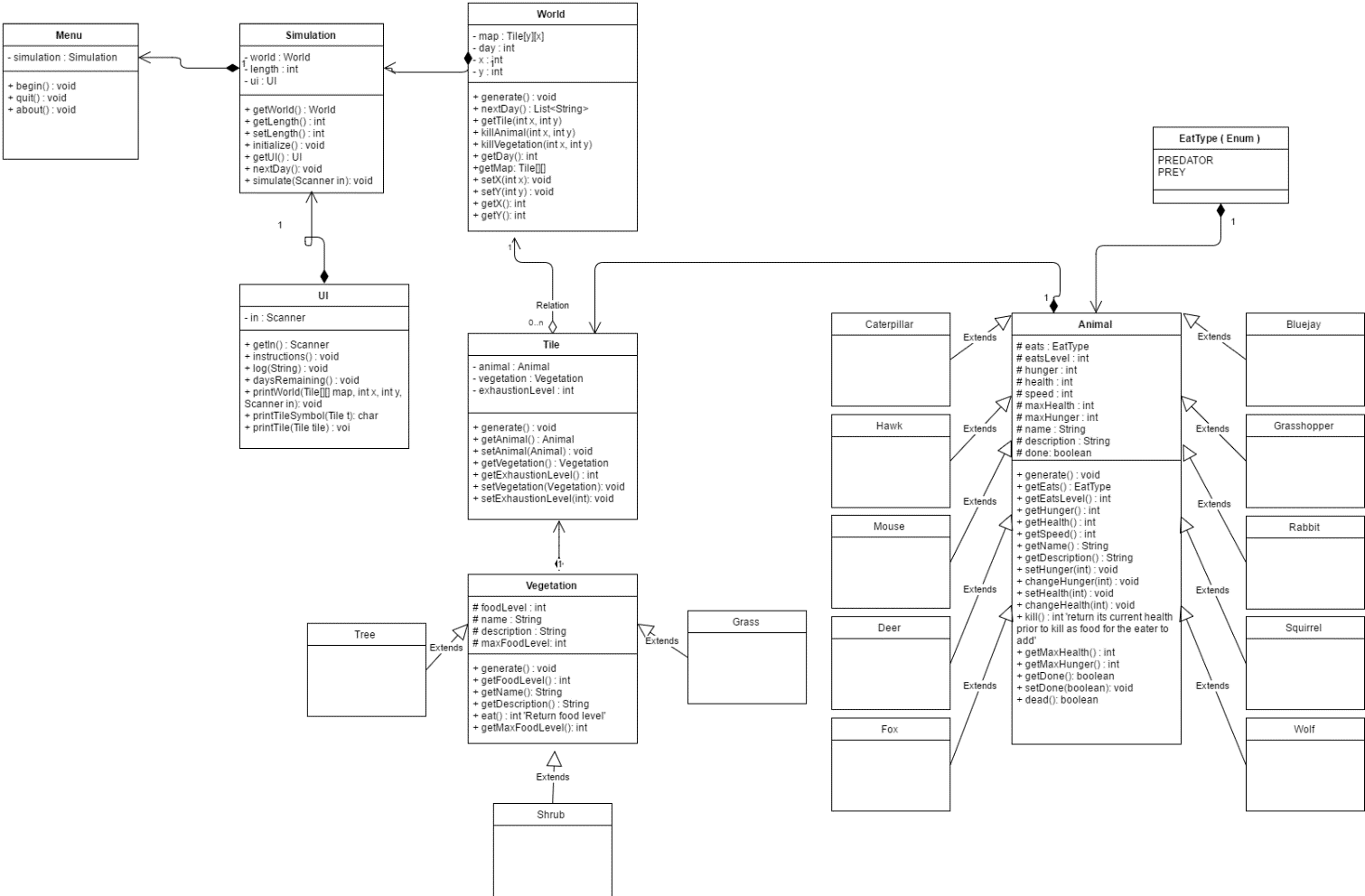
**Figure 1: Animal Food Chain (Lab 4 Handout, 2016)**

See references and rest of document for other relevant material.

* 1. **External Requirements**

1. The ability to start, stop, and get information about simulation
2. Allow the user to enter in the requirements for the simulation (animal #s, vegetation #s, size of world, number of days, etc.)
3. Allow user the option to print out world, days events, death statistics, moves on that day, etc.
4. Give user ability to quit the simulation at any moment
5. Allow user to skip simulation by certain number of days
6. Ability to print out individual tile stats at the end of each day
7. Ensuring the program is user friendly: simplistic UI, straightforward operation, instructions.
   1. **Internal Requirements**
8. Define different stats for each different type of animal, same for the vegetation
9. Define which animals eat which
10. Allow Animals to eat Vegetation
11. Allow animals to die of hunger
12. Store days’ events in a array list
13. Randomly move animals around the world
14. Randomly place the animals in the world at beginning of the simulation
15. Keep track of how many of each type of animal are left
16. Keep track of the days left
    1. **Design issues**
17. Determining factors that gets an animal and/or vegetation killed
18. Maximum size of the world
19. Maximum length of simulation
20. Number ratio of animals when randomly generated
21. Determining how an animal moves and ensuring they stay within the bounds of the map
22. Determining the rules that define what remains on a tile after a simulation day. For example, if a tile contains multiple animals and vegetation, who eats what, and what remains when the day is over.
23. Setting up a food chain.
24. **Internal Design**

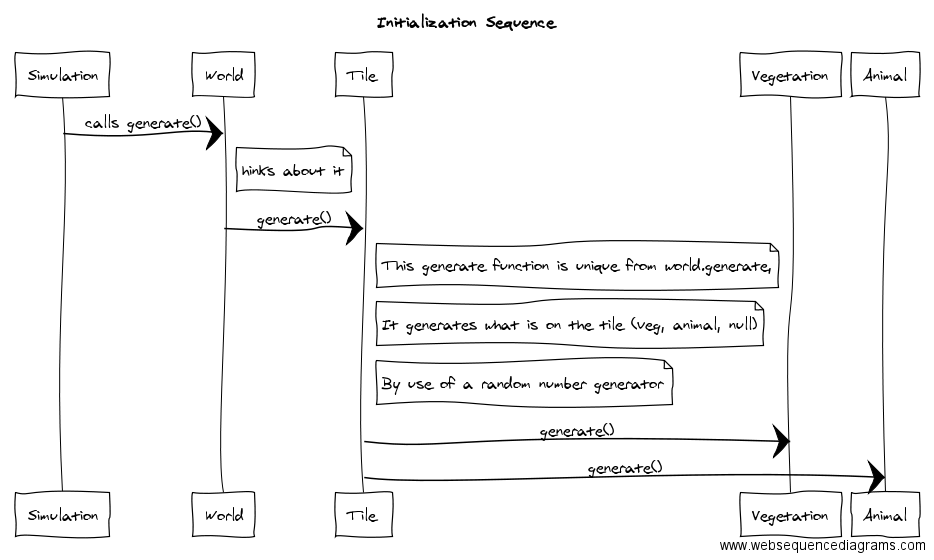
For determining the internal design, we created a UML diagram to list the classes and how they work together (as seen in figure 2). The program starts with the Menu class. This class allows the user to start the simulation, quit the simulation, or learn more about the simulation. This class is mainly a user interface, which is composed of a Simulation. The Simulation class is the next class in the diagram. Simulation is where the world is generated and the simulation is started. Simulation contains a World and a UI. The UI class is where the world and individual tiles are printed, as well as other functions that are used to communicate with the user, as well as receive some input from them as well. The World class represents the world we are simulating, it holds the map as an array of Tiles. It also deals with getting the day’s stats, as well as killing vegetation or animals. The next class Tile defines a certain location on the world map. Tiles will either contain an Animal, Vegetation, or neither. There are many different animals that are inherited from the Animal class such as caterpillar and deer. This is the same with the vegetation class, where grass, tree and shrub are inheriting from it. Each type of vegetation has a different max food level.



**Figure 2: UML Diagram**

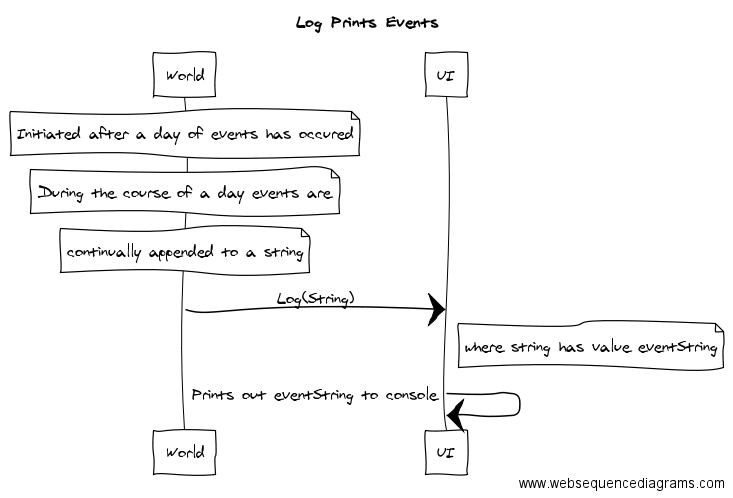
* 1. **Data Flows**

The flow of the program is defined by our three sequence diagrams. One represents the initialization of the program (Figure 3), the log printing event, and the day by day sequence (Figure 5).



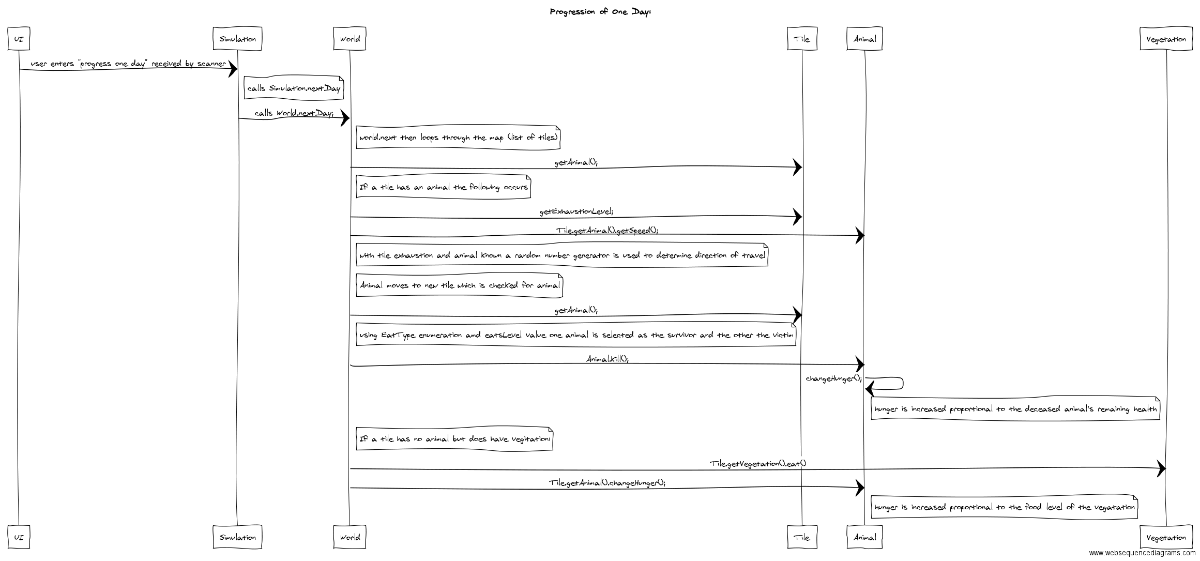
**Figure 3**

The initialization starts in the simulation where generate is called. This generate calls the worlds generate. In world, tiles generate function is called for each tile contained in the world’s map. In the Tile class, it randomly generates animals and vegetation. At the end of this sequence, the world has been properly initialized, with a map of the world we are simulating containing animals and vegetation.



**Figure 4**

This sequence helps create a string of the days’ events. In world, after each event in the day has occurred, it will then save that event in a string. Events may be if an animal died, if some flora were eaten, etc. This string is then returned to the UI class, where the stats are printed.



**Figure 5**

Starting in the UI class communicates with the Simulation class. From here, the next day function is started. Here, we call the World next day function to start the days’ events. The World class loops through each tile in the world map to start moving animals around the world. First it determines if there is an animal present. If there isn’t, it continues to the next tile in the array. If there is an animal present on said tile, we get the exhaustion level as well as the animals speed to determine the animals next movement. After this animal moves, it checks the tile it has moved to see if there is any life there. If there is another animal, the stronger animal will the kill the weaker one. The stronger animals hunger is then decreased by how much health the weaker animal had remaining, If the new tile has vegetation instead, the animal eats the vegetation placed on that tile and its hunger is reduced by the health of the plant. Once this sequence has finished, a day has passed and all animals have been either moved, killed, or ate.

1. **Unit Test Strategy**

Once the program was completed, a few tests needed to be run to make sure all aspects of the program functioned properly before the first demo. There were four main tests that were done: testing anything user defined such as length in days, printing the correct stats and map, world generation, and lastly how the food chain held up.

**3.1 User Defined Stats and Program Flow**

The user defined variables consisted of the length of the simulation in days, the x and y lengths of the world, as well as whether the user wanted to skip, continue, or quit the program. First the menu class was tested to make sure it took in values for length and the map within the proper range defined by the program. The menu class also was tested to made sure it quit the program properly, and could begin the simulation with no bugs.

In the Simulation and UI class, where is a lot of information given by the user. The test was to make sure that the program properly responded to the user’s instructions. The list of instructions to be tested were as follows:

* Quitting the simulation at any time
* Continuing the simulation for every day
* Skip by any number of days, or to the end
* Whether the user would like to print out individual tiles, and the coordinate of said tile

Once these requirements were met, anything defined by the user had to be working properly. If that is the case, the program will flow as it should without any obvious errors that can be viewed by the user.

**3.2 Printing of Map, Stats, and Tile Data**

Printing stats, whether the map of the world or simple tile stats, can be tested in the simulate class. At the end of each day, all the stats from that day should be saved into a string, which is then printed outlining the day’s events. Also in the simulate function is the print world function is called. This function needs to print out a grid format with the proper sizing and proper x and y values at each new row/column. As well, the grid should show if there is an animal, vegetation, or neither on a tile in the world. From the print world function, the print tile function is called to show the stats of whatever is on that specific tile. It should show the user what is on the tile, and if there is either an animal or flora on the tile it will continue to print out the stats of this specific object. If the world prints correctly, the user should be able to tell if a tile will print animal, flora, or no stats.

**3.3 World Generation**

The generation of the world is important, if this doesn’t work properly there may be no simulation to run. The world needs to generate a map of tiles on a two-dimensional grid. Each tile on this grid needs to randomly generate and animal, vegetation, both or neither. If this is successful, when the first day is ran the user should have a good idea of how many animals and vegetation are present in the program via the daily stats.

**3.4 Food Chain**

The last test is to determine if the food chain functions properly. The rules for the food chain are simple:

* All animals can eat vegetation
* Animals that are stronger, will kill the weaker animal
* Animals can starve to death, vegetation will not

If these rules are met, the simulation is function how it is intended. To do this, simply running the simulation and paying close attention to the statistics being printed at the end of each new day. These stats will determine whether there are any flaws in the food chain.

1. **Additional Information**

­­

* 1. **Design Methodology**

The design steps for the project is as follows:

-Lab 4, create UML and Sequence diagrams

- Here most of the important features are determined

- Code the project

- There were four different components coded by a different team member each

- User Interface

- Generating World

- Let a Day pass and save stats

- Identify each individual animal and vegetation

- Update UML and have Demo 1

- Update code and adding stats feature, as well as user defined animal generation

- Update documentation and have Demo 2

The design process went very smoothly. All features that were required for the final demo were already included in the very first UML diagram. This made updating the code less intensive and less time-consuming. Along with the solid UML diagram and the use of branches in GitHub, the coding process was smooth and testing very easy. The biggest design miscommunication seemed to be the use of Scanners, which was defined in the UML diagram from the beginning but not properly implemented in the program. This was the most time-consuming component to test, however even that did not take up much time. The design of the code from Lab 4 and on helped the creating of Animals run as smoothly as possible up to the final Demo.

**4.2 Improvements for Future Versions**

Although this is the final demo and release for Animals, there is still room for improvement if there is a possibility of future versions. The first may be to have the program extend until all animals are dead in the simulation. Since the program is tracking the number of animals living, it will not be hard to determine the end where all animals are dead. Another major improvement would be to strengthen the food chain. With the current rules, any animal that is stronger than the other will automatically eat the encountered creature. However, this system has birds eating birds and other animals eating creatures that they would not normally eat (refer to figure 1). This needs to be cleared up for a relatively realistic simulation to occur.

1. **References**

Douglas, T. (2016). ENSE 374 Lab 4 Handout.