Week 1.

This week we have implemented the following features:

- **TestReader** - reads test files for information
- **Distributer** - distributes testfile paths
- **Borrow** - Rabbit holes
- **Grass** - Sweat green grass for the simulation
- **Rabbit** - A cute rabbit with various functions
- **BabyRabbit** - A cute baby rabbit (inherets from Rabbit)

TestReader

The TestReader reads the test files for information out the types and amount of entities in the simulation. The TestReader is being used along side the **Distributer**, to distribute the test files to the simulation. There are numores functions in the TestReader, so it is easy to get the information out of the test files. The TestReaders main function is **getMap** function, that returns a map of the test file with the corrusponting entity type and thier amount. TestReader also has a corrusponting test file, that is used to test the TestReader.
TestReader has the following functions:

getFilePath(); getFileContent(); getFileContentString(); getMap(); getRandomIntervalNumber(); getWorldSize(); isNumeric();

Distributer

The Distributer distributes the test files path for easy use in the simulation. The Distributer is a simple enum that stores all the test file paths and has some functions to get the test file paths.
Distributer has the following functions:

getUrl();

Borrow

The Borrow is a simple class that is used to create rabbit holes in the simulation. The does don't have a constructor and is only used to create and dispay rabit holes. This could be interface
Borrow has the following functions:

none

Grass

Grass is a class with diffrent functions that is used for the behavior of the grass object. One of the functions is the **spread** function. The function is used to spread the grass object to the surrounding tiles with a 10% chance of the grass spreading.
Grass has the following functions:

act($World$ world); spread($World$ world)

Rabbit

The Rabbit class constructs cute rabbits to the simulation with normal rabbit behaviors. Rabbits can move and eat grass, but unfortunately die of starvation as well. Most of the development time was spend on the Rabbit class. all requirements has not been fully developed this week, but the Rabbit class is somewhat functional. There has also been made a baby rabbit class, that is used to create baby rabbits for the reproduction function.
Rabbit has the following functions:

act(*World* world); digBorrow(*World* world); move(*World* world); die(*World* world); eat(*World* world); reproduce(*World* world); getRandomSurroundingTile(*World* world);

Baby Rabbit

The Baby Rabbit class is a class that is used to create baby rabbits for the reproduction function. The baby rabbit class is a simple class that is used to create baby rabbits. This class is not fully functional after this week, but will be worked on.
Baby Rabbit has the following functions:

@override act(*World* world);

Week 2.

Animal class

We've made a lot of changes to the rabbit class, and added other new classes. With 3 animals, we decided to make an Animal superclass.
We figured it would make sense to make a moveAway and moveTowards function, as a lot of the code was already written for the rabbit class. This utilizes our already created move function. We implemented a vision varaiable, that is used to determine how far away the animal can see, and a hunger variable, that is used to determine how hungry the animal is.

Rabbit class

We reworked the rabbit class to be more object oriented. We made a lot of the functions of rabbit into functions of the Animal superclass. We made a kind of behavior tree for rabbits, where they first will move towards their home if it is night, then move away from potential predators, then eat if possible, otherwise move towards grass, dig a burrow if it does not already have a home, a 20% chance to reproduce if possible, and finally a 50% chance move randomly.

Predator interface

We made a predator interface, that is implemented by the wolf and bear class. This is to make it easier to check if a class is a predator. We experimented with making an abstract predator class, but decided against it. The interface could be compared to the NonBlocking interface from the libary. The only function in predator is the abstract attack function, that the predators can use to attack other animals. This is primarily meant to be used by the bears if another animal is in their territory, or wolves if they are in a pack strong enough to attack a bear.

Bear class

The bear class is supposed to have a territory based on where it was spawned. The bear has different priorities based on some varaiables. If it is starving (has a hunger level below 3) it will search for food, not nessesarily in its territory. Food sources can be berrybushes or rabbits. If it is not starving, it will try to protect it territory. If any class that inherets from Animal is in its territory, it will attack it. if it is instead hungry, (has a hunger level below 8) it will search for food in its territory. We have not yet implemented the behavior of only finding food in the territory if it is not starving. otherwise it will move towards the center of the territory, or a random tile in the territory if it is already in the center.

Bush class

The bush (or berry as it is called in the input files) are a food source for the bear. We decided to not make them spread like grass, as they don't disappear when eaten. After the bear has eaten from the bush, it will take 5 steps during the day before the berries regrow.

Wolf class

The wolf, like the bear, implements the predator interface. It is supposed to hunt rabbits, and eat them. It functions a bit like the bear, except it does not have a territory. Instead it has a pack, that is a list of wolves.

Wolfpack class

The wolfpack is a class that is supposed to make the wolves hunt in packs. The leader of a wolfpack is the first wolf in the list containing the wolves. This class is responsible for adding the wolves to the lair.

Lair class

We remade the burrow into a lair class, that can also be used by the wolf.

FileReader class

We had to make changes to our FileReader class, as we had to read the input files in a different way. Since the way wolf packs and bears are defined in the input files, we had to slightly alter the way it reads them.