**Brief abstract and introduction**

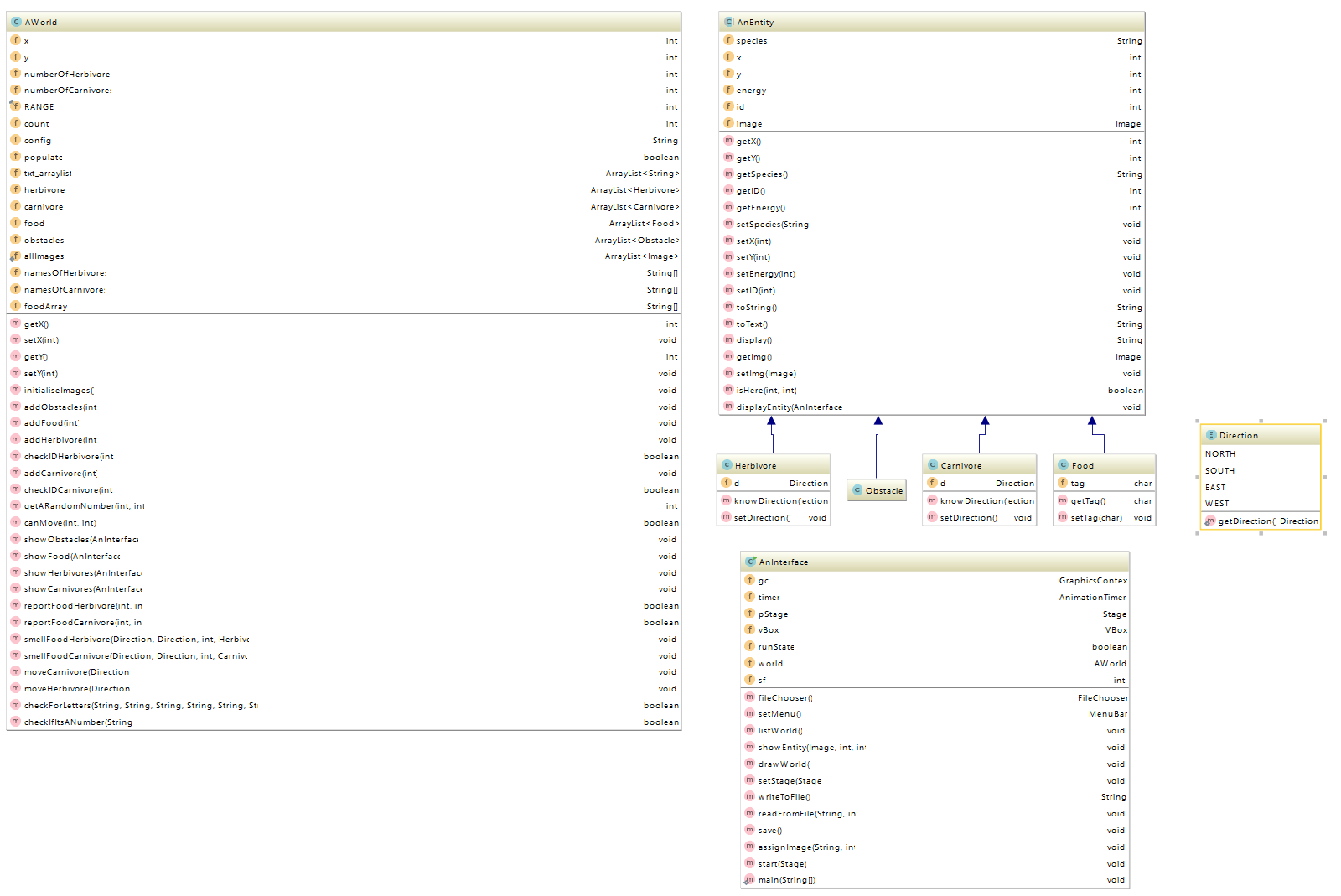
This project required us to develop a 2 dimensional “Artificial Life Simulator”, where in entities move towards food by smelling them and by avoiding other entities and obstacles. When the entities encounter food, they consume it, thereby, increasing their energy. The program development consisted of making a world configuration, and running and managing it on GUI windows.

**Description of the OOP design of the Simulator**

**package reading.Kartik.ArtificialSimulator**

* **AnEntity(abstract class)**: This is an abstract class which defines what an entity is supposed to be. Other entities inherit from this class, thus, defining their unique behavior. Some of the key variables include:
  + int x, y,energy,id: They hold the x-coordinates, the y-coordinates, the energy and the ID’s of the entities respectively.
  + Image img: Stores the image of entity, which gets displayed via a GUI window.
* **Herbivore:** This is a class which inherits the class AnEnity, to store the herbivore entities in the world. Apart from using the superclass’s constructor, it also has a variable ’d’ of type ‘Direction’, which stores the direction of the herbivore’s current movement.
* **Carnivore:** This is a class which inherits the class AnEnity, to store the carnivore entities in the world. Apart from using the superclass’s constructor, it also has a variable ’d’ of type ‘Direction’, which stores the direction of the carnivore’s current movement.
* **Food:** This is a class which inherits the class AnEnity, to store the food entities in the world. Apart from using the superclass’s constructor, it also has a variable ’tag’ of type ‘char’, which stores the food’s state (vegetarian or non-vegetarian).
* **Obstacles:** This is a class which inherits the class AnEnity, to store the obstacle entities in the world. It uses the superclass’s constructor, and has an energy of 0, since it cannot move and eat.
* **Direction:** This is an enum, which generates and returns random direction coordinates for the life forms (herbivores and the carnivores).
* **AWorld:** This is the class which defines the world’s size, and the implementation for adding, removing and printing the obstacles in it. Some of its characteristics are as follows:
  + It has methods to set and get the x-coordinates, y-coordinates, images for the entities.
  + It has an ArrayList of Obstacles, Food, Herbivore and Carnivore to store the life
  + It has methods to add obstacles, food, herbivores, and carnivores to the world.
  + It has methods to display all the entities.
  + It has a smell method which allows the life forms to smell their surroundings for food.
* **AnInterface:** This is the class which loads the world with the GUI interface on the monitor. Some of its key functions and variables are as follows:
  + pStage: Sets the stage for the scene to take place.
  + vBox: Stores the toolbar on the right of the Borderpane
  + timer: This variable is of type AnimationTimer, and is used to draw the world every second.
  + sf: This is an integer variable which is used to scale the whole GUI for better visibility. By default, it is set to 40. This means each pixel will be 40x larger.
  + menu(): This function is responsible for initializing the top menu bar, which has some functions to control the simulation, like saving and loading the configuration, editing life forms, viewing info about the application and author, etc.
  + drawWorld(): This draws the world on a Canvas variable.
  + setStage(): This function sets the stage for the scene to execute. It initializes the toolbar’s buttons and their action, when clicked, and initializes the BorderPane variable, where the simulator is displayed at each iteration.

**UML Diagram:**



**Tests:**

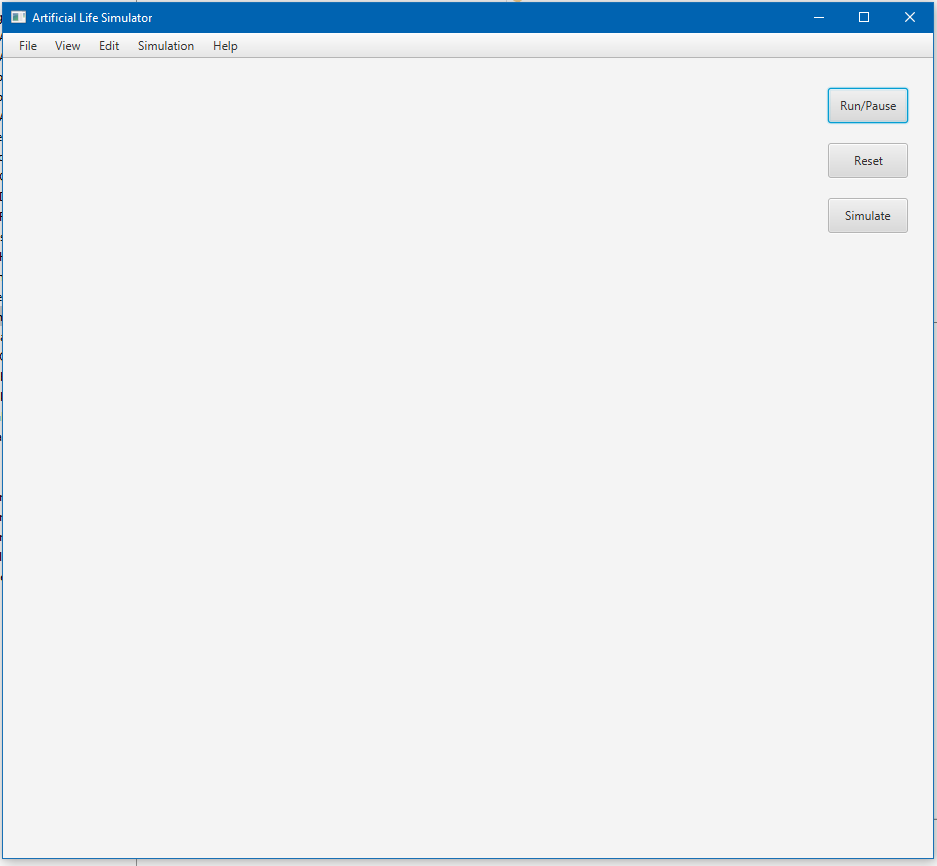
The application was tested and the following worked successfully:

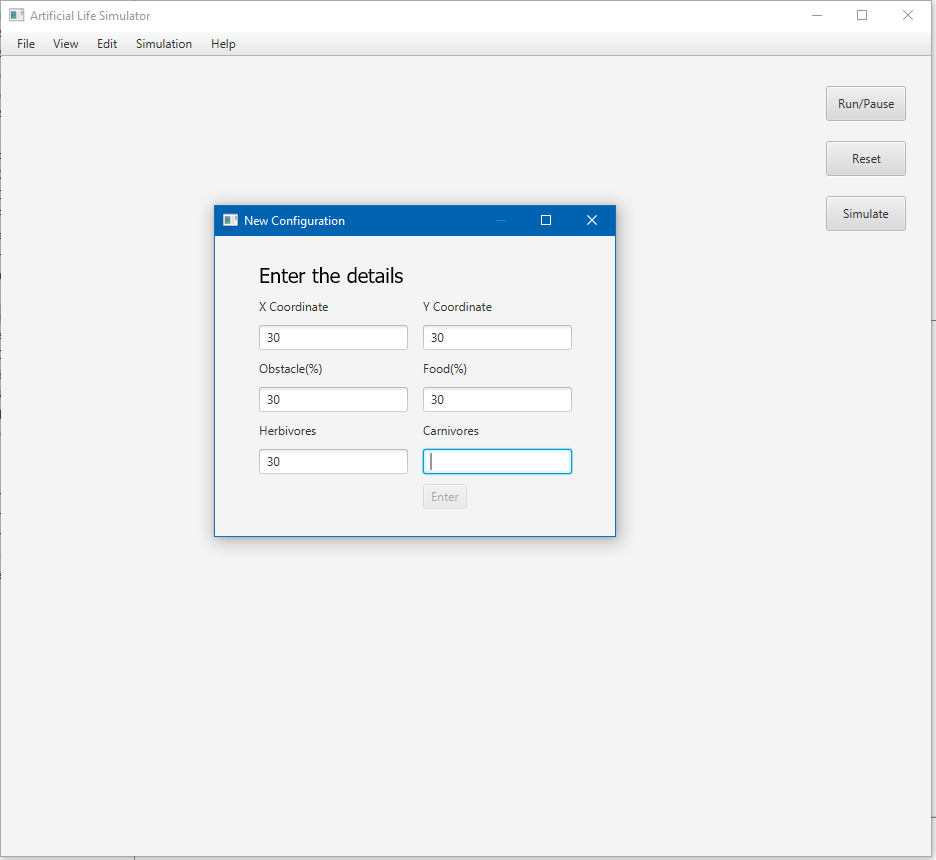
* Using the ‘File’ menu to create, load configuration, save, save as, and exit.
* Using the ‘View’ menu to display information about the world and its entities.
* Using the ‘Edit’ menu to add new life forms, delete and modify their parameters.
* Using the ‘Simulation’ menu to run, pause, reset, and display map at each iteration
* Using the ‘Help’ menu to display information about the application and the author.
* Using the toolbar on the right to run, pause, reset, and simulate (display map at each iteration).
* Testing if the total number of entities is greater than the worlds area. If yes, display error message, and prompt the user to enter valid values.
* Scaled each pixel by about ‘sf’ times successfully without any performance hiccups.
* Able to accept large values of world and its entities, without affecting the performance.

**Informal Presentation & manual**

1. **Launching the application and creating a new configuration**

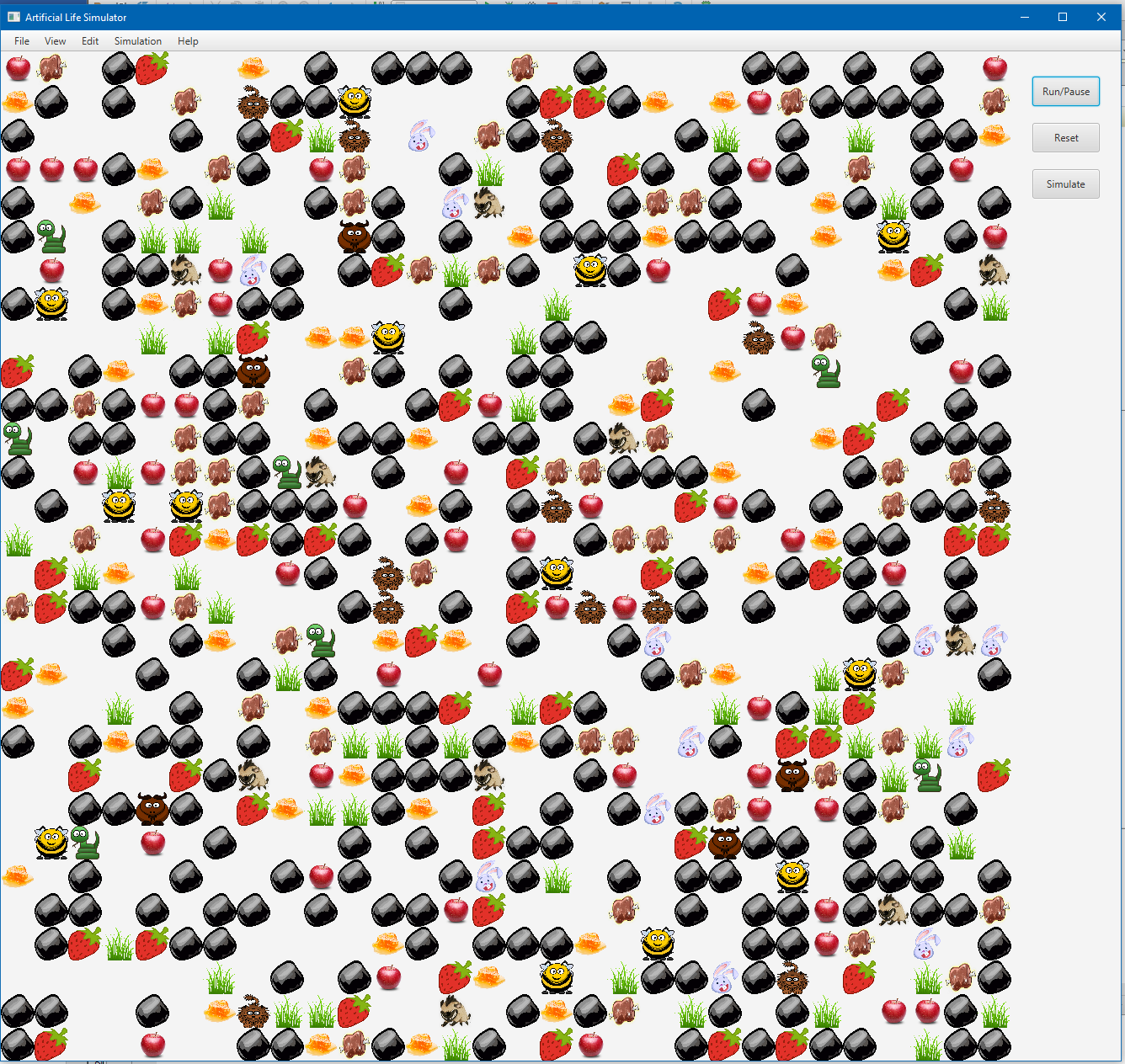
Upon launching the application, the user will be greeted with a blank canvas. You can create a new configuration by going to File> New Configuration. You will now be presented with another window to enter the world’s details. Note how the ‘Enter’ button stays grayed out until all the fields are filled.





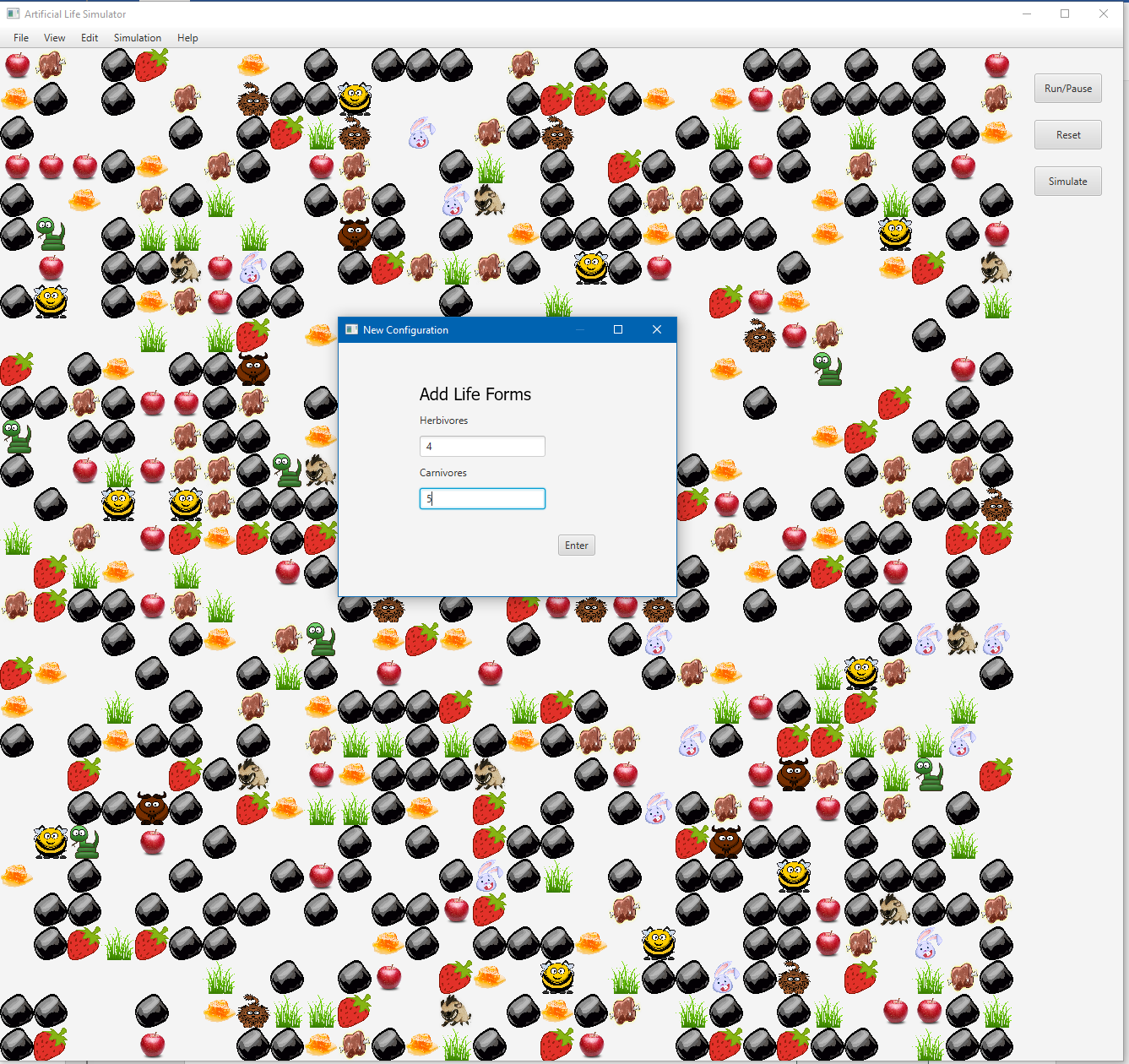
2) **Controlling the simulation**

After clicking enter, you will now be presented with the world. You can make the life forms move, reset, or simulate step by step by either pressing the buttons on the toolbar, or by using the option in the ‘Simulation’ menu.



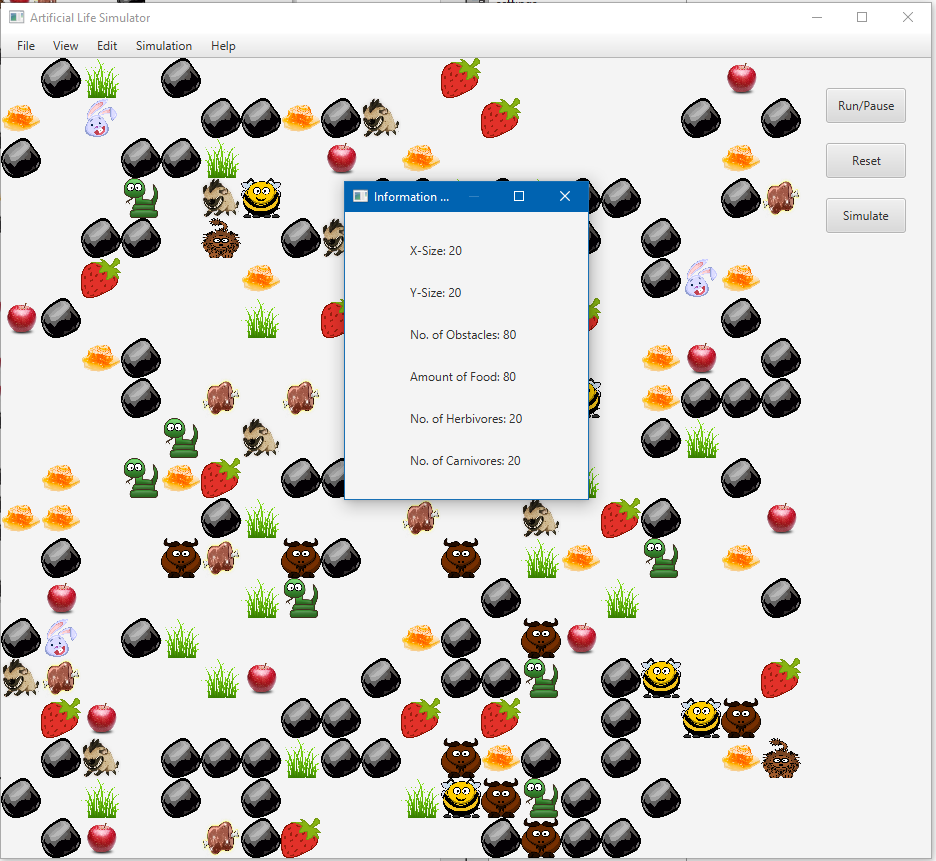
3) **Adding/removing life forms**

You can add and/or remove life forms by going to the Edit menu.



4) **Viewing information about the current configuration**

You can go to View> Display configuration or Display info about the map to view the configuration.



**Personal reflection and conclusion**

Developing this “Artificial Life Simulator” in Java has helped me learn a lot about the process involved in making a java program, and appreciate the efforts of countless developers around the globe developing applications. I have been doing Java for more than a couple of years now, so I understand and appreciate the OOP principles. However, I did learn JavaFx while doing this project, and now have a better understanding of GUI.

Even though JavaFx was a new concept, the documentation and examples provided by Oracle, along with the help of StackOverFlow helped me complete my simulator. Since I made the world, and all the entities in it move with a click of a button by giving them a sense of smell, it technically made me/the user their god while using the simulator!

In conclusion, I’m glad with the way I have done my project. I would have, however, liked to improvise my program further, if it were not for the time constraints that were present, by giving my entities a sense of vision and hearing, and possibly create an early prototype of Skynet. I’m confident that these new skills I picked up while working on this project will help me in the next ‘Android Game Development’ coursework.