**Abstract**

The aim project was to create an artificial life simulator in Java. The project had some requirements but was mostly open to interpretation, the main requirement was to implement a GUI using the JavaFX API. Several weekly practicals session were used to help obtain a basic artificial life simulator than ran in the console, this then had to be developed over Christmas. After the base classes, had been established, it was quite simple to use inheritance to add the different types of entities. I have managed to implement the simulator with a working GUI, I have also added several types of life forms, food and obstacles. The project ran very smoothly and I ran into very little problems.

**Introduction**

In this project, I was required to design and implement an artificial life simulator. Several practicals were set to assist in creating the base of this program, although they were designed to be open to interpretation. These allowed me to create a simple artificial life simulator that ran in the console, which I would then be able to adapt and make my own. The final part of this project was to design a graphical user interface (GUI) for this program and use inheritance to add different types of entities in the world which would be extensions of the base classes written in the practicals. The brief for this was very brief which allowed me to develop the simulator in whatever way I wanted and to make it my own.

**Design**

**GUInterface**

This class deals with the bulk of the JavaFX portion of the program. It is the main class that runs when the program is run. It sets up the scene, with a BorderPane inside it which then has other panes added to each section. One section contains a GridPane which is what represents the world. One contains a TabPane with two tabs that hold information about the entities and world. The bottom section contains a HBox which then contains three buttons. The buttons control the running of the world (Run, Pause/Restart, Reset). More on the use of this class can be found in the Presentation of Completed Project section of this report.

**AWorld**

This class deals with the world in which the entities live. It contains an ArrayList in which all entities are held. It has methods which control setting up the world using a configuration string input by the user, adding/removing entities from the world, modifying entities, the movement of entities, decreasing energy, evolving entities and checking if food has grown back. There is also an enumeration for choosing directions (North, East, South and West), it contains a method for getting a random direction. There are also various functions for getting different information about the world and entities.

The movement of the entities works by detecting what kind of entity is being dealt with at the time, it then creates a list of directions and shuffles them to make it a random order. The smell food function of the entity is then run for each direction, stopping when food is found within the range. Then using the selected direction, the check move function is run which will detect if an entity is able to move to the next space in this direction, if the entity is able then its co-ordinates are changed accordingly. Check move will also dectect what is at the space the entity is moving to, so if it is food, the energy of the entity will be changed and the food removed or the grow timer set.

**AnEntity (Superclass)**

This class is the superclass for any entity in the world (Life forms, food and obstacles). It contains all the base variables that an entity needs to exist in the world:

* String species – The species (name) of the entity.
* Image entIm – The image to be used to represent the entity in the GUI
* int x – The x Co-ordinate of the entity
* int y – The y Co-ordinate of the entity
* int id – The id number of the entity
* AWorld world – The world in which the entity exists

It contains a constructor which allows these variables to be set, which enables the entites to be correctly identified when needed. It also contains getters and setters for each variable. It contains a method called showEntity, which takes a GUInterface as a parameter which calls the show entity function in the GUInterface, which will take the entities image and Co-ordinates to draw it into the world. There are then two more method which are very similar called toString and toText. They return a string containing the species and Co-ordinates of the entity and a string containing all the variables of the entity respectively.

The AnEntity class is also an abstract class which means no instances of it can be made and it can contain abstract methods. This is useful because it means I can have subclasses of AnEntity which do different things and will never accidentally create an instance of AnEntity. I have also used an ArrayList in AWorld which is of type AnEntity but contains instances of all the subclasses of AnEntity. I have also made the getter and setter for the energy of an entity an abstract method, this has allowed me to have it return and set a different variable in each subclass of AnEntity.

**Lifeform (Subclass of AnEntity)**

This class deals with Life forms which are all living entities. It has two extra variables the energy of the entity and the wait time, which is used to determine if a lifeform is in a trap or something else that makes them wait turns. It has a constructor which takes in all the variables from the AnEntity class and then calls the super constructor with these values. It also contains getters and setters for energy and wait.

**FireType (Subclass of LifeForm)**

This class handles lifeforms of FireType, they are entities that like to eat red berries and honey. It has a constructor which calls the super constructor. It contains two functions called smellFood and Evolve.

The smellFood function takes in a direction and range. Depending on the direction it will “smell” up to range spaces away, it first uses the checkObstacle function to see if there is a rock up to range/4 spaces away, this is to make sure the entity doesn’t smell food past this rock, causing it to get stuck forever. It will then use the checkFood function to check up to range spaces away for each kind of food the entity likes. The smellFood function returns true if an entity can smell food in the range and false if It can’t.

The evolve function checks if the energy of an entity is above the threshold for evolving, if it is it then changes the species and image of the entity to its evolved form. The entities I have available now also has a second evolution which the function also checks for in the same way as the first evolution. If I was to add another FireType entity, then I would need to make minor changes to the evolve function, along with making the images for that entity available.

**WaterType and GrassType (subclasses of LifeForm)**

These two classes are pretty much the same as the FireType class, they contain the same variables, constructor and methods, however the smellFood and Evolve functions are slightly different. WaterTypes like to eat blue berries and honey and GrassTypes like black berries and honey, so each of their smellFood methods are changed to take this into account. The Evolve functions are different because they need to change the different types of LifeForm to the correct images and species names.

**Food (Subclass of AnEntity)**

This class deals with all the types of food in the world. It contains two extra variables from AnEntity, nutrition which is how much energy will be added to the entity when it eats the food and time which is used to keep track of when food will grow back. It has a constructor which calls the super constructor and getters and setters for both variables. The subclasses of Food, do not necessarily need to be subclasses, however it makes it easier when creating instances as the nutrition will always be the same for each one and when checking for them.

**Berries (Subclasses of Food)**

There are three berry classes in my program, BlueBerry, RedBerry and BlackBerry. They all contain a constructor which calls the super constructor and sets the nutrition and time. Berries can grow back, so when they’re eaten the time variable gets set to 5, then for each tick in the animation, this value is decreased and when it reaches 0 again, the berries grow back.

**Trash (Subclass of Food)**

Trash is a food item that an entity will not “smell” but if they walk onto the same space as it they will eat it; however, it will damage them instead of giving them energy. It has the same as the berry classes, however the nutrition variable is set to -10 instead of 5. Trash can also not grow back. In the can move function there is a check to see if an entity is moving on to trash which will then change their energy accordingly.

**Honey (Subclass of Food)**

Honey is a Food item that all entities like and will “smell” for. It is also very similar to the other food items, having a nutrition of 5 and not being able to grow back.

**Obstacle (Subclass of AnEntity)**

This class deals with all the obstacles in the world. These are things that will block an entity, they can simply stop an entity walking somewhere or heal/damage them. This class is also an abstract class, this is because I wanted the abstract getEnergy function to be in each of the Obstacle subclasses, as it will need to be different for each one. This class only contains a constructor that calls the super constructor. After completing this project, I realised this class is not necessarily needed, however it makes it easier when checking to see if there is a generic obstacle at a position and not a specific one.

**Center (Subclass of Obstacle)**

Center is essentially a healing station for entities, it will cause them to wait one turn but will give them 5 energy. It contains a variable, healing, which is set to 5. It has a constructor which calls the super constructor and a getter and setter for the healing variable.

**Trap (Subclass of Obstacle)**

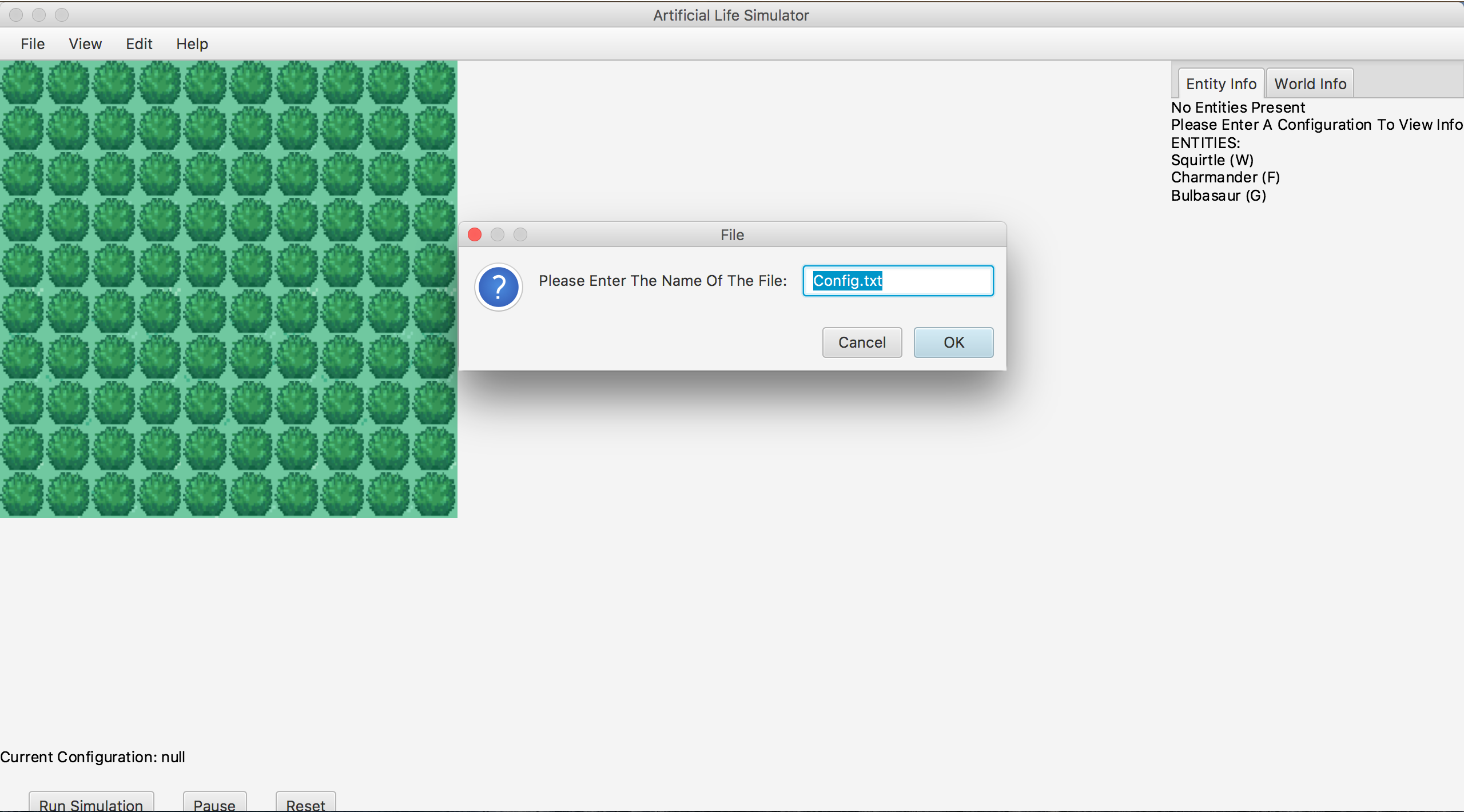
Trap will trap an entity for 5 turns and do damage to them. It contains the same things as Center but the healing variable is a damage variable and is set to -10.

**Rock (Subclass of Obstacle)**

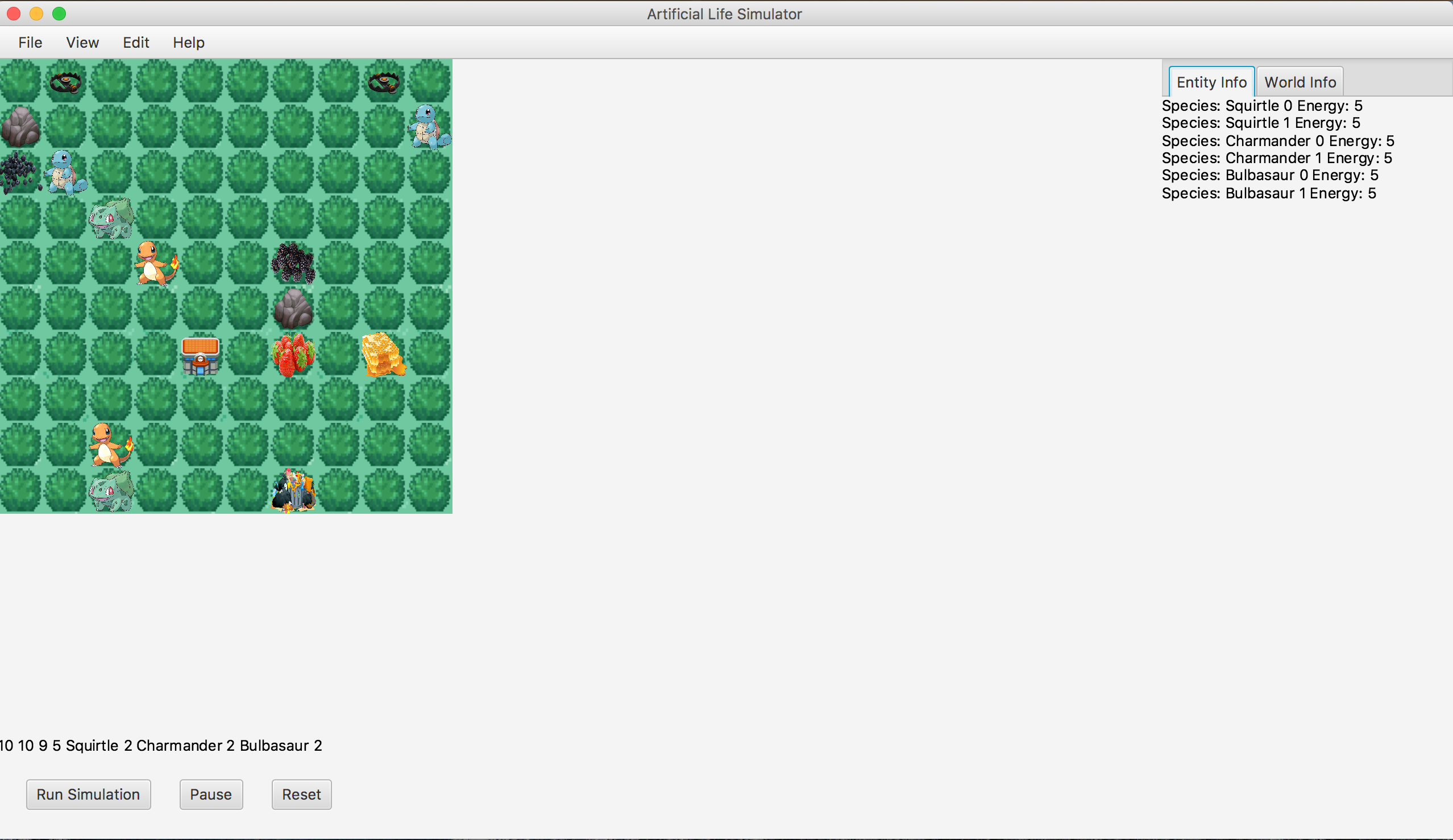
Rock is essentially just a path blocker, an entity is not able to move past a rock. It contains the same things as Center and Trap but doesn’t have a damage/healing variable. Rock must have the get and set energy functions as they are abstract in the AnEntity class, however the getter will just return 0 and the setter will do nothing.

**Presentation of Completed Project**

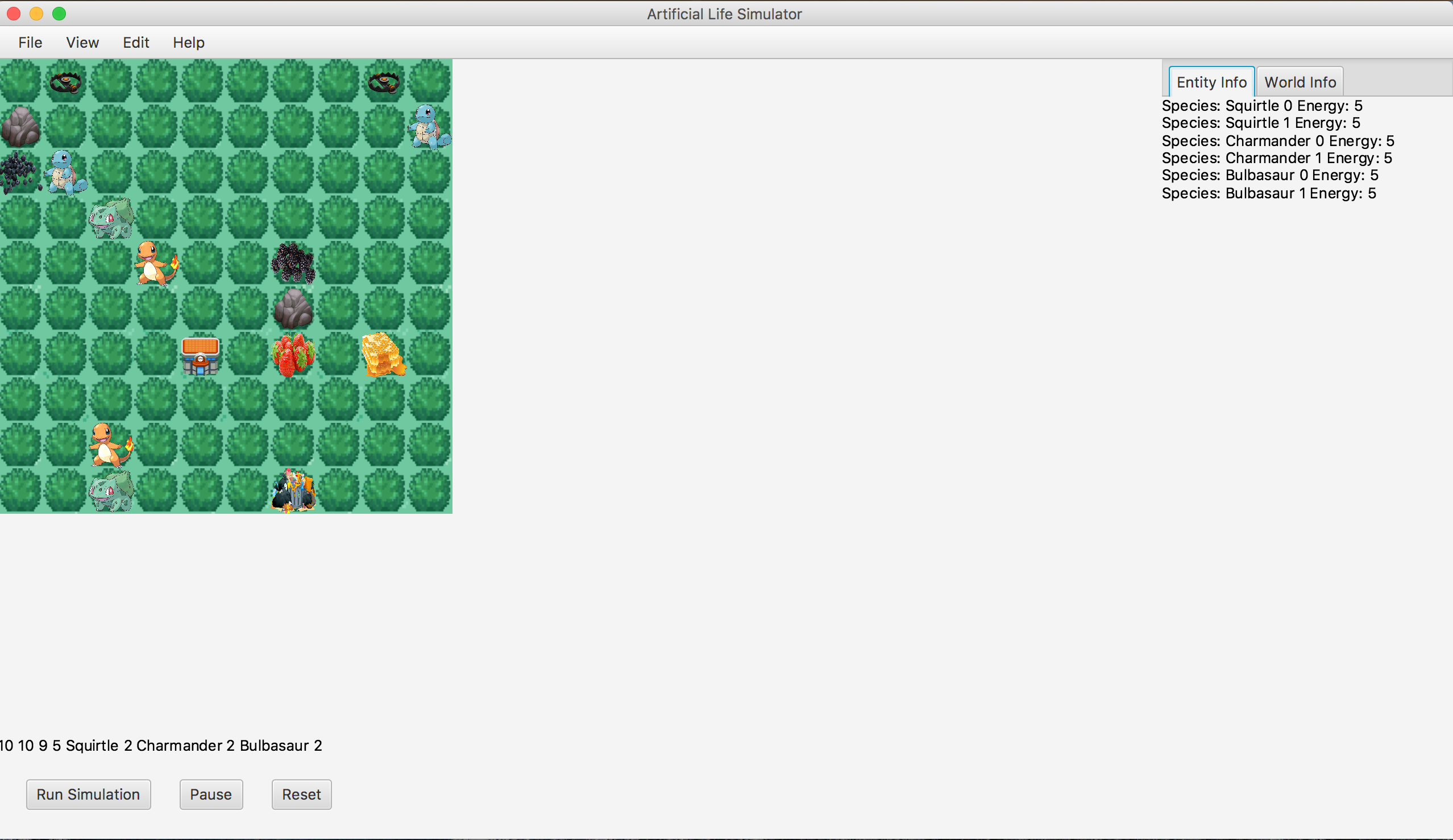
When the program is run, it loads up the GUI with a textinputdialog which asks the user to enter the configuration file they would like to use.

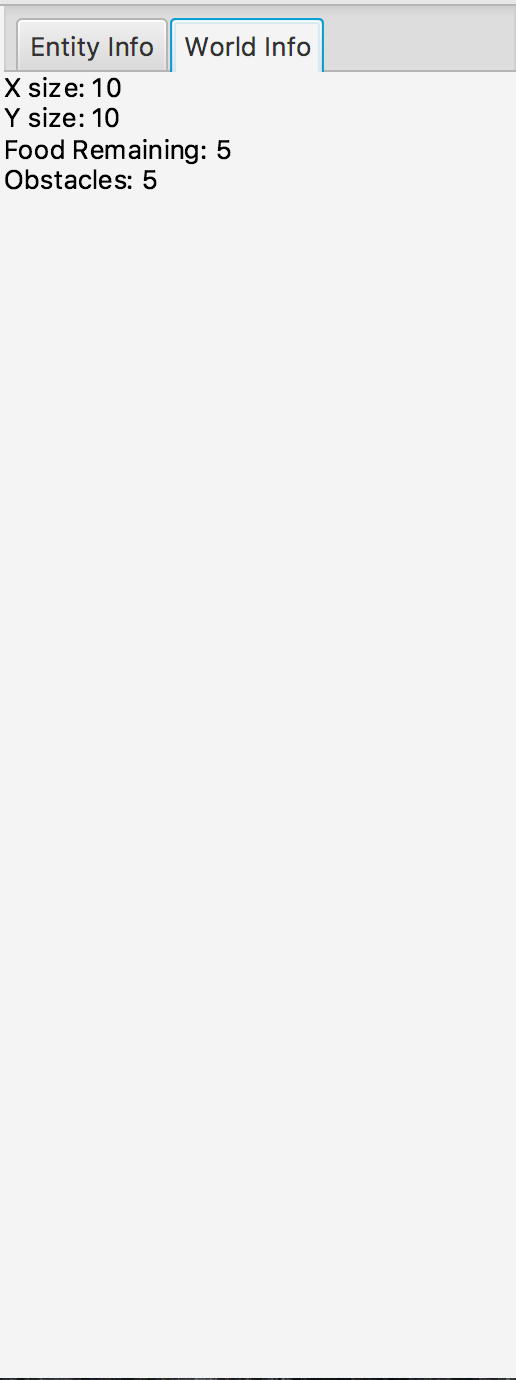
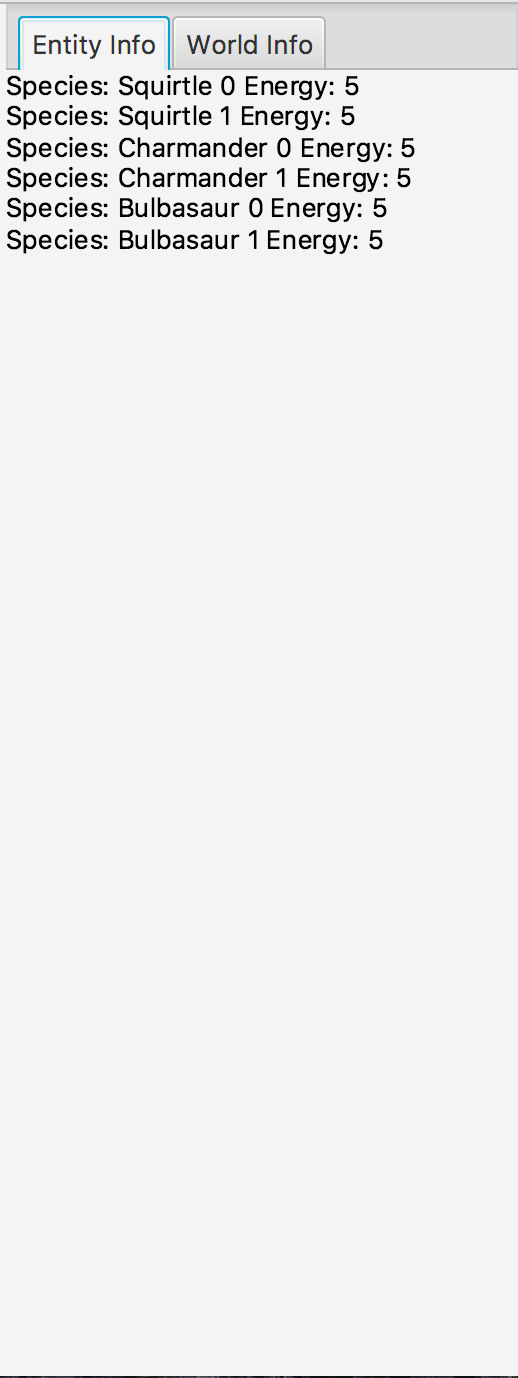


If the file chosen exists and contains a configuration string then that string will be loaded in as the current configuration, otherwise the world will be loaded with no entities. If the file doesn’t exist, then the file will be created, when the user saves a configuration.

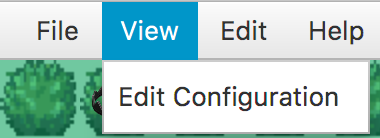
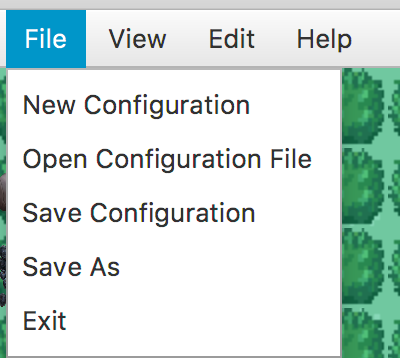


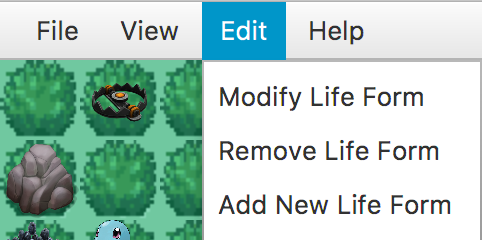
The main section of the GUI is the world in which the entities live and move. This is done using a GridPane that has had the correct image view of the entity added at the entities x,y position. I chose to use a GridPane because it allowed for a simpler x,y Co-ordinate system to be used and it took less time to implement than free form movement.

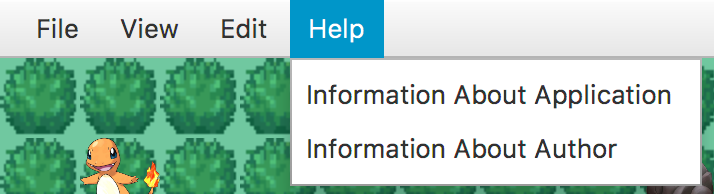


****On the right had side of the GUI is the TabPane, this has two tabs which hold the information about the entities in the world and the information about the world. These are updated constantly while the simulation is running.

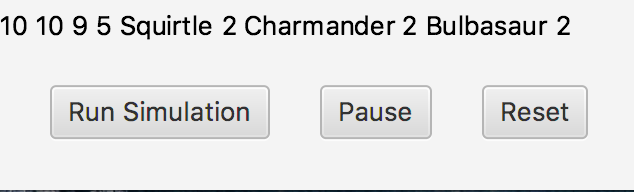
The top of the GUI contains a MenuBar with Menus and MenuItems to control most of the operations for the program. They deal with all the file operations, configurations, modifying and removing entities.

****

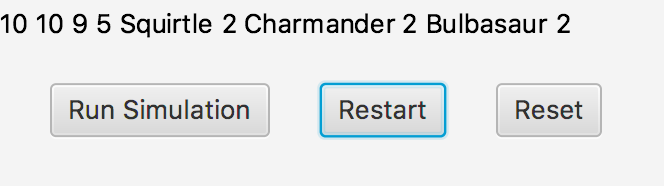
****

****

Underneath the world there is text displaying the current configuration, along with three buttons. The buttons control the simulation of the world, Run, Pause/Restart and Reset.



When the Pause button is pressed the text changes to Reset and vice versa.



**Tests**

The weekly practical sessions were very good as they allowed me to write code to create a simple artificial life simulator and where no too difficult to complete. This meant that when implementing the GUI, I already had most of the code written and working, only needing to make very minor changes. I then needed to add the GUInterface class and any other classes I wished. The GUInterface class was quite tricky to do as I had to learn how to use JavaFX, the hardest part was deciding whether to use a GridPane or a canvas for displaying the world. Adding different types of entities was not very hard as I had the superclasses from the practicals, this meant that not much testing was needed. I also had a clear idea of what I wanted to add and how to do this. The majority of my testing was just writing and implementing code, running the program and then debugging any errors that occurred.

**Discussion**

I decided to use a GridPane to represent the world instead of a canvas or a pane, this was because I found that the grid pane was a lot easier to implement the x,y Co-ordinate system and took a lot less time. I also kept my array for holding all entities as type AnEntity this was so I could hold instances of all the subclasses in one ArrayList. This meant I had to use the instanceof command to find out which class the entity I was looking at was from, I also had to cast the objects from the ArrayList to the correct class when using functions not in AnEntity which meant my code was a bit longer than if I had separate ArrayLists.

**Conclusion**

In conclusion, the project was very successful, I believe I have created a good artificial life simulator that has met almost all the criteria in the brief. My program can show entities living in a world and simulate them moving and finding food, along with many other functionalities like loading configurations from a file and modifying entities. There are however, some things is think could’ve been better. If I was to do this project again I would try to use a canvas to represent the world and use free form movement as opposed to a fixed grid, I did try to do this however, I couldn’t get my head around it and did not want to run out of time to implement other functionalities. I would also have liked to add a life form that would group together and hunt in packs, and for entities to be able to attack each other.