Program Specification

General Description of the Project

Our program will demonstrate the adaptation of bacteria to antibiotics over time through a game that we created. The design of the program will be represented by a screen populated with bacteria, antibiotics and a virus (player). The bacteria and the antibiotics will be randomly placed on the screen (with some constraints). At the beginning of every game, the virus will be placed at the same location and its movement will be controlled by the user (the virus will basically follow the mouse on the screen). To make the bacteria behave as naturally as possible, we will implement the genetic algorithm to make them evolve and, eventually, evade the antibiotics using repulsion forces.

Once the antibiotic is in contact with the bacterium, it will destroy it, and the next generation of bacteria will be created. New generations will evolve through Darwin's heredity principle. Every bacterium will have a property which describes how repulsive the force is between itself and the antibiotic (repulsion coefficient). We will use rejection sampling, which will make it more probable for the bacteria with longer lifetimes to share their "genetic" information with the next generation. Eventually, we are going to observe how every time an antibiotic kills the bacteria, the rest of the bacteria population starts to recognize the antibiotic by evading it. Antibiotic will have its own "health" which will decrease when it does not consume bacteria and vice versa - if it stays without nutrients for a certain time, it will eventually die.

It is also important to note that the virus can also be “killed” by the antibiotic. If the antibiotic actually consumes the virus, the game will be over for the player. The virus itself can consume bacteria, and the more bacteria it consumes the better the score of that player is. The game can also end, and with a win for the player, if the virus ends up outliving the antibiotics.

This game is a clear way to demonstrate how through adaptation, bacteria can survive, evolve, and fight against antibiotics in our world. It also indicates that antibiotics should be used only after careful observation of the patient, as there can be an unpropitious impact on his or her immune system.

Detailed Description of the Design

Here is a list of classes which we plan to create: Main, Vector, Evolution, KinematicObject, Bacterium, Antibiotic, BiologicalDS<E extends KinematicObject>, BiologicalNode<E extends KinematicObject>, DNA, Virus. The following list contains the interfaces used: Consumable, Hungry, WindowInfo. MouseMotionListener interface was also used, along with MouseEvent library. Note: all classes and interface will be in separate files. (with some exceptions)

Main Class

Firstly, let us consider the "Main" class, it will be public and the only class in its file. Main will extend JPanel, which will be used to create the animation. It will also contain public static void main(String[] args) method which will initialize and create a JFrame object giving it all the default values. It will also create an instance of the Main class. Main class will also have one constructor which creates an instance of the Thread class. Furthermore, Main will also contain a nested Runner class which will implement Runnable. It will override its public void run() method, where the naturalSelection() method of the Evolution class will be called. Overall, the general purpose of the Main class will be to create the animation and call the static methods of the Evolution class. It is also important to mention that the Main class will override public void paintComponent(Graphics g). This is the place where all the drawing methods will be called. This is also where we display the scores and the best score.

Vector Class

The purpose of this class is to make it easier for us to work with kinematical and force equations.

It will contain three fields, which are all public and double: x, y, and magnitude. They describe the vector's x, y-component, and magnitude respectively. The class will also have the following methods to perform mathematical operations on the vectors, namely add(Vector _v2), subtract(Vector _v2), multiply(Vector _v2), divide(Vector _v2), getDistance(Vector _v1, Vector _v2). All of the following methods will be public with the return type Vector, and with a Vector object parameter, except for the getDistance(Vector _v1, Vector _v2) method. getDistance(Vector _v1, Vector _v2) will be a public, static method with the return type of double. It will take two parameters of the type Vector and return the distance between them. This class also has one constructor, which takes two double parameters and sets them to the this.x and this.y fields. It also uses the Pythagorean theorem to calculate the magnitude of the vector and sets the result to this.magnitude field.

KinematicObject Class

This is an abstract class which will have methods and fields describing how objects move in space.

The following are the fields of this class: public Vector position, public Vector velocity, public Vector acceleration, public double lifeTime. They semantically show the kinematical properties of moving objects, namely Bacteria, Antibiotics and Virus, which will extend this class. The field lifeTime will just hold the information telling us how long the object has lived. One of the methods of this class will be public void move (double time), which will take time as a parameter and will use object's acceleration and velocity to change its position. Another

method is going to be public void bounce(), which will not allow bacteria and antibiotics to go off the screen by making them bounce back. There is one more important field to note, namely public double mass, which will be used when calculating the force of repulsion between a bacterium and an antibiotic. Note: the masses should be a property of both antibiotics and bacteria, but it is just bacteria who experience the force of repulsion, which depends on the mass of the former as well. The class will also have the following methods to perform mathematical operations on the vectors, namely add(Vector _v2), subtract(Vector _v2), multiply(Vector _v2), divide(Vector _v2), getDistance(Vector _v1, Vector _v2). All of the following methods will be public with the return type Vector, and with a Vector object parameter, except for the getDistance(Vector _v1, Vector _v2) method. getDistance(Vector _v1, Vector _v2) will be a public, static method with the return type of double. It will take two parameters of the type Vector and return the distance between them.

DNA Class

DNA class has the main purpose of containing information which will be delivered to a new bacteria child in the next generation. First of all, it is important to mention that every bacteria will have a field called ”public DNA genome”. The DNA itself will have following member fields: public Vector initialVelocity, public double mass, public double repulsionCoefficient, public double fitness. The first three properties will be sent to the new generation of bacteria mainly because those are important when avoiding the antibiotic. It is especially necessary to note the repulsion coefficient which will have a number between 0 and 1 that will describe the intensity of the repulsion force between the bacteria and the antibiotic (the closer the value is to 1, the more intense the force is going to be). The last property - the fitness - will be assigned by

the static double fittestParent method in the Evolution class (we will talk about this later) which will use the genome of each bacteria, to calculate its natural fitness.

There will be two constructors in this class. Namely: "public DNA ()" and "public DNA (Vector initVal, double mass, double repCoef)". The former will take no parameters, and will use random values to create a DNA instance, and the latter will just set the aforementioned parameters.

There will also be a method whose signature is "public static DNA cross" which will take two parameters of type DNA (for example: "mother" and "father"). This method will take some information from the both parents' DNA instances in order to create a new DNA instance.

Finally, there will be another method called mutate, that will have void as its return type. It will make sure that after some time, DNA instance changes a bit, due to the mutation factor. It will take in a single parameter of type double, that will represent the probability of the change (naturally, it will be a double between 0 and 1).

Bacterium Class

The main purpose of this class is to describe the bacteria objects in our evolution simulation. It will extend the class KinematicObject and will implement the interfaces Consumable and WindowInfo. It will have only one member field: public DNA genome. This field contains information about the properties that will be sent to the next generation - namely mass, initial velocity and the repulsion coefficient.

This class will have two constructors. The first one does not take any parameters and its purpose is to create bacteria with random properties and genomes. The second constructor will take in one parameter of type DNA. It will set the properties of the new generation of bacteria to those that come from the genome itself. The second constructor will be used when creating

bacteria that inherit properties from their parents, and the first one will be used when creating the first, random population of bacteria.

This class will override the method called public void disappear() that will be inherited from the interface Consumable. The purpose of this method is to detect when the bacteria is too close to the antibiotic so that it can destroy the instance that crosses this threshold.

We are also going to have a method called drawBacteria(), which will be public and will have void as its return type. It will take in a Graphics object as a parameter. Its purpose will be to draw an instance of a Bacterium class.

Furthermore, we will have a method called isIn(Antibiotic anti1) whose return type is boolean, which takes the parameter of type Antibiotic. The purpose of this method is to return true or false depending on whether the bacteria is in the area of the antibiotic.

Another method that is going to be part of this class will be called public static Bacteria crossBacteria(Bacteria bact1, Bacteria bact2). This method will use the static method cross from the DNA class to create a new bacteria with evolved DNA.

The method we added is: public void changeAcceleration(). This method finds the antibiotics that is the closest to our bacterium instance and changes the acceleration of the bacterium accordingly.

Antibiotic Class

This class will describe how antibiotics will behave in our simulation. It will extend KinematicObject class and will implement Hungry interface.

Antibiotic Class will have a member field called radius of type double. This member field will directly influence the size of the antibiotic instance. It will also have a field called lifeTime that will be of type int and will be public. It will show us how much time the antibiotic has left if

it does not consume a bacteria. It will also have fields called consumedPopulation (it will tell us how many bacteria the antibiotic has consumed) and isExtinct, which will be a boolean variable that will tell us if the antibiotic is still alive.

The class will have one constructor which will take a parameter called radius of type double. The constructor will determine the mass of the antibiotic using its radius.

The Antibiotic class will override a method called eat, which will come from the Hungry interface. It will take a parameter of type Bacterium. It will be called by isIn(Antibiotic anti1), which is the method of the Bacterium class, to determine if the antibiotic should destroy the bacteria instance. Naturally, if the method returns "true" the instance of the Bacterium class will be deleted and vice versa.

We will also need a method called drawAntibiotic(Graphics g) which will take in a graphics object and will basically present an instance of the bacteria class.

Evolution Class

It will have five member fields. One is going to be public static BiologicalDS<Bacterium> bacteriaPopulation. This one is going to be a data structure that stores all of the bacteria population. The next one is going to be public static BiologicalDS<Antibiotic> antibioticsPopulation which is exactly analogous to the first one. It is going to store the whole population of the antibiotics. We will also have: 1) public static final int ***BAC_POPULATION*** - this will tell us how many instances of the Bacterium class we will have in the beginning; 2) public static final int ***ANTI_POPULATION*** - this is analogous to the previous field insofar that it tells us how many antibiotics we have in the game; 3) public static Virus hiv - this is going to be the field that represents the instance of the virus class - it will be the only instance and, as we mentioned before, it will be controlled by the user.

This class will have a number of methods. Firstly, the method called public static void randomlyPopulate() which will, naturally, randomly create a number of Bacterium instances on the screen (it will randomly populate our simulation). It also introduces the antibiotics into our simulation.

We will have a method called private static Bacterium fittestParent(). This method will use rejection sampling - it will choose a parent which (with more probability) has a longer lifeTime.

The naturalSelection() method will call fittestParent() method twice to find the father and the mother with potentially longer lifeTime(s). It will then call the cross method of the DNA class to perform a process which is known as the cross-over to combine the genomes of the parents in order to create a child bacterium. It will also eventually call the mutate() method of the bacterium's genome property which will have a small probability of changing the genome of the child bacteria in order to create some variation in our population.

The next two methods are called public static drawAllBacteria(Graphics g) and public static drawAllAntibiotics(Graphics g). They will, naturally, be called in the paintComponent and they will draw the given elements in every frame.

Another method that we will have is called: public static void disappearAllConsumed(). This method will naturally make sure that all consumed bacterium instances disappear from the screen.

Virus Class

The virus class describes an object which will be controlled by the player. The object can consume bacteria; therefore, the class will implement Hungry interface. The virus can also be

consumed by the antibiotics, which means that it will have to implement the interface Consumable as well. It will also be possible for the player to control the virus by using the mouse, so the class will also implement the MouseMotionListener interface.

This class will extend KinematicObject class because our virus instance will obey all the kinematic motion laws of the simulation.

The fields of the class are the following: 1) public static final double ***VIRUS_RADIUS*** - the constant value for the radius of the virus; 2) public static final double ***SPEED*** - the constant value for the speed of the virus; 3) public static int *consumedPopulation* - the variable that represents the number of bacterium instances consumed by the virus; 4) public static boolean *isExtinct* - the variable that tells us whether the virus has been consumed by the antibiotic - if it acquires the value *true* the simulation stops, 5) public Color virusColor - naturally, the field just stores the color of the virus instance.

First, it is important to note that there is only one constructor that does not take in any arguments. It sets the position of the virus to a certain place every time - the top left corner (we are doing this to avoid it being consumed by the antibiotics as soon as it appears on the screen).

These are the methods of the class: 1) private void followMouse(MouseEvent e) - this method will make sure that the virus is responsive to the motion of the mouse made by the user. That motion will itself change the value and the direction of the velocity vector for the virus, thus causing movement and apparent following of the mouse on the screen. This method is called in a different method - namely public void mouseMoved(MouseEvent e). We did this to make sure that the event will be handled every time the mouse is moved. 2) public boolean isIn() - this method comes from the Consumable interface. It returns true if the virus comes in contact with the antibiotics. 3) public void disappear() - the method will check to see if the virus is in contact

with antibiotics by calling the isIn() method. If isIn() returns *true* the method will make sure that the simulation stops by making the value of isExtinct field *true*. 4) public void eat() - this method makes sure to increment the value of the consumedPopulation thus letting us know that the virus has consumed another bacterium instance. 5) public void drawVirus(Graphics g) - the method naturally draws the virus. It is called in the paintComponent method of the Main class.

Progress Class

The class will be responsible for saving and loading the best score to and from a .txt file.

It has just two methods which are both called in the Main class. 1) public static int loadScoreFromFile(String fileName) - the method will make sure that the score appearing on the screen every time the player enters the game is actually the best score that was stored in the .txt file. 2) public static void saveScoreToFile(int bestScore, String fileName) - the method will save the new score to the same .txt file if it is, in fact, larger than the previous one.

BiologicalNode<E extends KinematicObject> Class

This class will be generic. It will work just with Antibiotic instances and Bacterium instances because it is only those two classes which implement the Living interface. The purpose of this class is to create a node where the instances of the aforementioned two classes can be stored. This class will contain two member fields. The first one will be public BiologicalNode<E> next. The second will be public E biologicalElement. These two fields will have a similar purpose to the ones that were in the user-defined data structures which we were creating in the class.

Its constructor will take one parameter of type E, setting it to this.biologicalElement.

BiologicalDS<E extends KinematicObject> Class

This class will have the following fields: private BiologicalNode<E> end, which will be set to null at first. It will also contain the length member field. It will give us the current number of the biological nodes in our data structure. Another field that we will have will be: public double lifeTimeSum - this field will store the total sum of all bacteria lifetimes in order to find the fraction of the combined lifeTime that each bacterium will receive.

The class will contain the method public void append(E element1) which will add another node to our data structure. It will have a method public void pop() which will remove the last element from our data structure. It will also have a method public void remove (int index) which will remove the element that has the given index. It will also have a method public E peek() which will return the element of the last node in our data structure.

Interface Consumable

This interface will contain two methods called "public void disappear()" and "public boolean isIn()". The former will be overridden by those objects that can disappear from the screen (bacteria and antibiotics). The latter will determine whether an object is in the range of the other object. It will mainly be used to determine whether virus or bacteria is inside an antibiotic.

Interface Hungry

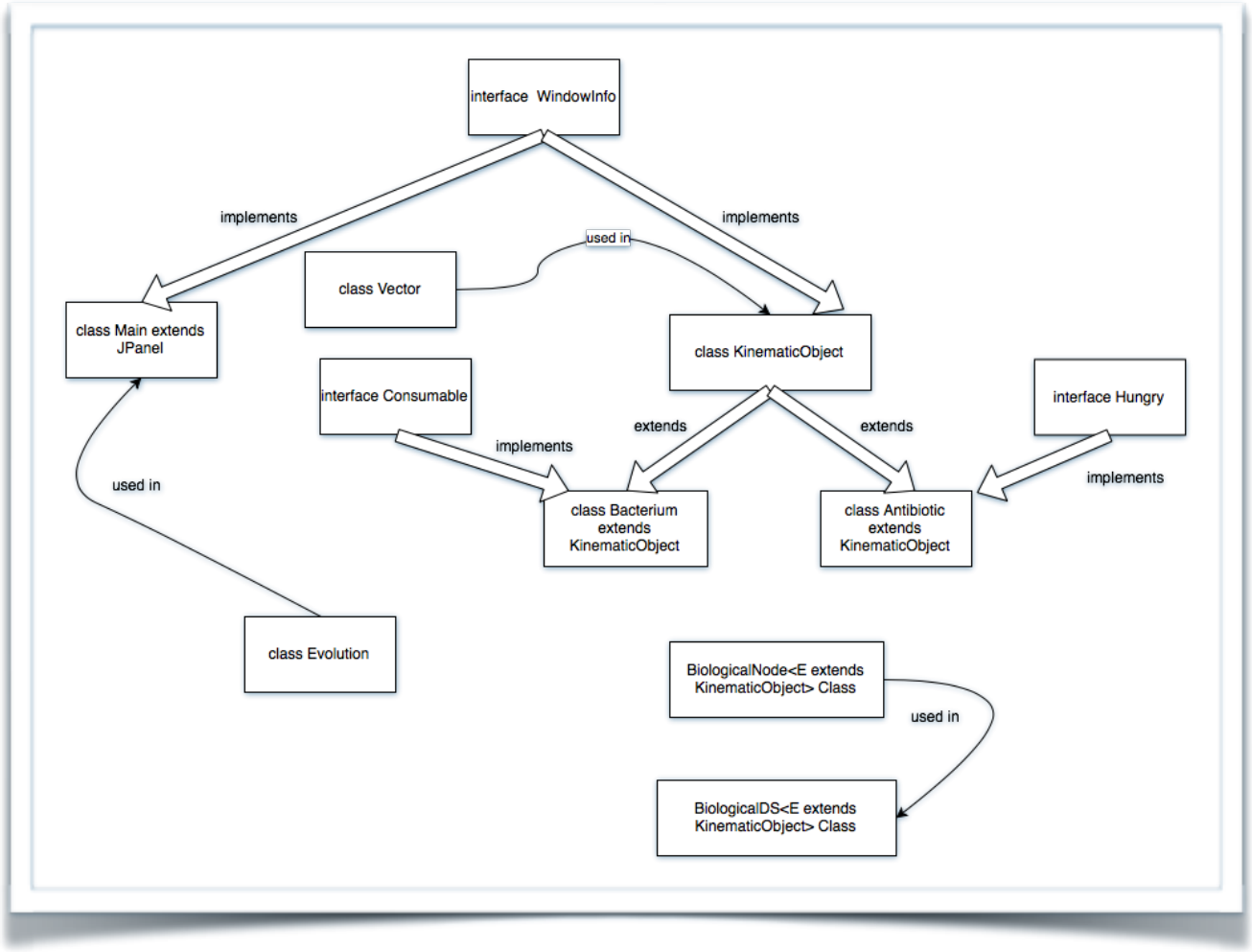
This interface will contain a single method called "public void eat()". This method will be overridden in every class that describes that can consume others.(e.g. Antibiotic class instances)

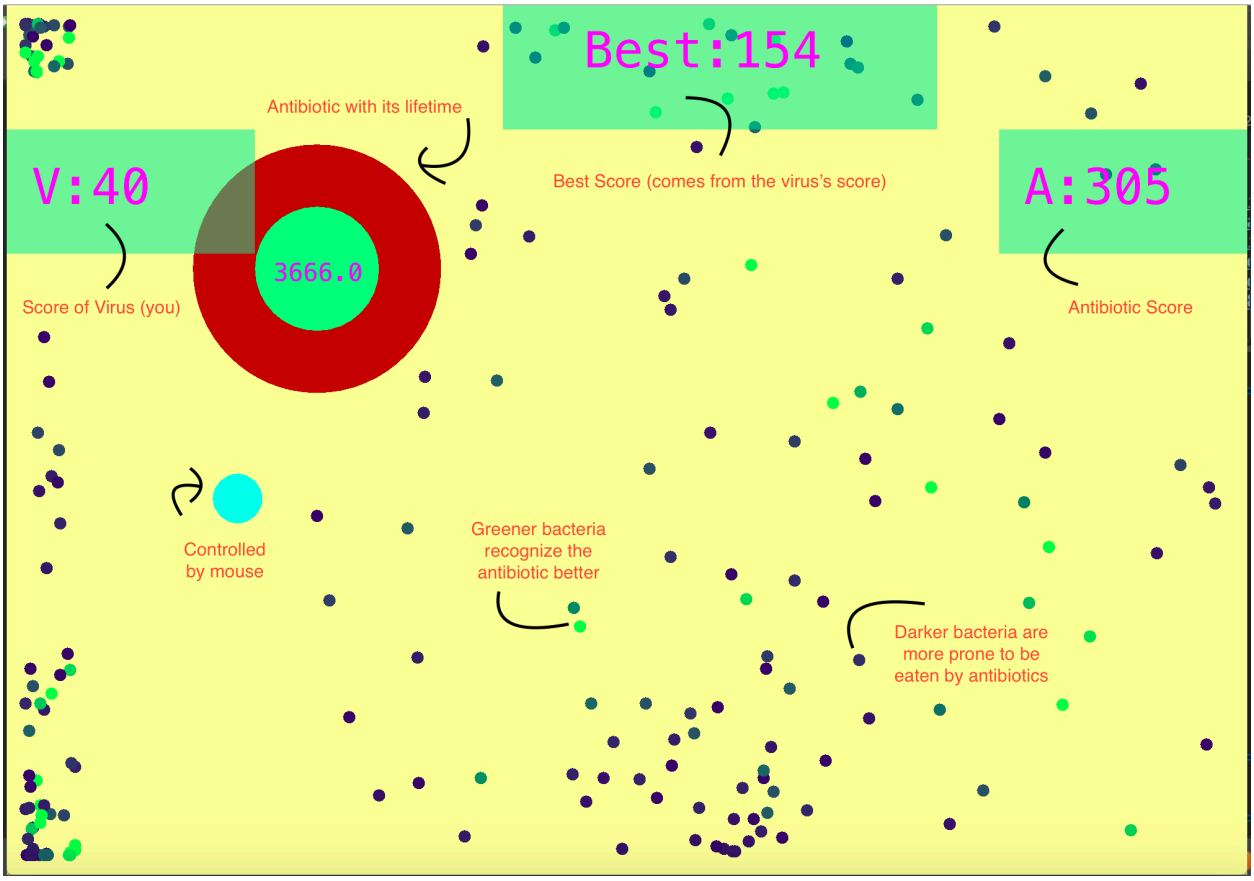
Interface WindowInfo

This interface will contain two static and final member fields of type double. Those will be width and height - they will describe the dimensions of the screen. This interface will be implemented by all the classes where those dimensions are necessary.

Diagram of Class Relationship

Note: the diagram remains the same. It is just a Virus class that is added. It extends Kinematic Object class and implements Consumable and Hungry interfaces




Best:154
Antibiotic with its lifetime
V:40
A:305
Best Score (comes from the virus's score)
3666.0
Score of Virus (you)
Antibiotic Score
Controlled
by mouse
Greener bacteria
recognize the
antibiotic better
Darker bacteria are
more prone to be
eaten by antibiotics