Game of Life

(Coursework 3)

Talal Alohali (21130307)

Christopher Herre (22001776)

Life Forms

Mycoplasma:

This cell is represented using the ORANGE or YELLOW colour on the grid. The cell has a basic set of rules concerning whether it lives or dies from state to state, but it also has a separate set of rules for when it is diseased. The basic rules of the cell for living are:

- if the cell is living and has two or three living neighbours, then it will continue to live.
- if the cell is dead and has exactly three living neighbours, then it will come back to life.

The disease rules of the cell are:

- if the cell has more than four diseased neighbours then it will 'catch' the disease, otherwise there is a 25% chance of it becoming diseased.
- if the cell is diseased then it will turn YELLOW, but so long as it has more than one neighbour it will continue to live.
- if the cell is diseased and has more than five living neighbours, then it will be cured.

Lactobacillus:

This cell is represented using the GREEN or BLUE colour and is non-deterministic. The cell has a basic set of rules concerning whether it lives or dies from state to state, but it also has a separate set of rules for when it is diseased. Each of these rules are implemented based on random chance, with the likelihood of each behaviour listed below. The basic rules of the cell for living are:

- if the cell is living and has less than three (50%), exactly eight (35%), or not exactly four (15%) living neighbours, then it will continue to live.
- if the cell is dead and has zero (75%) or more than six (25%) living neighbours, then it will come back to life.

The disease rules of the cell are:

- if the cell has more than four diseased neighbours then it will 'catch' the disease, otherwise there is a 10% chance of it becoming diseased.
- if the cell is diseased then it will turn BLUE, but so long as it has more than two living neighbours it will continue to live (50%)
- if the cell is diseased and has more than 4 living neighbours, then it will be cured (50%)

Cyanobacteria:

This cell is represented using the RED or PINK colour on the grid. The cell has a basic set of rules concerning whether it lives or dies from state to state, but it also has a separate set of rules for when it is diseased. These rule sets change depending on the number of generations that have passed, with the condition for each behaviour listed below. This cell also has the potential to form a mutualistic symbiosis with the *Cyphobasidiales* cell, but only when both cells are alive. The basic rules of the cell for living are:

- if the cell is living and has more than seven (gen < 7007) or more than six (gen >= 7007) living neighbours, then it will continue to live
- if the cell is dead and the number of living neighbours is a multiple of three (gen % 4 == 0) or more than three (gen % 4 != 0), then it will come back to life

The disease rules for the cell are:

- if the cell has more than seven neighbours then it will catch the disease, otherwise there is a 35% chance of it becoming diseased.
- if the cell is diseased then it will turn PINK, but will continue to live if (gen < 1000), if (gen >= 1000 && gen % 50 = 0) or if the number of living neighbours is less than two or more than 6 (gen >= 1000 && gen % 50 != 0)
- if the cell is diseased and has between two and six (gen % 5 == 0) or between one and four (gen % 5 != 0) living neighbours, then it will be cured

The Symbiotic rules for the cells are:

- if the cell is diseased, alive and in symbiosis, then it will continue to live indefinitely, but there is no effect on curing it.
- if the cell is not diseased, is alive, is in symbiosis and has more than two living neighbours, then it will continue to live.

Cyphobasidiales:

This cell is represented using the MAGENTA colour on the grid. The cell has a basic set of rules concerning whether it lives or dies from state to state. This cell cannot become diseased but does have the potential to form a mutualistic symbiosis with the *Cyanobacteria* cell, but only when both cells are alive. The basic rules of the cell for living are:

- if the cell is alive and has more than 5 living neighbours, then it will continue to live.
- if the cell is dead and has more than three living neighbours, then it will come back to life.

The Symbiotic rules for the cells are:

- if the cell is alive and has more than zero living neighbours, then it will continue to live.

<u>Base Tasks</u>

Life and death of the Mycoplasma (1-4):

Classes Changed: *Mycoplasma*

(1-3). For the first three challenge tasks, all that was required was to add an *if* statement, which would *setNextState(true)* if *neighbours.size()* was equal to two or three.

Classes Changed: *Mycoplasma*

(4). For the last of these initial tasks, we added an *else* clause to the outermost *if* statement, so that we could add conditions for when the cell was already dead. Within the *else*, we added and *if* statement that *setNextState(true)* if *neighbours.size()* was equal to exactly three.

New Life Incorporating Colour and Time (5-6):

Classes Changed: *Mycoplasma, Lactobacillus, Cyanobacteria, Cyphobasidiales*

(5). All our cells incorporate colour in their rule set. *Mycoplasma, Lactobacillus* and *Cyanobacteria* all change colour depending on whether they are diseased or not (*see 'Life Forms' & 'Challenge Tasks: Incorporating Disease' for specifics*). This was done by using the *setColor()* method after a cell became diseased or was cured. In addition, *Cyanobacteria* and *Cyphobasidiales* use colour to determine if there is a living neighbour that they can form a symbiotic bond with. This was done using a *for-each* loop, checking if a cell existed in the list of neighbours that *equals()* the specified colour of the cell that could form the bond (*see 'Life Forms' & 'Challenge Tasks: Incorporating Symbiosis' for specifics*)

Classes Changed: *Cyanobacteria*

(6). The *Cyanobacteria* cell is the only cell that uses the number of generations to affect its behaviour. To begin, I created a new, *protected* variable *generation* in the superclass *Cell*. This meant that every cell could now track the number of generations in the simulation, with the variable being updated every time *act()* is called. This meant that I could then incorporate it in the *if* statements of the disease and basic rule sets *(see 'Life Forms: Cyanobacteria' for specifics)*.

<u>Challenge Tasks</u>

Incorporating Non-determinism:

<u>Classes Changed:</u> *Cell, Lactobacillus*

To introduce a non-deterministic nature to the cell, we initialised a *protected* variable *random* that could be used in any cell. In the *act()* method we created a new random number *ruleRandom* between 0 and 99. This was then used in various *if* statements throughout the rule set of the cell (*see 'Life Forms: Lactobacillus' for specifics)*. The basic concept was to use numeric boundaries (i.e. <50 [50%] or <80 [30%] or <100 [20%]) and depending on the boundary the number fell in, a different rule would be implemented. Given that there is an equal chance of any number between 0 and 99 appearing: the large the boundary, the larger the probability of the rule being chosen.

Incorporating Disease:

<u>Classes Changed:</u> *Cell*, *Mycoplasma*, *Lactobacillus*, *Cyanobacteria*

Implementing disease was a little trickier because we had to consider, getting the disease, behaviour while diseased and becoming cured of the disease. It was clear to us we needed a way to store whether a cell was diseased or not, so we added two variables and two methods to the *Cell* class (all cells have a state of being diseased or not). These were *disease, nextDisease* (both Boolean and intialised to false), *hasDisease()* (return if a cell was diseased or not) and *setDiseaseState()* (take boolean value and set 'next disease state' equal to it). We also had to modify the *updateState()* method, so the disease state of the cell was updated at the same time as the alive state. Disease In this way, *disease* works very similarly to *alive*, making the implementation much easier to understand.

Catching the disease has two methods: neighbours that have the disease or random chance. Initialising variables *diseaseRandom* (0-99) and *count* (initialise to 0), we used a *for-each* loop to cycle *neighbours*. Imbedded in this was an *if* statement, checking whether each cell had a disease or not, and incrementing *count* if true. If the count met a certain condition – different for each cell – then *setDiseaseState(true)* and *setColor(____)* was called. This updates the diseased state of the cell and changes its colour on the GUI, so that it is obvious. However, this can also happen if the *diseaseRandom* is within a certain numeric boundary, giving it an additional, random chance of becoming diseased every generation.

Behaviours and curation were both implemented within a new *if* statement that wrapped completely around the one concerning *isAlive()*. This meant that if the cell was diseased, it ignored the standard rule set and followed a different one. Within this, there are several specified conditions for the cell's continued survival and potential curation. If it fulfills the 'curing conditions' then: *setNextState(true), setDiseaseState(false)* and *setColor('original').*

Incorporating Symbiosis:

<u>Classes Changed:</u> *Cyanobacteria*, *Cyphobasidiales*

This worked differently in each of the cells, but in both, it made the cell more likely to survive, by reducing the number of living neighbours the cell required to remain alive (the relationship can't occur if one is dead). In *Cyanobacteria*, the cell uses the *for-each* loop already being used to implement disease and adds another *if* statement that evaluates true if there is a *Cyphobasidiales* cell in *neighbours* (using color as a marker). Upon evaluating true, a *symbiosis* Boolean flag is set to true, and the conditions implemented in the disease rule set and the basic rule set then come into play. These conditions are at the top of the *if* statements, so that they are looked at first. In *Cyphobasidiales*, the cell uses the same method to check for a *Cyanobacteria* cell, but insteas of using a flag, it uses a *neighbourReq* integer variable to update the number of neighbours it requires to remain alive.