4CCS1PPA Coursework 3: Game of Life

Background pattern

Description automatically generatedCoursework completed by: Caleb Chan [caleb.chan@kcl.ac.uk](mailto:caleb.chan@kcl.ac.uk) and Alexander Wickman [alexander.wickman@kcl.ac.uk](mailto:alexander.wickman@kcl.ac.uk)

# Life Forms

**Mycoplasma**

* Follows rules outlined in base task. If cell has fewer than two neighbours, it will die. If the cell has two or three live neighbours, it will live on the next generation. If the cell has more than three neighbours, it will die. Any dead cell that is dead and has exactly three neighbours, will come alive.

**Influenza**

* Becomes alive after a random number of generation cycles after initial creation. Once alive, it will remain alive only if it has three or more living neighbours. If it has two or less neighbours, it will die. It will also die if at least one neighbour is a white blood cell. If it is dead, it will come back to life if it has at least one neighbour and none of its neighbours are white blood cells.

**Flavobacterium**

* Changes colour depending on the number of neighbours. If it has no neighbours, it dies. If it has 1 or more neighbours, it continues to live. If it has 1 neighbour it changes colour to red, if it has 2 or more it changes to pink. If it is currently dead and there are exactly three neighbours, it comes back to life.

**White Blood Cell**

* Under the assumption that white blood cell only recognises and targets the influenza cell and the Devourer cell. White blood cell only remains alive if it has 2 or three neighbours. For any other amount, it will die. If it is currently dead, two or more of its neighbours must be influenza or devourer in order to come alive.

**Melanocyte**

* Nondeterministic cell that executes a rule from a set of rules with a given probability. Contains one ruleset for whilst it is alive and one ruleset whilst it is dead. Each rule has an assigned probability to which it will occur in a given generation.

**Devourer**

* Devourer is a cell that can kill(devour) other cells. If the Devourer has 1 neighbour and is alive, it will devour it. It cannot devour another cell if it has more than 1 neighbour. If the Devourer is dead and has 1 neighbour, it will come back to life. If the Devourer has no neighbours or has not devoured in 5 generations, it will die. Devourers can kill each other.

# Challenge Tasks:

1. **Non-Deterministic Cell**

In our project we chose a random cell to use for a nondeterministic cell. This is the melanocyte cell. Within the act() method, a random generator generates a double, between 0 and 1 representing a percentage. Using this random double, we can proportion probabilities to a set of rules, which simulate a non-deterministic cell. For the rules, we decided to use a mix of colour changes, rules that cause the cell to die and rules that cause the cell to live on. We have also included a separate set of rules for when the cell is dead and may come back to life.

Ruleset whilst alive:

Assumed to always die unless rule says otherwise.

10% - Changes colour to blue.

10% - Changes colour to black (default colour).

20% - If two or more of its neighbours is mycoplasma it will die.

30% - If it has exactly 1 neighbour it will live on.

30% - If it has exactly 3 neighbours it will live on.

Ruleset whilst dead:

40% - Comes back to alive in presence of white blood cell.

60% - Comes back to alive with 3 or more neighbours.

1. **Symbiosis**

Symbiosis achieved by white blood cells and influenza cells. The nature of a white blood cell is that it will only target cells that are recognised by it. Hence, we are under the assumption that the white blood cell targets the influenza cell (parasitic relationship), the influenza’s rules set ensure that in the presence of a white blood cell, it will die. Likewise, in the white blood cell, it will come back to life to kill living influenza cells or devourer cells. Whilst the white blood cell is living, at any point if one of its neighbours is an influenza cell or the devourer cell, it kills that cell one by one. These rules were implemented in both the act() methods of the Whiteblood class and the Influenza class.

1. **Disease**

Disease is an infection which is inherent to all cells. When a cell is infected, the disease stays dormant for the current generation. In the next generation, it kills the host (overrides the act method) and infects all neighbouring cells. If the cell dies before the next generation, the neighbouring cells do not become infected. Once a cell dies it loses its infection.

I decided to implement Disease to be compatible with all cell classes without the subclasses needing to be altered in any way. However, in order to change how the cell acts, I had to alter the Simulator and the Cell classes. I did this by creating two new variables: willInfect and isInfected. willInfect represents the Disease’s initial infection in the first generation. This has no effect on how the cell acts. Then, during the state change, the isInfected variable becomes true. This now affects all cells all at once and produces deterministic behaviour. When isInfected is true, the infected action happens.

The infection act consists of two parts: killing the host cell and spreading the disease to other cells. In order to make sure that the other cells are always infected, regardless of act order, the method for determining willInfect can only set the variable to true and cannot override infection from another cell.

1. **Devourer**

Devourer is a cell that can kill(devour) other cells. If the Devourer has 1 neighbour and is alive, it will devour it. It cannot devour another cell if it has more than 1 neighbour. If the Devourer is dead and has 1 neighbour, it will come back to life. If the Devourer has no neighbours or has not devoured in 5 generations, it will die. Devourers can kill each other.

Implementing the rules on the devourer itself were very straight forward, using simple neighbour logic. However, it was more difficult to determine how a neighbouring cell would die. This is because through the normal method of setting nextAlive, if the neighbouring cell had not acted yet, it could override the death. Therefore, I decided to use a new variable in the abstract class isDevoured. Then, during the status update, apply all of the devour deaths then so that all deaths were deterministic. Therefore, the devourer cell does not directly alter the nextAlive state of other cells.