Covid Antibody Response and Infection Neutralization (CARIN)

Amidst the pandemic, the human bodies have responded! Upon the arrival of Covid viruses, *you as a player* will be controlling the immune system to eliminate the incoming viruses before they destroy your body.

Overview of CARIN

A human body is an $m×n$ field of cells, where m and n are as specified in the **configuration file**. At each time unit, a virus can spawn at one of the empty cells. After the time unit a virus has spawned, the virus will behave according to its genetic code. A virus may be eliminated by an antibody, but can also harm an antibody. It's a matter of survival!

You are in control of your antibodies. At any time unit, you can choose to place a new antibody in an empty cell. Alternatively, you can choose to move an existing antibody to another location. But antibodies don't come for free. At the beginning of time, your body has *antibody credits* that you can spend, as specified in the **configuration file**. Every time you place a new antibody, you lose some antibody credits. You can gain additional antibody credits by having an antibody destroy a virus.

Similar to viruses, antibodies wait until the next time unit after placement before following its genetic code.

In a human body, there should be at least three kinds of viruses that can be spawned, and at least three kinds of antibodies that can be placed. Each virus has some health. Its health will decrease if an antibody attacks it. A virus dies if its health reaches zero. Nevertheless, a virus can gain some health by attacking an antibody. Similarly, each antibody has some health, which can increase by destroying a virus, and can decrease when a virus attacks it. An antibody will turn into a brand-new virus if its health reaches zero. The genetic code of the antibody-turned-virus is that of the virus that destroyed it.

If you think that the existing antibodies don't work as expected, you can choose to move one of them in each time unit. For each move, the antibody will lose some health. This can be useful when you do not have enough antibody credits to place a new antibody.

The amounts of gains and losses mentioned in the preceding paragraphs are as specified in the **configuration file**.

The pandemic is over when all viruses have been eliminated by antibodies, or when your body no longer has antibodies. Which side will rule the human race?

Setup

Before the human body is exposed to the world, the immune system should generate at least three kinds of viruses and three kinds of antibodies that could appear in the body. The players can choose to provide their own genetic codes if desired. If the players choose not to provide genetic codes, the immune system should load additional, default genetic codes.

Finally, the immune system should read the configuration file for necessary parameters.

Grammar for genetic codes

Genetic codes of viruses and antibodies are governed by the following grammar. Terminal symbols are written in `monospace font`; nonterminal symbols are written in *italics*.

Program → *Statement*+
Statement → *Command* | *BlockStatement* | *IfStatement* | *WhileStatement*
Command → *AssignmentStatement* | *SensorCommand* | *ActionCommand*
AssignmentStatement → `<identifier>` `=` *Expression*
SensorCommand → `virus` | `antibody` | `nearby` *Direction*
ActionCommand → *MoveCommand* | *AttackCommand*
MoveCommand → `move` *Direction*
AttackCommand → `shoot` *Direction*
Direction → `left` | `right` | `up` | `down` | `upleft` | `upright` | `downleft` | `downright`
BlockStatement → `{` *Statement** `}`
IfStatement → `if` `(` *Expression* `)` `then` *Statement* `else` *Statement*
WhileStatement → `while` `(` *Expression* `)` *Statement*
Expression → *Expression* `+` *Term* | *Expression* `-` *Term* | *Term*
Term → *Term* `*` *Factor* | *Term* `/` *Factor* | *Term* `%` *Factor* | *Factor*
Factor → *Power* `^` *Factor* | *Power*
Power → `<number>` | `<identifier>` | `(` *Expression* `)`

- + means at least one
- * means zero or more
- `<number>` is any nonnegative integer literal.
- `<identifier>` is any string not a reserved word.
  Identifiers must start with a letter, followed by zero or more alphanumeric characters.

The following strings are reserved words and cannot be used as identifiers: `antibody, down, downleft, downright, else, if, left, move, nearby, right, shoot, then, up, upleft, upright, virus, while`

Genetic code evaluation

Evaluation order

At each time step, viruses and antibodies (collectively called *hosts*) are evaluated in the order they enter your body.

Variables

Each variable has the initial value of 0. After each time unit, a variable retains its value for the beginning of the next time unit. For example, if a variable *x* has been assigned a value of 47 at time *t*, then, its value will remain at 47 at the beginning of time *t*+1, and could be assigned to another value in the next round of evaluation.

Variables are local, i.e., they are never shared between different hosts. If two hosts have genetic code containing variables of the same name, their values can be different. Assignments to a variable *x* in host A's genetic code do not affect *x*'s value in another host.

Special variables

A special identifier `random` may be used as a variable in genetic code. This variable is read-only: an attempt to assign this variable results in a "no-op": it's like this assignment never exists. Whenever this variable is evaluated, a random value between 0 and 99 (inclusive) should be returned.

Sensor commands

virus and antibody commands

The `virus` command returns the location of the closest virus in one of the eight directions from the host of the genetic code. The following table shows a pattern of the command's results for the locations to be returned. Viruses in blank locations are never reported.

38			31			32
	28		21		22	
		18	11	12		
37	27	17	host	13	23	33
		16	15	14		
	26		25		24	
36			35			34

The value of each location is relative to the host. The unit digit is between 1 and 8, and indicates the direction of the location. The remaining digits indicate the distance from the host to that location.

If there are multiple viruses with the same minimal distance from the host, return the location of the one with the lowest direction number. In other words, the return value is the smallest possible one.

If there are no viruses at all, the `virus` command should return 0.

The `antibody` command works similarly, but returns the location of the closest antibody instead.

nearby command

The `nearby` command looks for the closest organism (virus or antibody) closest to the host of the genetic code in a given direction.

If the closest organism in the given direction is a virus, this command should return 10*x*+1, where *x* is the distance from the host to that organism as shown in the table above.

If the closest organism in the given direction is an antibody, this command should return 10*x*+2, where *x* is the distance from the host to that organism as shown in the table above

If no organisms are present in the given direction, the `nearby` command should return 0.

Action commands

During each time unit, at most one action command can be executed. If a host attempts to perform more than one action command, subsequent commands act like a no-op (as if that command does not exist).

move command

The `move` command relocates the host of the genetic code one unit in the specified direction. If the destination location is occupied, the command acts like a no-op.

shoot command

The `shoot` command attempts to attack an organism located one unit away from the host in the specified direction. If the target location has no organism, the command is a no-op.

If the target location is an organism of the same type as that of the host (e.g., a virus attacking a virus), the command has the same effect as the normal scenario. Be careful who you shoot at!

Boolean evaluation

Conditions in the `if` and `while` statements evaluate to integers. To interpret integer values as booleans, positive values are considered true, and negative values and zero are considered false.

Avoiding infinite loops

It is possible for a `while` loop to execute for too many iterations. To avoid infinite loops during an evaluation of a genetic code, a `while` loop that has run for 1000 iterations is terminated, i.e., its guard automatically becomes false, and the subsequent statement is then evaluated.

In effect, a `while` statement in a genetic code

```
      while (e) s
```

works like the following code in most programming languages:

```
      for (int counter = 0; counter < 1000 && e > 0; counter++) s
```

Sample genetic code (still in progress)

x = y + 5
y = x + 1
if (y) then move upright else shoot down

Display and controls

A human body may have a lot of cells. Sometimes, it might not be possible to show every cell and have you focus on a region of the body. Therefore, you should be able to zoom in and out parts of the body. Even if you are looking at a certain part of the body, that doesn't mean the other parts stop working.

The player should be able to pause the immune system at any time.

When the immune system is running, the player should be able to slow down or speed up the duration of each time unit.

We recommend that you display the immune system on a webpage. As coders, you can use the Spring Framework to link your Java backend to your web frontend. As players, you should be able to send commands from the frontend to the Java backend.

Think about how to display the immune system so that players have enough information to decide what to do.

Configuration file (still in progress)

The configuration file contains many parameters that can be adjusted to balance the immune system.

- The first line contains two positive integers *m* and *n* (the dimension of a human body).
- virus spawn rate
- antibody credits
- antibody placement cost
- antibody move cost
- speed multiplier
- initial virus health
- initial antibody health
- virus attack damage and gain
- antibody attack damage and gain

Project timeline (subject to change)

- Tue 8 Feb: Design overview document due
    - This includes the outline of the display (view and controller) and your class hierarchy (model).
    - Your design will be given feedback, so you can address flaws before you write your code.
- Wed 16 Feb: Evaluator for genetic codes due
    - At this point, your project should be able to parse and execute genetic code.
- Thu 24 Feb: Project due
    - Possible extension: after the final exam period