# COMP122 - Assessment 3

## Information

|  |  |
| --- | --- |
| **Name:** | Oliver Legg |
| **Email:** | [O.Legg@student.liverpool.ac.uk](mailto:O.Legg@student.liverpool.ac.uk) |
| **Student ID:** | 201244658 |

## Requirements

### Part 1

My first requirements are to:

* Create an *abstract* class called *‘Cockroach.java’*
* Create an *abstract* method called takeStep() taking no parameters
* Create an attribute name for the cockroach (String)
* Create an attribute that keeps track of the cockroach’s current location
* Include accessors

Those requirements will help me to set up the cockroach class. There will be different types of cockroaches named Don and Bella. These are the requirements for those types:

* Don and Bella should have its own individual subclass of Cockroach
* Don and Bella should implement takeStep()

Don’s take step is very particular, he should:

* Fall to floor 1 with a 0.01% chance each step
* Move down a floor if he rolls a 1 or a 2
* Move up a floor if he rolls 3, 4 or 5
* Re-roll the dice and move up the number of floors corresponding to the number on the dice if he rolls a 6.

In Bella’s step, she should roll a dice too and must:

* Move down 1 floor if she rolls a 1, 2 or 3
* Move up 2 floors if she rolls a 4
* Move up 3 floors if she rolls a 5
* Do nothing if she rolls a 6
* If Bella tries moving through 86, she should remain there for a turn
* If Bella lands on 86, she should remain there for a turn

Once all of this has been implemented, I should carry out the experiment on a file called RW.java

The first experiment I should carry out is this:

* Have both cockroaches walk for 100 time steps
* Save the maximum floor reached in those 100 steps
* Repeat this process 2000 times and calculate the average

The second experiment should be carried out like so:

* Have both cockroaches walk forever until they reach the top floor
* Save the number of steps it took to reach the top floor
* Repeat this process 2000 times and save all the number of steps for each experiment
* Calculate the average steps it took to get to the top

The third experiment should be like so:

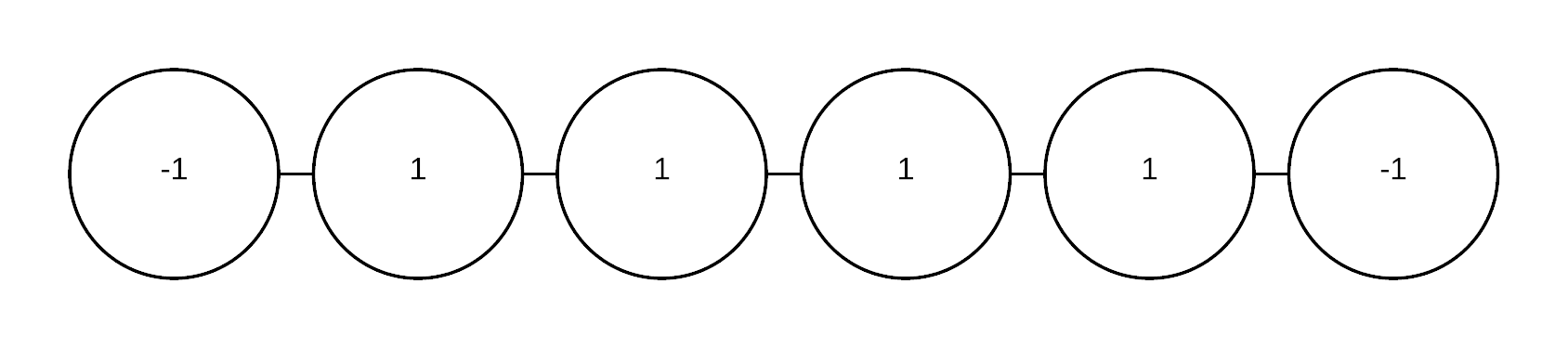
* Simultaneously run Don and Bella’s takeSteps()’s and count the number of times they’re on the same floor
* Their starting position does count for their cooccurrences.
* Step 2000 times for 2000 experiments
* Calculate the average time over the 2000 experiments

### Part 2

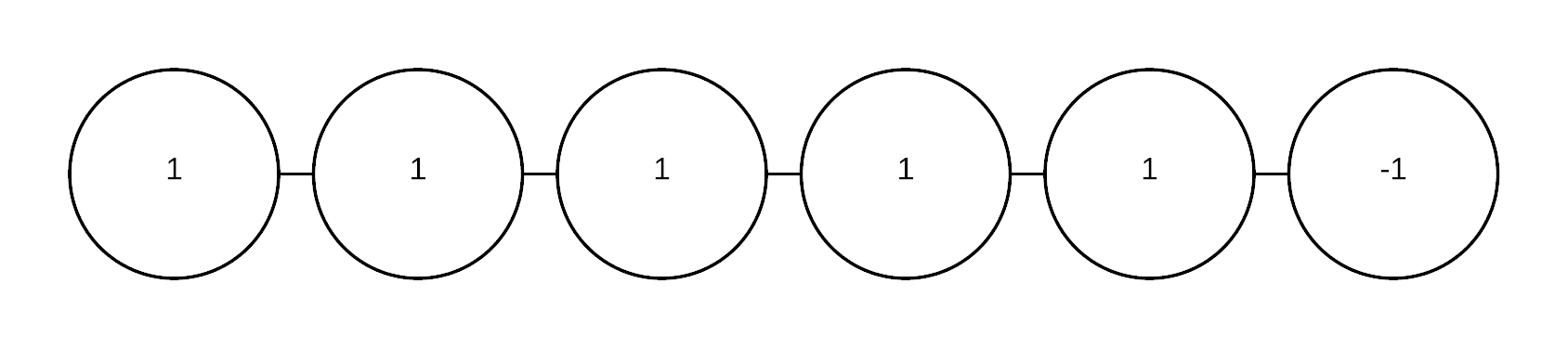
For this next task I will have to write a graph class with two subclasses: cycle and path. On these subclasses I will implement one move of the random process for example, selecting one edge and updating the labels on the two vertices.

I will have to call my application program “CoopGame.java”. This is where I will call my subclasses. In my CoopGame.java, I will run my program from here and will pass parameters as user input. 1 parameter is the number of vertices on the graph and the second parameter is the number of trials. This will be an optional input which upon entering nothing for this, it defaults to 2000. I will also have to handle user input error, for example, if the user enters too many parameters, or too little, the program should produce an error.

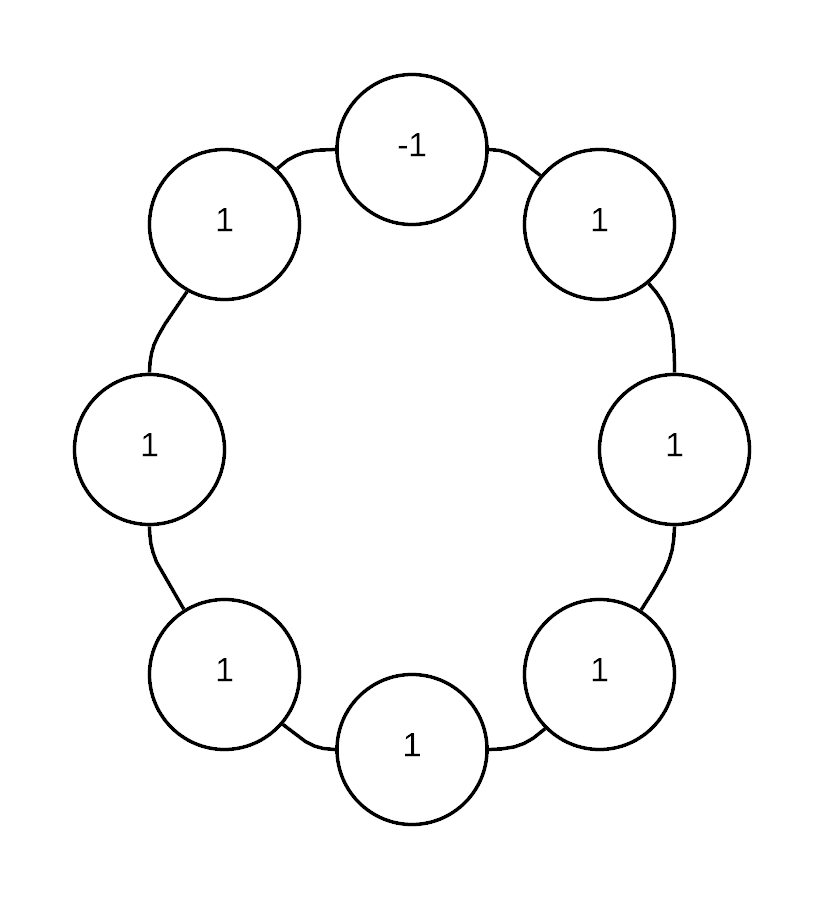
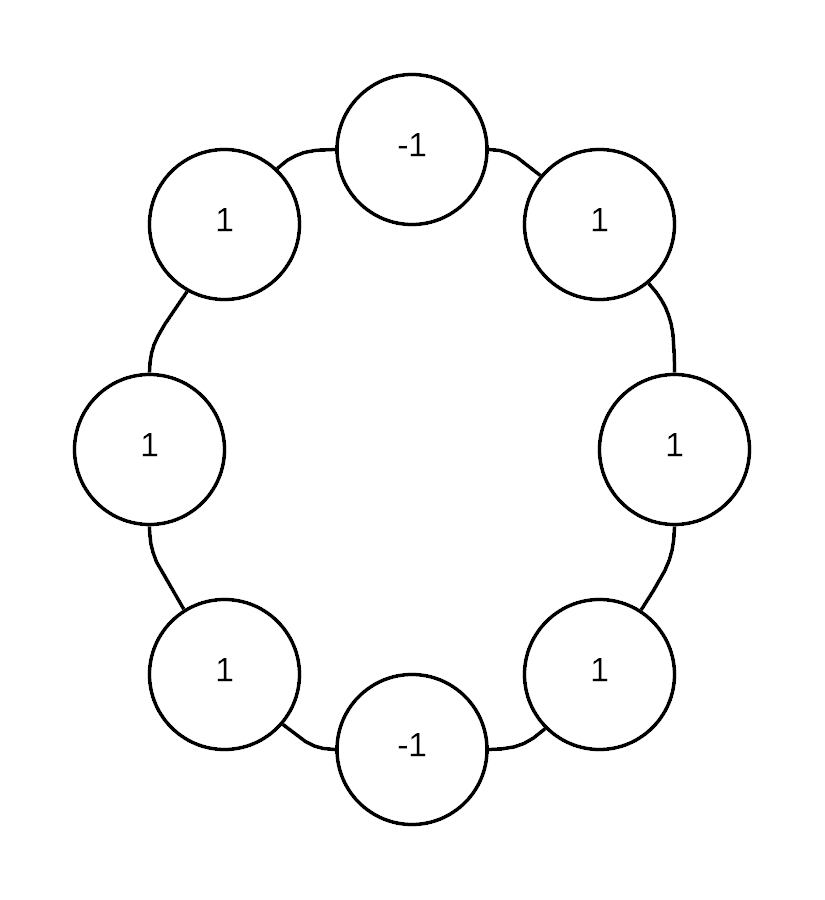
When I create the graph for the path. Each end should have non-cooperatives nodes like so:



The path above should occur if we start with one non-co-operator, the path below should occur if there is are two non-co-operators.



The cycle nodes should look like the diagram on the left below if they have one non-cooperator. It should look like the image on the right below if the cycle has two non-cooperators. If the two non-cooperators scenario is chosen then the nodes picked should be the last node in the cycle and the node in the middle of the cycle. Each node should be able to interact with the node adjacent to each other around the cycle e.g. n1 = v, n2 = v+1.

I will have to design my program so that if for example, the two nodes v1 and v2 are selected:

IF V1 = 1 and V2 = -1

THEN V1 = -1 and V2 = -1

Scenario 2

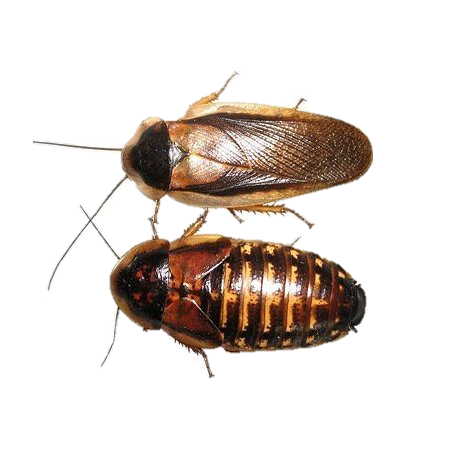
IF V1 = -1 and V2 = -1

THEN V1 = 1 and v2 = 1

After all this is done, I will have to make my program so that it picks 2 random spots on the graph and compares the two like the algorithm above. Once all the nodes display 1, then the program should stop. The 2 nodes selected must be random but still adjacent. The program must save the number of times the algorithm above is applied until they all display 1. I will save these values to a list, so I can get the average.

## Analysis and Design

### Part 1

For this section, my objective is to measure how many time steps it would take for 2 different types of cockroaches to traverse the empire state building. The 2 cockroaches traverse with something known as a “timestep” or simply “step” to refer to one “unit of time”, which is the amount of time necessary for a cockroach to move exactly once. It is possible that during this single move the cockroach could traverse several floors, or possibly remain stationary, but at the beginning of a time step the cockroach is on some floor, and at the end of the timestep it has moved once (or possibly remained stationary) and is again on some floor of the building (i.e. not “in between” floors). These 2 cockroaches are called Don and Bella. We must consider how long it will take a cockroach to reach the top floor? And given a certain time frame, what is the highest floor that has been reached during that period? Other things we will have to consider is how often the two cockroaches will be on the same floor during their random walks. When processing these values, they will obviously be a very broad range of results, thus, it makes sense that we carry out more than just 1 experiment. If we carry out 2000 experiments, we can compute the average. The more experiments, the more accurate our results will be. A cockroach cannot go below floor 1 or above floor 102. Therefore, the values produced will always be in a range between 1-102.

I plan to create my program with a superclass called cockroach where Don and Bella inherit the traits that they have in common. This means that I will have two sub classes called Don and Bella. I plan to create a step function which will work like a dice with a chance that it will fall and a chance that it will climb with other factors. I will run my Experiments from a file called RW, this is where I will call some of step functions in the cockroach subclasses.

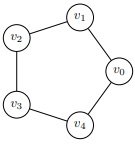
### Part 2

In part 2, we have another element of randomness, however, we are looking at the case of the prisoner’s dilemma. The prisoner’s dilemma is the conundrum of a group of prisoners with no interactions between people. When two people interact, they can “cooperate” for their mutual benefit, one could “cooperate” and the other “defect” (so the “defector” gets more benefit than the “co-operator”), or both can “defect” (meaning they get less than they would if they both cooperate).

I will be creating a model which is conceptually similar. I can model the prisoner’s dilemma using a graph. I’m going to look at two different graphs, namely the Path and the Cycle. A path on n vertices is just a collection of n vertices that form a “line”.

I will create a function that takes a step that picks a vertex at random. This vertex will also pick another vertex which is adjacent to the first vertex. If two -1’s are selected, then the vertices change to 1’s. If one -1 and one 1 is selected, then the vertices become 1’s. If the two selected vertices are 1’s, they remain the same. The vertices can go all the way around in the circular graph, but they don’t in a path.

In the image below, this will display the circular graph:



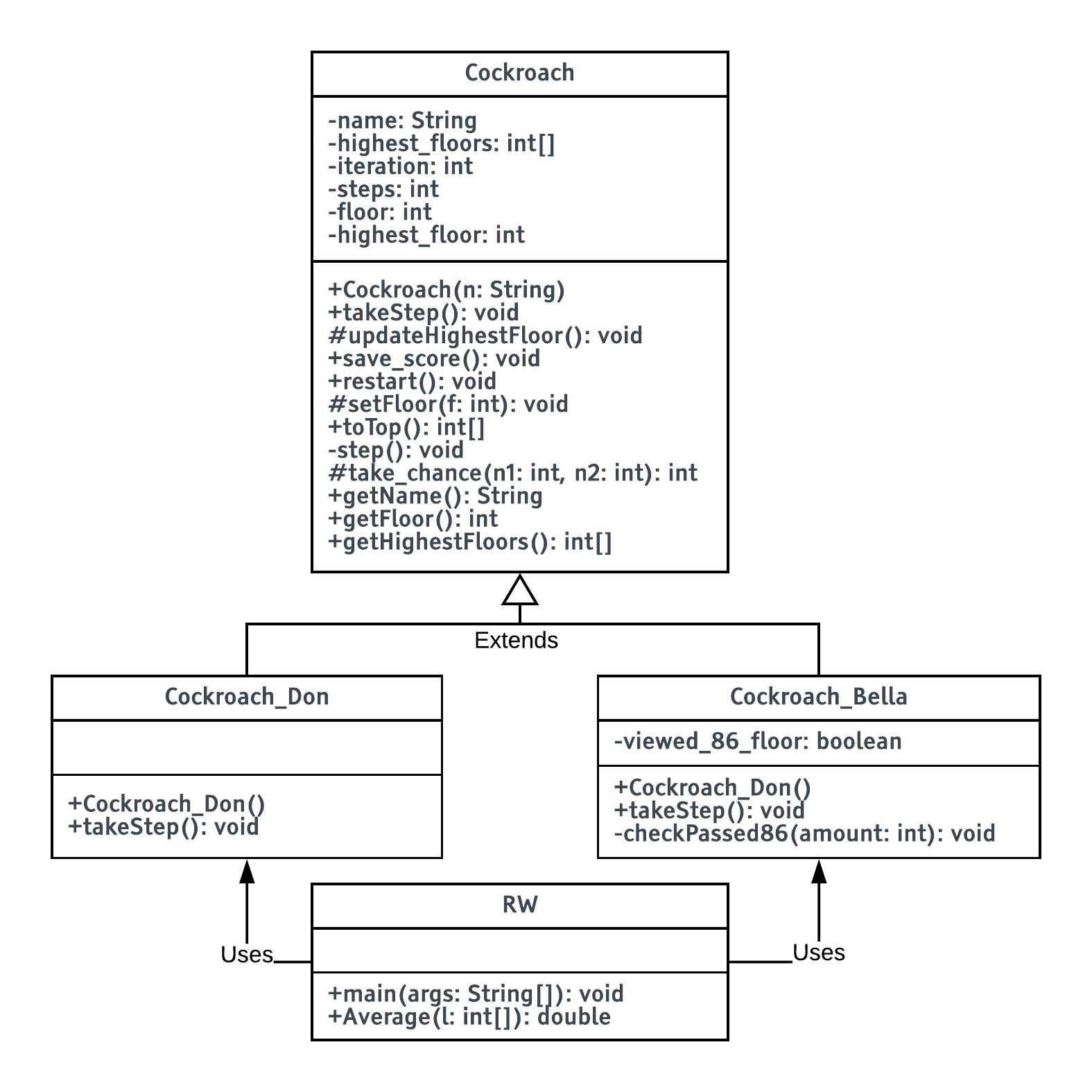
The path graph looks like so:

https://i.gyazo.com/9bfee4a91d624425cd059f73bf7b511a.png

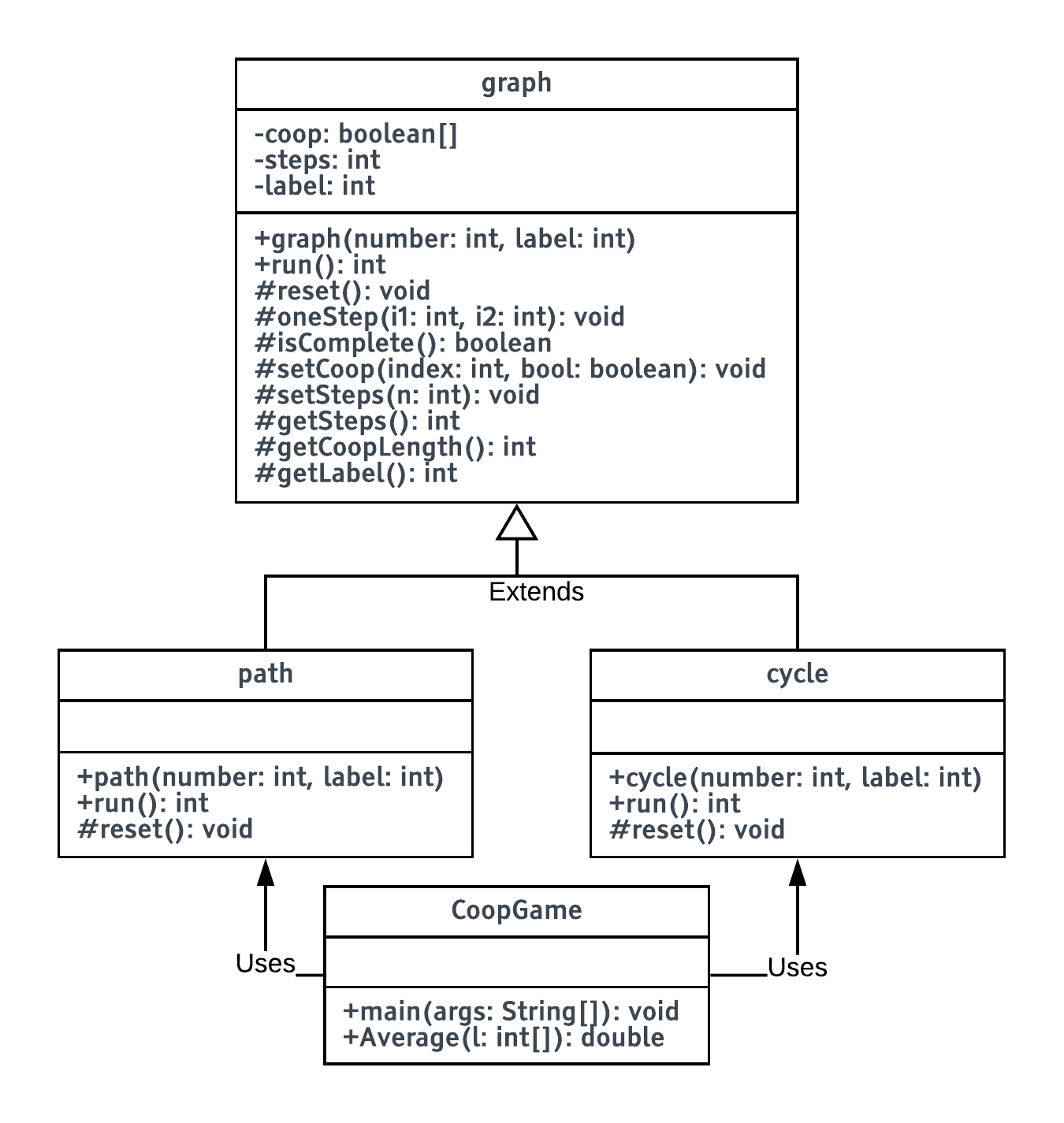
I will run the step function I mentioned before infinitely until all the vertices are 1. I plan to return the number of steps it takes for this conundrum to solve. Like the cockroach task, this will be a hugely varied result, so I will run the test multiple times and get a varied number of results. Afterwards, I will calculate the average of these results. This should get me an accurate result of how many times this takes to get everyone to cooperate.

## Class diagram

### Part 1



### Part 2



## Pseudocode

### Part 1

#### RW

FUNCTION main(args)

Cockroach[] cockroaches <- NEW Cockroach[2]

Cockroach\_Don Don <- NEW Cockroach\_Don()

Cockroach\_Bella Bella <- NEW Cockroach\_Bella()

cockroaches[0] <- Don

cockroaches[1] <- Bella

cooccurancesList <- NEW int[2000]

cooccurances <- 1

*//Cockroach\_Bella Bella <- NEW Cockroach\_Bella()*

FOR i in 0, 1999

FOR v in 0,99:

FOR q in 0, cockroaches.length

cockroaches[q].takeStep()

FOR q in 0, cockroaches.length

cockroaches[q].save\_score()

cockroaches[q].restart()

FOR q in 0, cockroaches.length

OUTPUT("----------")

OUTPUT(cockroaches[q].getName())

OUTPUT("----------")

OUTPUT("2000 experiments, walking 100 steps, the maximum height achieved has average value: "+AVERAGE(cockroaches[q].getHighestFloors()))

OUTPUT("2000 experiments, the steps it took to reach the top has average value: "+AVERAGE(cockroaches[q].toTop()))

OUTPUT()

FOR i in 0, 1999

cockroaches[0].restart()

cockroaches[1].restart()

FOR 0, 1999

cockroaches[0].takeStep()

cockroaches[1].takeStep()

**if** (cockroaches[0].getFloor() == cockroaches[1].getFloor())

cooccurances++

cooccurancesList[i] <- cooccurances

cooccurances <- 1

OUTPUT("----------")

OUTPUT("2000 experiments the number of times Don and Bella share the same floor has average value: "+AVERAGE(cooccurancesList))

OUTPUT("----------")

FUNCTION AVERAGE(l):

avg <- 0

FOR (int i <- 0 i < l.length

avg <- avg + l[i]

RETURN avg / l.length

#### COCKROACH

*//methods*

FUNCTION Cockroach(n)

SELF.name = n

FUNCTION ABSTRACT takeStep()

FUNCTION updateHighestFloor()

IF (SELF.floor > SELF.highest\_floor)

SELF.highest\_floor = SELF.floor

*//saves the score*

FUNCTION save\_score()

SELF.highest\_floors[SELF.iteration++] = SELF.highest\_floor

*//resets the cockroach's attributes for the next experiment*

FUNCTION restart()

SELF.highest\_floor = 1

SELF.floor = 1

SELF.steps = 0

*//sets the floor that they're on*

FUNCTION setFloor(f)

IF (f < 1)

SELF.floor = 1

ELSE IF (f > 102)

SELF.floor = 102

ELSE

SELF.floor = f

*//returns the number of attempts the cockroach takes to get the floor 102*

FUNCTION toTop()

FOR i in 0, 1999

WHILE (SELF.floor < 102)

SELF.takeStep()

SELF.step()

SELF.stepsList[i] = SELF.steps

SELF.restart()

RETURN SELF.stepsList

*//takes a step*

FUNCTION step()

SELF.steps++

*//rolls the dice*

FUNCTION take\_chance(n1, n2)

RETURN n1 + ((n2 \* MATH.random())

*//returns the cockroach's name*

FUNCTION getName()

RETURN SELF.name

*//returns the floor they're on*

FUNCTION getFloor()

RETURN SELF.floor

*//returns the highest floor list*

FUNCTION getHighestFloors()

RETURN SELF.highest\_floors

#### COCKROACH\_DON

FUNCTION Cockroach\_Don()

SUPER("Don")

FUNCTION takeStep()

*//0.1% chance of flying into the centre of the strailwell and falling to gnd floor*

IF (SELF.take\_chance(1,1000) == 1)

SELF.setFloor(1)

ELSE

chance <- SELF.take\_chance(1,6)

IF (chance == 1 || chance == 2)

SELF.setFloor(SELF.getFloor() - 1)

ELSEIF (chance > 2 && chance < 6)

SELF.setFloor(SELF.getFloor() + 1)

ELSE

SELF.setFloor(SELF.getFloor() + SELF.take\_chance(1,6))

SELF.updateHighestFloor()

#### COCKROACH\_BELLA

FUNCTION Cockroach\_Bella()

SUPER("Bella")

FUNCTION takeStep()

IF (!SELF.viewed\_86\_floor && SELF.getFloor() == 86)

SELF.viewed\_86\_floor <- **true**

ELSE

chance = SELF.take\_chance(1,6)

IF (chance > 0 && chance < 4)

SELF.viewed\_86\_floor <- **false**

SELF.checkPassed86(-1)

ELSE IF (chance == 4)

SELF.viewed\_86\_floor <- **false**

SELF.checkPassed86(2)

ELSE IF (chance == 5)

SELF.viewed\_86\_floor <- **false**

SELF.checkPassed86(3)

SELF.updateHighestFloor()

*//this method checks if bella passes floor 86*

FUNCTION checkPassed86(a)

*//f = floor*

*//a = floor amount to change*

f <- SELF.getFloor()

IF ((f < 86 && f+a > 86)||(f > 86 && f+a < 86))

SELF.setFloor(86)

ELSE

SELF.setFloor(f+a)

### Part 2

#### GRAPH

FUNCTION graph(number, label)

SELF.steps <- 0

SELF.label <- label

SELF.coop <- NEW [number]

FUNCTION ABSTRACT run()

FUNCTION ABSTRACT reset()

FUNCTION oneStep(i1, i2)

IF (NOT SELF.coop[i1] && NOT SELF.coop[i2])

SELF.coop[i1] <- TRUE

SELF.coop[i2] <- TRUE

ELSE IF ((NOT SELF.coop[i1] && SELF.coop[i2]) || (NOT SELF.coop[i2] && SELF.coop[i1]))

SELF.coop[i1] <- FALSE

SELF.coop[i2] <- FALSE

SELF.setSteps(SELF.getSteps()+1)

FUNCTION isComplete()

FOR (i in 0, SELF.coop.length)

IF (NOT SELF.coop[i])

RETURN FALSE

RETURN TRUE

FUNCTION setCoop(index, bool)

SELF.coop[index] <- bool

FUNCTION setSteps(n)

SELF.steps <- n

FUNCTION getSteps()

RETURN SELF.steps

FUNCTION getCoopLength()

RETURN SELF.coop.length

FUNCTION getLabel()

RETURN SELF.label

#### PATH

FUNCTION path(number, label)

**SUPER** (number, label)

FUNCTION run()

SELF.reset()

WHILE (TRUE)

IF (SELF.isComplete())

RETURN SELF.getSteps()

randomNum = RANDOM(0, getCoopLength()-1)

SELF.oneStep(randomNum, randomNum + 1 )

FUNCTION reset()

FOR i in 1, SELF.getCoopLength

SELF.setCoop(i, TRUE)

SELF.setCoop(0, FALSE)

IF (SELF.getLabel() == 2)

SELF.setCoop(SELF.getCoopLength()-1, FALSE)

SELF.setSteps(0)

#### CYCLE

FUNCTION cycle(number, label)

**SUPER**(number, label)

FUNCTION run()

SELF.reset()

WHILE (TRUE)

IF (SELF.isComplete())

RETURN SELF.getSteps()

randomNum <- RANDOM (0, SELF.getCoopLength()-1)

SELF.oneStep(randomNum, (randomNum+1 MODULUS SELF.getCoopLength()))

FUNCTION reset()

FOR i in 0, SELF.getCoopLength()

SELF.setCoop(i,TRUE)

IF (SELF.getLabel() <- 2)

SELF.setCoop(Math.floor(SELF.getCoopLength()/2)-1, FALSE)

SELF.setCoop(SELF.getCoopLength()-1, FALSE)

SELF.setSteps(0)

#### CoopGame

FUNCTION main(args)

n <- 100

trials <- 2000

IF (NOT args.isAllInteger)

OUTPUT("\tn <- (int) number of vertices")

OUTPUT("\n\tUsage: java CoopGame n [trials]\n")

OUTPUT("\ttrials <- (int) number of trials (optional, defaults to 2000)")

IF (args.length < 1)

OUTPUT("\n\tUsage: java CoopGame n [trials]\n")

OUTPUT("\tn <- (int) number of vertices")

OUTPUT("\ttrials <- (int) number of trials (optional, defaults to 2000)")

ELSE IF (args.length == 1)

n <- toInt(args[0])

ELSE IF (args.length == 2)

n <- toInt(args[0])

trials <- toInt(args[1])

IF (n < 1 || n > 5000 || args.length > 2)

OUTPUT("\n\tOops, check your parameter(s). The first parameter specIFies n,")

OUTPUT("\tthe number of vertices (at least 3). ")

OUTPUT("\tAn optional second parameter specIFies the number of trials to")

OUTPUT("\tperFORm (between 1 and 5000).")

cycle c1 <- NEW cycle(n, 1)

cycle c2 <- NEW cycle(n, 2)

path p1 <- NEW path(n, 1)

path p2 <- NEW path(n, 2)

cycle\_results1 <- NEW [trials]

cycle\_results2 <- NEW [trials]

path\_results1 <- NEW [trials]

path\_results2 <- NEW [trials]

FOR i in 0, trials

cycle\_results1[i] <- c1.run()

cycle\_results2[i] <- c2.run()

path\_results1[i] <- p1.run()

path\_results2[i] <- p2.run()

OUTPUT("Cycle of size "+n+" ("+trials+" trials) with one starting -1 value: "+Average(cycle\_results1))

OUTPUT("Cycle of size "+n+" ("+trials+" trials) with two starting -1 value: "+Average(cycle\_results2))

OUTPUT()

OUTPUT("Path of size "+n+" ("+trials+" trials) with one starting -1 value: "+Average(path\_results1))

OUTPUT("Path of size "+n+" ("+trials+" trials) with two starting -1 value: "+Average(path\_results2))

FUNCTION Average(list)

sum <- 0

FOR (i to list)

sum <- sum + i

RETURN sum / list.length

## Testing

### Part 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Description | Expected Results | Actual results | Remedial Action |
| 1 | 2000 experiments, walking 100 steps, the maximum height achieved has average value | Around 75 | 75.1925 | - |
| 2 | 2000 experiments, the steps it took to reach the top has average value | Around 145 | 145.9915 | - |
| 3 | 2000 experiments, walking 100 steps, the maximum height achieved has average value | Around 40 | 39.649 | - |
| 4 | 2000 experiments, the steps it took to reach the top has average value | Around 310 | 312.315 | - |
| 5 | 2000 experiments the number of times Don and Bella share the same floor has average value | Around 320 | 322.4525 | - |

### Part 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Description | Expected Results | Actual results | Remedial Action |
| 1 | Entering a non int input | Controlled error and input tip | Yes | - |
| 2 | Entering an input out of range | Controlled error and input tip | Yes | - |
| 3 | Entering nothing as a parameter for the program | Controlled error and input tip | Yes | - |
| 4 | Cycle of size 100 (2000 trials) with one starting -1 value | 381.377 | 368.967 | - |
| 5 | Cycle of size 100 (2000 trials) with two starting -1 value | 590.746 | 599.8953 | - |
| 6 | Path of size 100 (2000 trials) with one starting -1 value | 352.518 | 347.9675 | - |
| 7 | Path of size 100 (2000 trials) with two starting -1 value | 510.855 | 508.2215 | - |

## Evidence

(Test number correspond with tests on previous section)

### Part 1

|  |  |
| --- | --- |
| Test number | Screenshot Evidence |
| 1 | https://i.gyazo.com/eaf8731b500080d753e230333cbf1bc0.png |
| 2 | https://i.gyazo.com/eaf8731b500080d753e230333cbf1bc0.png |
| 3 | https://i.gyazo.com/eaf8731b500080d753e230333cbf1bc0.png |
| 4 | https://i.gyazo.com/eaf8731b500080d753e230333cbf1bc0.png |
| 5 | https://i.gyazo.com/eaf8731b500080d753e230333cbf1bc0.png |

### Part 2

|  |  |
| --- | --- |
| Test no | Screenshot Evidence |
| 1 | https://i.gyazo.com/59c590a938780e35ed40aa92196ca82a.png |
| 2 | https://i.gyazo.com/cda4d4449cbc1625ca645c1e253d529f.png |
| 3 | https://i.gyazo.com/bc11203e33de8a5c2f7538573b18ad80.png |
| 4 | https://i.gyazo.com/fd4392911ba025cb23c914ea4b5b4710.png |
| 5 | https://i.gyazo.com/fd4392911ba025cb23c914ea4b5b4710.png |
| 6 | https://i.gyazo.com/ce5832c56437fe8e23b927187b61a80f.png |
| 7 | https://i.gyazo.com/ce5832c56437fe8e23b927187b61a80f.png |

## Extra questions

### Part 1

**Suppose that on a very windy day, there is a strong cross breeze on floor 95 due to some open windows. For each cockroach, again starting from the ground floor, they begin their random walk. If they land exactly on floor 95, they are blown out of the Empire State Building. If they make it to the top floor, they stop. What are the chances that they will make it to the top floor? (This probability will depend upon the particular cockroach in question. . . ) In your report, describe what kind of experiments you would run to determine this value, as well as giving the probability of reaching the top floor. Include your Java code in your submission. What other kinds of interesting things could you report on?**

Answer:

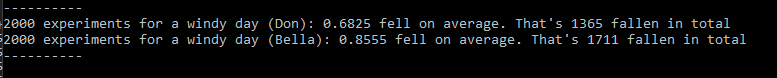
There is a lot to consider for this feature to added. The first 88 floors for Don and Bella wouldn’t be relevant as they cannot just from floor, let’s say, 50 to 95 immediately, therefore, but they can do so from floor 89. If Don rolls a 6 and a 6 again, he would get knocked back to the beginning. While this is a 1 in 12 chance there, the chances would become more likely as they get closer. The chances will be different for both cockroaches. For example, Bella can only move to the 95th floor if she is stood on the 93rd and 92nd and 96th floor as she can move up 2 and 3 floors. If Bella lands on floor [92, 93], there will be a 1 in 6 chance that she will land on it. If she is on floor 96, there is a 1 in 2 chance that she lands on floor 95.

For Don, there is a 1 in 2 chance for him to land on floor 95 if he is on the 94th floor. If he is on floor 96, there is a 1 in 3 chance that he will fall to 95.

If Don is on floor one of the following floors: [89,90,91,92,93,94], there will be a 1 in 12 chance that he will land on the 95th floor.

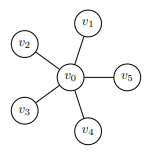
For both cockroaches, there is always that chance that they will fall from a floor above 95 back to 95 again.

This is the experiment I ran in java.



Therefore, for don, there is a 1365 in 2000 that Don will fall and a 1611 in 2000 chance that Bella will fall

### Part 2

**How about if the graph is a star? The start, Sn, consists of a single “middle” vertex connected to n other vertices, so that it looks like a star. So the star Sn has n + 1 vertices, one at the center, and n vertices on the “spokes”. You can also represent a star for this random process just by using a Java array of the appropriate length. You can represent the interactions by choosing one of the outer vertices at random (choose a random integer between 1 and n inclusive), and having it interact with the center vertex v0 to update their labels (as these are the only possible interactions). How long does it take this process to converge? Note that if you go beyond about 20 or 25 vertices, your process will take a very long time to converge! It’s very, very slow to finish on the star? Can you give any idea why it’s so slow to converge on the star? Does it matter if, with one starting −1 if it starts on the center vertex or one of the others? If you want, you can include calculations for the star in your main program you wrote for the path and the cycle. As I said, I would recommend not running the process on a star for much more than 20 to 25 vertices as it’s very slow to converge!!**

Answer:

The 2 vertices selected will always be v0 and vn. This is because all v’s are connected to v0 but vn is not necessarily connected to vn. To converge, this will take the length of the number of vertices \* 2 to converge.