Programming 2 Project

The Battle of the Sexes

Gravili Niccolò, Pasqualoni Saverio, Rastogi Archit,
Spadoni Grabriele, Zaneti Sato Giovanna

July 2022

1 Project Goal

We aimed to develop a simple Simulator in Java in order to study a basic evolutionary model comparing two different traits, using FPCS playoff tables and analyze differences between our results to Dawkins' case study.

The core of the idea behind it is to give the user enough parameters and tools to play with the simulation and see how the results will deviate from the expected ones.

1.1 Basic features

Without going into the specifics *(that will be elaborated on in the next pages)*, the main problems and definitions we had to look over were:

Division of people We needed to define what a person is in our model, its traits and characteristics, which of those characteristics are hereditary and how they work.

Mating of People How two people meet and reproduce and for how long they will keep on mating.

Offspring Generation The creation of a new person when two people mate and the definition of its traits.

Evolutionary Success Punishing and rewarding behaviours with respect to FPCS table by Dawkins.

Equilibrium of the Simulation Lastly we had to implement a method to stop the simulation once a certain type of equilibrium is reached. We needed to understand what this equilibrium could be and how to check it.

1.2 Inputs

The possible inputs are:

1. **a**,**b** and **c** values with respect to the FPCS table;
2. initial population's **starting percentages** of Dawkins' traits;

3. a **life expectancy** value used to change the lifespan of a single person *(default=10)*;

4. a δ (**delta**) value used to check the precision of the equilibrium *(default=0.3)*;

5. a **sleep** value used to check the frequency of the equilibrium control process *(default=4)*;

Only inputs 1 and 2 are available by the end user.

Be aware that inputs 3, 4 and 5 are still editable in the code but we choose to not give the user the possibility to not cause malfunctions.

2 Traits Organization

The focus of the simulation is to understand which reproductive approach works better from an evolutionary prospective.

We can call the chosen traits **libertine** and **puritan**.

Every person in this simulation has a **gender** and one of the previous traits, in this way each individual is divided into 4 groups as in *table 1*.

Gender	Libertine	Puritan
Male	Philanderer	Faithful
Female	Fast	Coy

Table 1: division of the individuals

The two genders are defined as **classes** named **Men** and **Woman**, both extending a **super-class** named **Human** that extends **Thread** (more on that later).

Since we have only two possible traits we decided that the best way to describe it is by using a simple **boolean**.

The presence (or absence) of that boolean is enough to describe all the possible cases.

For reasons that will be clarified in a further paragraph we have also added two other **volatile boolean** in each individual.

Those two boolean, called **father** and **mother** are used when determining the trait of a newborn.

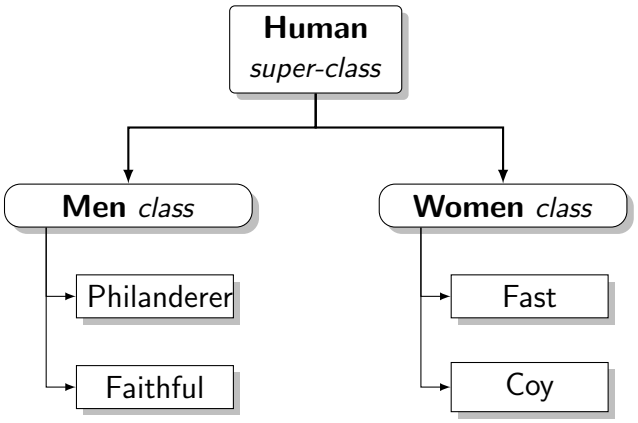

Figure 1: basic definitions and division of a person

3 Mating Process

3.1 Threading

One of the biggest problems we had to overcome when coding the simulation was:

how do we randomize the mating process?

Our solution is heavily dependent on the thread theory in java.

We, indeed, found out that the nature of threads was a strong enough argument to have a randomization of the process.

As said before, in our implementation both men and women run as separate **threads**.

Inside the **men** and **women** classes there's a **method** called **run**, which describes the mating process.

Each individual execute its run method in a loop ruled by a **static boolean** value called **running**, which is turned **True** at the start of the simulation and **False** at the end.

3.2 Woman Stack

The next logic step would be to create a sort of *"queue object"*, but this object is not symmetric and works diffrently for men and women.

Women, as soon as generated, are **pushed** in a **synchronized data structure** called **Women_stack** (this way only one thread at a time can gain access to the structure).

In the meantime all men threads try to **pop** a woman thread from the stack and start the mating process.

Once the mating is ended and the offspring generated (more on that later) the woman thread is **pushed** in the stack again.

While the man thread tries again to **pop** a new woman from the stack.

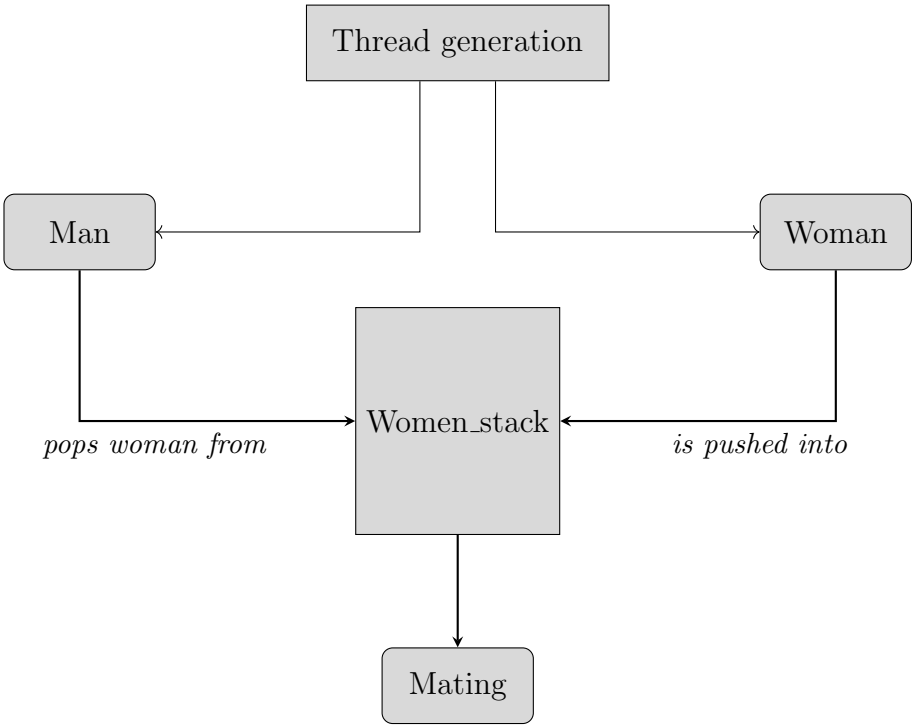

Figure 2: Overview of the program workflow

3.3 Offspring generation

Once the mating process begins a new baby is born.

The gender of the offspring is randomized, but its trait is not.

To generate the behavioural boolean we choose to look after the traits of its grandparents.

Those traits are stored in two boolean for each parent thread.

A **random integer** $R \in \{0, ..., 3\}$ is generated.

For each **libertine** (*boolean=true*) grandparent a **counter** C is raised ($C \in \{0, ..., 4\}$).

If $R < C$ the baby is **libertine** otherwise is **puritan**.

We have created a new thread that goes into the cycle described above.

4 Evolutionary Success

The declared goal of the project is to find the composition of a final stable population.

It directly translates into tracking the mating success of a behaviour and thus *punishing and rewarding* behaviours to give and take advantages.

4.1 Life Expectancy Value

Inside **Human** class there's an integer variable L_e **life expectancy** (initialized for each individual at *default=10*).

This value gets updated each time a person mate:
a method called **give_take_points** is used after the mating process.

It adds (or subtracts) L_e points according to the *FPCS table* (table 2).

	F	P
C	$(a - \frac{b}{2} - c,\ a - \frac{b}{2} - c)$	(0,0)
S	$(a - \frac{b}{2},\ a - \frac{b}{2})$	$(a - b,\ a)$

Table 2: basic FPCS table

With default $a = 15$, $b = 20$ and $c = 3$ according to **Dawkins**' case study (table 3).

	F	P
C	(2,2)	(0,0)
S	(5,5)	(-5,15)

Table 3: Dawkins' FPCS table

4.2 Death

The first way in which we punish individuals based on their behavior is **death**.

The **give_take_points** method after updating the L_e of both individuals checks for a death condition.

Any individual will be killed if $L_e < 0$.

This translates to stopping the thread through the use of the **interrupt** method.

The individual, upon his death, will update one of the **controller** method's variables.

The way in which this mechanism influences the simulation is obvious:

- by killing a unsuccessful individual we prevent the spreading of his behavior
- a lesser thread competing to synchronize on the woman stack is an advantage for all other individuals

4.3 Sleep

Every time a thread is about to access the woman stack it will go through a **sleep statement**.

It will sleep for a time

$$t = \frac{20}{L_e}$$

So, a higher life expectancy, will result in a lower sleep time, hence an advantage in the competition for the access to the woman stack.

This mechanism makes it so that successful individuals mate more often, spreading their behavior more effectively across the simulation.

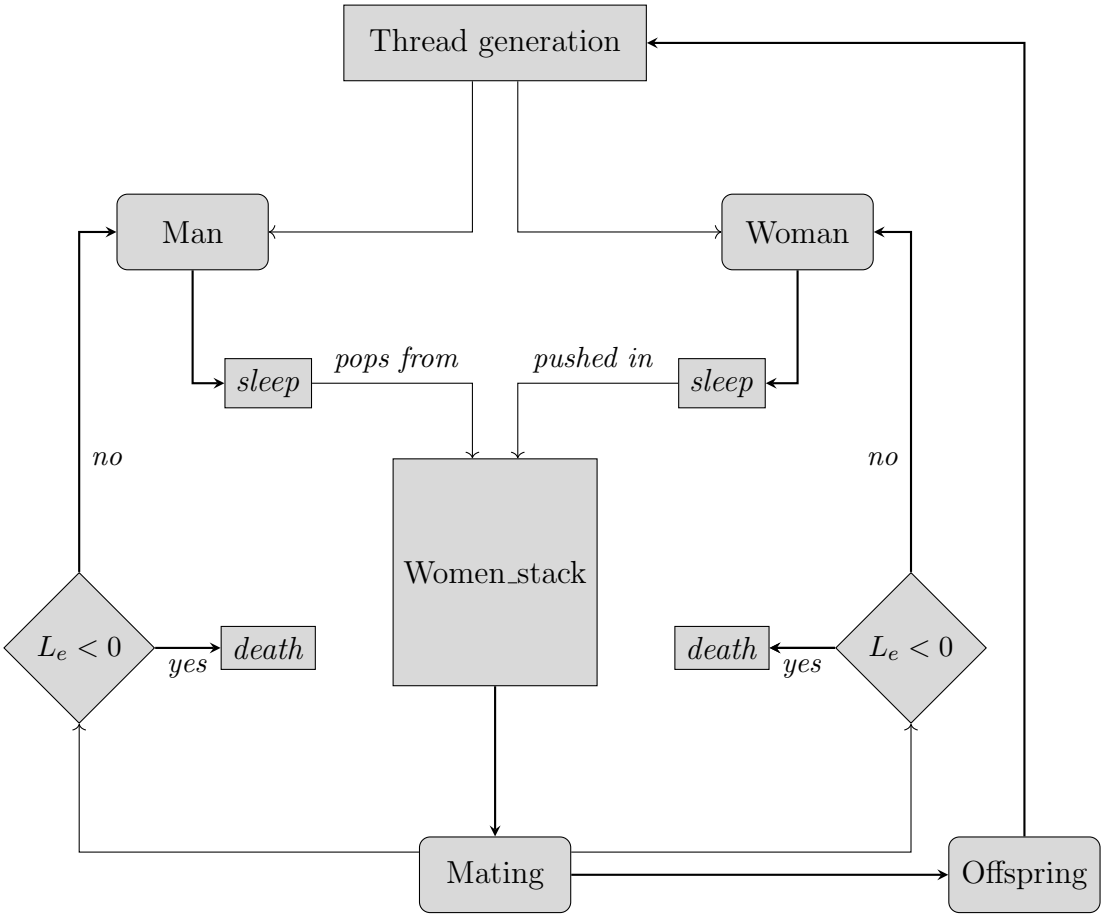

Figure 3: Complete program workflow

.

5 Equilibrium

5.1 Theory

Given the nature of the simulation, one of the most important step towards its success is to understand which results are good and when to stop the program from running indefinitely.

In order to do that we have to define two distinct ratios:

- a **male ratio**
$$Mr = \frac{faithful}{philanderer}$$
- a **female ratio**
$$Fr = \frac{coy}{fast}$$

The basic idea is that, when those two numbers reach a certain type of stability we should kill all the process and print out the results.

But this raises few questions:

1. Do they reach stability?
2. If so, do they reach it together?

A simple answer to 1. is:

under Dawkins' terms and with a large and various enough starting sample size, **experimentally** yes. *(more on that later)*

The question 2. is a little bit trickier, but the good news is that we can ignore it altogether by defining:

- a **total ratio**
$$Tr = Mr + Fr$$

It's easy to understand that the convergence of Tr is a good enough condition to the stability of the simulation (it is *extremely* implausible to have a stable Tr and unstable Mr and Fr).

5.2 Implementation

We have created a **Controller** class which tracks the number of people in each of the 4 groups.

This class gets updated each time a new thread is created or stopped (i.e. each time a person is born or dies).

But the heavy-lifting is done by another class.
The class **Equilibrium** is an **extension** of the basic Java class **Thread**.

Inside this class there are two main values.

- a δ (**delta**) value *(default=0.3)* used as threshold value
- a **goal** value for a counter *(default=20)* that, if reached, stops the simulation

If the difference between two consecutive measurements of Tr is smaller than a control value (i.e. $\delta \geq |Tr_1 - Tr_2|$), a new control value is set to $\delta' = |Tr_1 - Tr_2|$ and a **counter** is raised by 1.

The new control value is used in the same way in a successive iteration. If we still have $\delta' \geq |Tr_3 - Tr_4|$, then $\delta'' = |Tr_3 - Tr_4|$ and the counter gets raised again.

Otherwise if we encounter a situation in which $\delta^n < |Tr_m - Tr_{m+1}|$ the counter returns to 0 and the δ value is reset to its default value.

Once *counter* == *goal* the simulation is stopped, all the running threads killed and the percentages of the population printed.

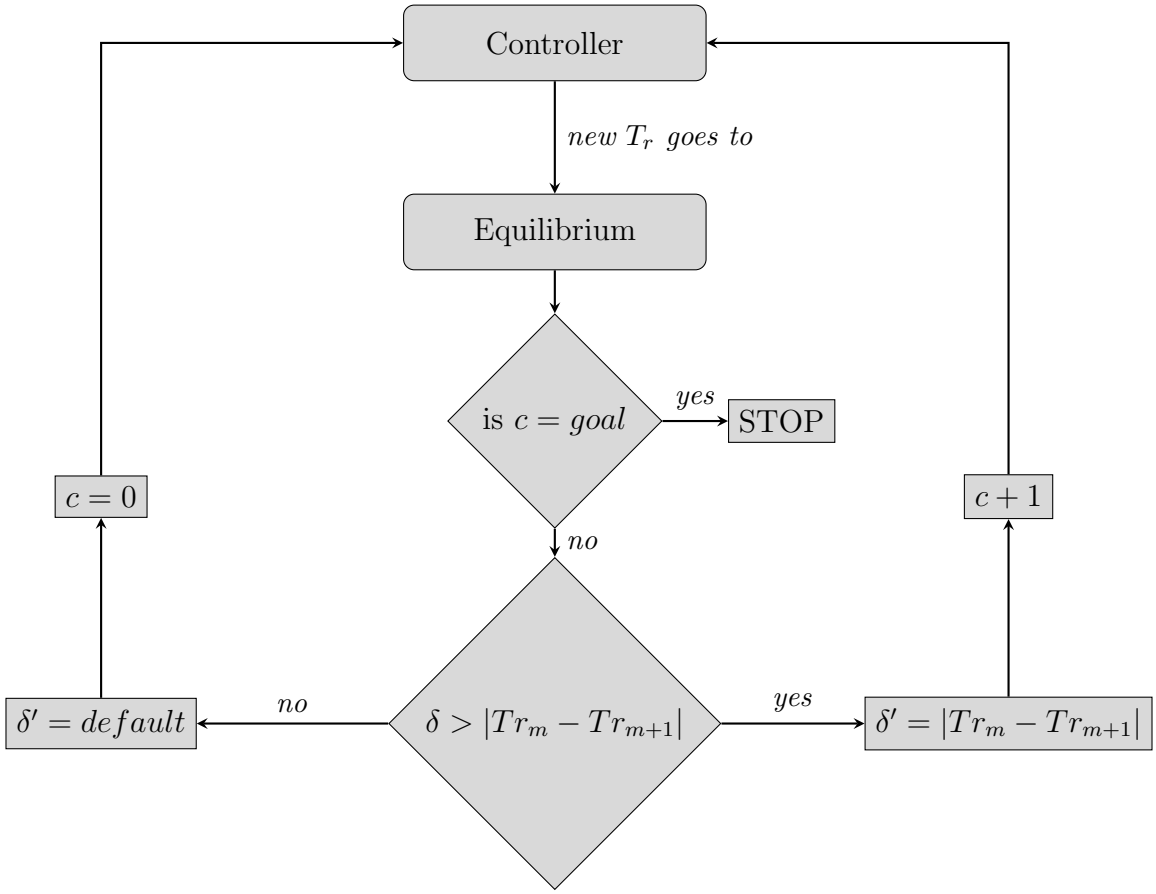

Figure 4: Equilibrium chart

.

6 Results and Interesting Cases

We executed the project changing different parameters one at a time, documenting and analyzing the results.

Every result is the **average** of a *group of runs* with the same parameters.

The percentages of the output are calculated in this way *(let's take coy women as an example)*:

$$Coy\% = \frac{\#Coy}{\#Women} \cdot 100$$

This is done to compare the output numbers directly to the theoretical results given in the project specification.

Co	coy women
Co_r	coy women result
Fs	fast women
Fs_r	fast women result
Fth	faithful men
Fth_r	faithful men result
Ph	philanderer men
Ph_r	philanderer men result
t	time (s)

Table 4: Legend

6.1 Dawkins' Case

The first group of test was done using Dawkins **a**, **b** and **c** parameters and changing the initial population.

Constant parameters:

- $a = 15$; $b = 20$; $c = 3$
- $\delta = 0.30$
- $L_e = 10$

Initial population	Co	Fs	Fth	Ph	t
$Co: 25\%$ $Fs: 25\%$ $Fth: 25\%$ $Ph:: 25\%$	78,8	20,9	66,3	33,6	7,6
$Co: 15\%$ $Fs: 30\%$ $Fth: 25\%$ $Ph: 30\%$	73,2	26,8	59,2	40,4	6,8
$Co: 30\%$ $Fs: 15\%$ $Fth: 30\%$ $Ph: 25\%$	82,9	16,9	73,5	26,3	6,4

Table 5: Dawkins' results with reasonable populations

We can use the first test as a benchmark for future ones because it gives no advantages to any behaviour.

In fact we see that in the first case the numbers yielded by the program are **pretty close** to those foreseen by Dawkins *(only being off by* $4 \sim 5\%$*)*.

In the next two cases we've only made small changes from the *"benchmark case"*.

We notice the **general tendency** is still achieved, but the observed number diverges (coy women and faithful men make up for a grater part of their respective gender's population with coy women being the most prominent group).

Now we propose a table with more **extreme changes** to the initial population.

Initial population	Co	Fs	Fth	Ph	t
$Co: 10\%$ $Fs: 70\%$ $Fth: 10\%$ $Ph: 10\%$	70,5	29,5	53,2	46,8	8,9
$Co: 10\%$ $Fs: 10\%$ $Fth: 70\%$ $Ph: 10\%$	80,4	19,4	72,2	27,6	10,9
$Co: 2\%$ $Fs: 41\%$ $Fth: 2\%$ $Ph: 55\%$	64,8	36,2	47,5	51,5	13,1

Table 6: Dawkins' results with extreme case populations

In the **first case** we strongly increase the number of **fast** women.

As we can see this causes an increase of the fast women in the final result but also an increase of similar magnitude in philanderer males.

This result can be interpreted as a consequence of our definition of the fast woman as a combination of a behavior, libertine, and a gender, female:

a rise in fast women translates to a overall greater presence of libertine behavior in the simulation.

Despite this the **general tendency** we found in previous tests is also clearly identifiable here.

In the **second case** we strongly increase the number of **faithful** men.

This leads to a **slight increase** in the coy population, and a **strong increase** in the faithful population.

This result seems to mirror the last one but in this case we notice a difference between the increase of the faithful and coy populations.

In the **third case** we chose as input a very **extreme** initial population, with lots of fast and philanders and few coys and faithful.

The results **diverges** a lot from Dawkins theoretical results.
a correct tendency is still kept for the females but it is lost on the males.

6.2 Changing a, b and c

This time we played around with **a**, **b** and **c** values.

Constant parameters:

- initial population $Co : 25\%$ $Fs : 25\%$ $Fth : 25\%$ $Ph :: 25\%$
- $\delta = 0.30$
- $L_e = 10$

a, b and c values	Co	Fs	Fth	Ph	t
$a = 20; b = 10; c = 3$	24,1	75,5	24,5	75,5	4,6
$a = 10; b = 5; c = 20$	2,1	97,1	5	94,5	9,1
$a = 10; b = 25; c = 5$	1,1	98,9	3.6	95,5	1,07

Table 7: Results with a, b and c changed

In the **first case** we raise the **a** value and lower the **b** one.

This change is a a clear incentive towards **philanderer-fast** couples as we can see in the equations in Table 3.
However it is not a complete win for the libertine behavior, as a higher a value and lower b value lead to a better payoff for every kind of mating.

In the **second test** we lower the **a** and **b** values with $a > b$ and raise the **c** value.

In this test a and b are lower but since $a > b$ every couple combination, especially **philanderer-fast**, yield a a better payoff as it was in the previous case.

The difference in results is caused by the high **c** value that strongly discourages **coy-faithful** couples.

In the **third case** we raise slightly the **b** and **c** values while lowering the **a** one.

Fast-philanderer couples are discouraged and **faithful-coy**/**faithful-fast** encouraged.

This resembles the Dawkins' tendencies but resulting in a more **extreme** end population.

6.3 Changing Life Expectancy

In this section we analyze the changes in the results with Dawkins' **a**, **b** and **c** values and with a change in **life expectancy** L_e.

Constant parameters:

- $a = 15$; $b = 20$; $c = 3$
- initial population $Co : 25\%$ $Fs : 25\%$ $Fth : 25\%$ $Ph :: 25\%$
- $\delta = 0.30$

life expetancy L_e	Co	Fs	Fth	Ph	t
5	94,5	5,5	80,1	19,4	8,7
20	52,3	46,6	50,1	49	1,4

Table 8: Results with L_e changed

In the **first case** we raise the L_e.

Here we notice that a lower **life expectancy** L_e leads to more **extreme** results.

This can be interpreted as a consequence of the fact that every **punishment** given by the *"point allocation/deallocation system"* (i.e. **sleep** and **death**) is enhanced by the lower L_e.

In the **second case** we lower the L_e.

As expected, for the same reasons of the previous case, the results appear **flattened out**.

Thank you for reading.

This project was made by:

Gravili Niccolò
gravili.2025398@studenti.uniroma1.it

Pasqualoni Saverio
pasqualoni.1845572@studenti.uniroma1.it

Rastogi Archit
rastogi.1982785@studenti.uniroma1.it

Spadoni Grabriele
spadoni.1995808@studenti.uniroma1.it

Zaneti Sato Giovanna
zanetisato.1983203@studenti.uniroma1.it