Ereditarietà delle mutazioni

Simulazione dell'ereditarietà mendeliana delle mutazioni in una popolazione

Bononi Andrea, Casadei Mattia, Checchi Daniele
Classe 4°P Liceo A.Roiti ▸ A.S. 2019-2020

Ereditarietà delle mutazioni

NOTE TEORICHE

PROGETTAZIONE DEL PROGRAMMA

NOTE TECNICHE DEL PROGRAMMA

NOTE TEORICHE

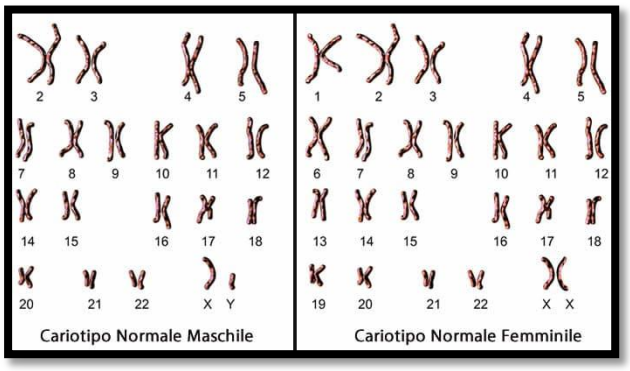
L'ereditarietà di tipo mendeliano nelle mutazioni genetiche

Si definisce mutazione una **modificazione delle funzioni metaboliche dell'organismo** conseguente ad una modifica del genoma dell'individuo. Le mutazioni possono essere:

- **MUTAZIONI SOMATICHE** (e quindi non ereditarie, poiché si manifestano solo nell'individuo malato).
- **MUTAZIONI DELLA LINEA GERMINALE** (e quindi ereditarie se il gamete mutato è coinvolto nella fecondazione).

Le mutazioni della linea germinale interessano quindi i **cromosomi** e possono a loro volta essere di due tipologie:

- **MUTAZIONI AUTOSOMICHE**: la mutazione avviene sui cromosomi non sessuali, quindi non è legata al sesso del nascituro al momento della fecondazione.
- **MUTAZIONI NON AUTOSOMICHE**: la mutazione avviene sui cromosomi sessuali, quindi dipende dal sesso del nascituro.



Le mutazioni trattate nel progetto sono di tipo **NON AUTOSOMICO,** pertanto sono legate al sesso: nel nostro caso si tratta di mutazioni legate al cromosoma **X**, sia come allele **DOMINANTE,** sia come allele **RECESSIVO**; la trasmissione della mutazione segue le leggi della genetica mendeliana ***(vedi approfondimento a lato).***

UN PO' DI STORIA...

Gregor Mendel *fu un monaco agostiniano vissuto a Brno tra il 1822 e il 1884 ed è considerato il* ***padre della genetica****, la branca della biologia che studia* ***l'ereditarietà dei caratteri*** *all'interno di più generazioni successive.*

Definiamo ***"mendeliani"*** *i caratteri che si trasmettono secondo le leggi elaborate dallo stesso Mendel.*

Mendel non conosceva i cromosomi e non poteva di certo sapere in cosa consistesse propriamente una ***"mutazione genetica",*** *ma con ottima approssimazione molte delle malattie genetiche sono considerate* ***"caratteri mendeliani".***

PROGETTAZIONE DEL PROGRAMMA

L'idea alla base della simulazione

L'idea alla base della simulazione è quella di rappresentare una generica **popolazione di individui** attraverso delle palline, ciascuna delle quali dotata di *"vita"* e *"movimento"* propri.

Ogni pallina *"nasce"* con una propria velocità, che le permette di muoversi nello schermo e che decresce con il passare del tempo fino ad azzerarsi. Solo a questo punto la pallina avrà completato il proprio ciclo vitale e verrà eliminata dallo schermo.

Il dilemma: Come differenziare le palline?

Ciascuna di esse viene generata con il **colore** corrispondente (a seconda del sesso) e con un **contorno** che ci permette di differenziare gli individui malati da quelli sani. Tutto questo è stato possibile grazie ad una **corrispondenza biunivoca** fra le palline contenute all'interno di un ***ArrayList*** e una stringa rappresentante il genoma, contenuta all'interno di uno ***StringList.***

La **regolazione della trasmissione genetica** segue le probabilità mendeliane e avviene tramite istruzioni che modificano il genoma dell'individuo. Per esempio, quando l'individuo sano si ammala, è il suo genoma a cambiare.

La lettura delle stringhe contenute all'interno dello ***StringList*** ci permette di regolare il ***display*** dell'individuo, proprio come se il **genotipo** si manifestasse **nell'espressione fenotipica.**

La riproduzione può avvenire solamente tra un individuo di sesso maschile e uno di sesso femminile. Quando due palline generano dei figli (il cui numero, variabile, è 1 oppure 2) assumono lo stato di ***"genitori",*** quindi non potranno riprodursi nuovamente.

L'obiettivo è stato quello di rappresentare una simulazione quanto più fedele possibile alla realtà:

- Il sesso e il genoma del nascituro vengono decisi casualmente (rispettando le **probabilità della genetica mendeliana)**
- Il numero dei figli generati viene stabilito casualmente (1o 2 figli a coppia) per rispettare l'attuale **tasso di fertilità in Italia,** che si aggira attorno al valore **1.3** per donna secondo le ultime stime condotte dall'**OMS** sulla popolazione italiana nel 2019.
- Sebbene il numero iniziale di individui dei due sessi venga deciso casualmente, questo è accettabile rispetto al valore del **50%** dato dalla concezione probabilistica.

PROGETTAZIONE DEL PROGRAMMA

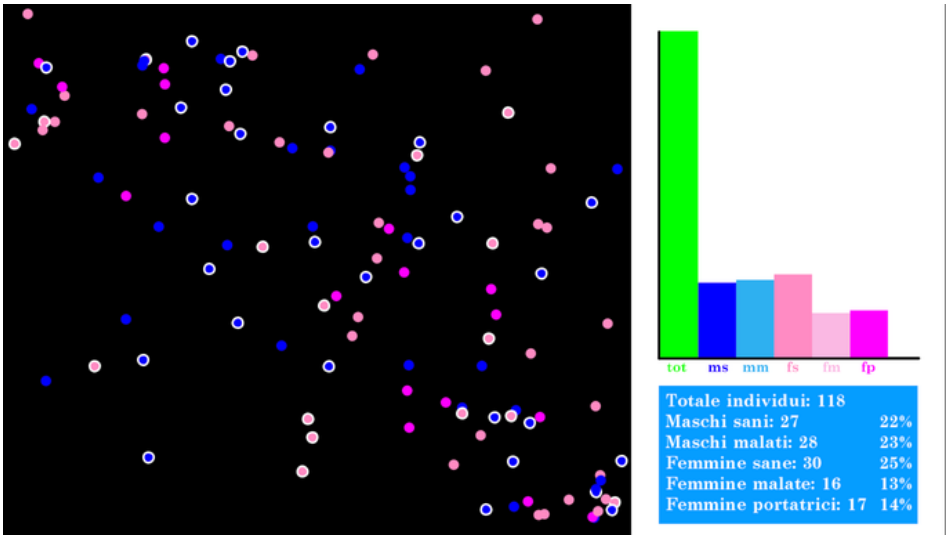
La simulazione su Processing

La simulazione su ***Processing*** è composta da tre schermate:

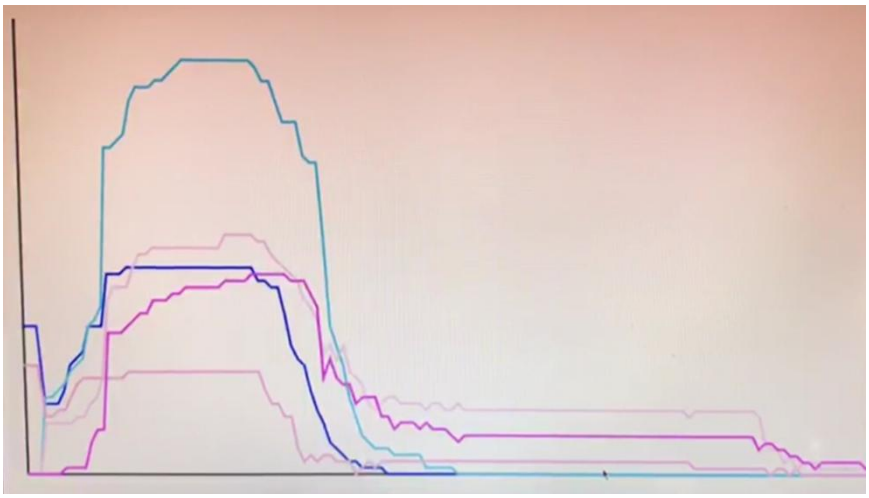
- **Una schermata iniziale *(figura 1),*** composta da titolo e comandi principali.
  A sinistra è presente una legenda delle palline che verranno rappresentate nella simulazione, mentre **a destra** della schermata un rettangolo grigio contiene i tre comandi principali:
  - Il tasto ***'TAB'*** permette di accedere alla schermata principale;
  - Il tasto ***'INVIO'*** permette di far apparire la mutazione sugli individui;
  - Il tasto ***'g'*** avvia la riproduzione degli individui quando questi si scontrano;

  In basso figura la parte interattiva del programma, dove l'utente può scegliere:
  - Il numero di individui che verranno mutati, rispettivamente **10, 20** o **30** su un totale fisso di 40 individui generati automaticamente all'inizio della schermata principale;
  - Il tipo di mutazione, scegliendo tra ***"DOMINANTE"*** e ***"RECESSIVA".*** Trattandosi di malattie che colpiscono il cromosoma X *(vedi pag. 3)* l'unica differenza consiste nella presenza di **individui femmine portatrici se viene scelta la mutazione recessiva,** in quanto le femmine eterozigote ***Xx*** non manifestano la malattia ma possono trasmetterla.

- **Una schermata principale *(figura 2),*** composta essenzialmente da due parti:
  - Una parte di larghezza (2*width)/3 con sfondo nero rappresenta lo spazio di movimento delle palline. Esse vengono generate in posizioni ***random*** all'interno della schermata e si muovono rimbalzando sui bordi.
  - Una parte di larghezza width/3 con sfondo bianco visualizza l'andamento della mutazione all'interno della popolazione, visualizzando un **grafico a colonne** in cui l'altezza della colonna è proporzionale al numero di individui presenti.
    Una tabella sottostante visualizza il numero di individui per ogni categoria e fornisce la **percentuale** rispetto al numero totale di individui.



- **Una schermata finale *(figura 3),*** nella quale viene visualizzato un **grafico riassuntivo**: l'asse ***x*** rappresenta il tempo (conteggiato in ***FrameCount***), mentre l'asse **y** rappresenta il numero degli individui. Il grafico permette quindi di visualizzare, per ogni categoria di individui, il loro numero con il passare del tempo.



NOTE TECNICHE DEL PROGRAMMA

La struttura del programma

Il programma si compone di due parti: lo sketch principale contiene la dichiarazione delle variabili, il ***setup()*** e il ***draw(),*** che permettono la visualizzazione della simulazione, mentre la **classe *Ball*** crea l'oggetto ***'pallina'*** associando ad essa attribuiti e metodi che ne permettono le funzioni principali.

Dichiarazione e inizializzazione delle variabili

VARIABILI INTERE:

t: assume un valore random, 0 o 1 → determinazione casuale del sesso degli individui
e, f : assumono valori random interi → determinazione casuale del genoma degli individui
i, j, k: contatori → regolano lo scorrimento di cicli for
figli: assume valori random interi (1 o 2) → determinazione casuale del numero di figli
cont, num_mutati: contatori → controllano il numero di individui con specifiche caratteristiche
mod=0: assume valori 0, 1, 2 → permette di entrare nella schermata corrispondente
var, xgrafico → indicazioni grafiche per il layout dello schermo
c1, c2, c3, c4, c5, c: contatori → ognuno di essi conta gli individui di una specifica categoria (es **c1** maschi sani, **c2** maschi malati … **c** indica invece gli individui totali)

STRUTTURE LIST

ArrayList <Ball> balls = new ArrayList<Ball> () → contiene gli oggetti della classe ***Ball***
StringList gen → contiene i genomi (si tratta di stringhe, es 'XY' o 'XX') degli individui
IntList data1, data2, data 3, data 4, data5 → Liste di interi che contengono rispettivamente i valori di c1, c2, c3, c4, c5 al termine del conteggio degli individui di ciascuna categoria

VARIABILI BOOLEANE:

a = true → quando a = false diviene accessibile il ciclo che permette la riproduzione degli individui
n1, n2, n → a seconda dello stato informano se una pallina ha assunto o meno lo stato di genitore
death → a seconda dello stato comunica se la pallina è viva, ossia se ha un'accelerazione !=0
gene → a seconda del valore, deciso dall'utente nella schermata iniziale, la malattia è dominante o recessiva. Questo booleano controlla più condizioni dell'intero programma.
regolacolore1, … 5 → controllano l'apparizione dei rettangoli nella schermata iniziale

VARIABILI FLOAT:

x, y, x1, x2, y1, y2 → rappresentano le coordinate del centro della pallina
d → rappresenta la distanza tra due palline (esse si toccano se d<2*raggio, cioè se d<15)

VARIABILE PFONT:

font1 → font utilizzato nel layout delle varie schermate

Il setup()

Nella funzione **setup()** creiamo la finestra con sfondo nero, stabiliamo il ***frameRate*** e il tipo di ***font*** utilizzato per il layout delle schermate.

Creiamo lo ***StringList*** **'gen'** che conterrà le stringhe corrispondenti ai genomi di ciascun inviduo.

Creiamo gli ***IntList*** '**data1', 'data2', 'data3', 'data4', 'data5',** che conterranno i valori rispettivamente dei contatori **c1,c2,c3,c4,c5** per la generazione del grafico della schermata conclusiva.

Riempiamo l'***ArrayList*** **'balls'**, contenente i primi 40 individui , il cui sesso viene deciso casualmente. Contemporaneamente riempiamo lo ***StringList*** **'gen'** con i genomi corrispondenti.

LA SCELTA STRATEGICA: GLI ARRAY LIST

Utilizzare un ***ArrayList*** *ci ha permesso di* ***adattare la dimensione*** *dell'Array al numero effettivo di palline, grazie all'istruzione*

balls.add (new Ball (x,y,col));

L'istruzione gen.set (i, "XX") ci permette di aggiungere il genoma corrispondente, creando la già citata ***corrispondenza biunivoca.***

Il draw()

All'interno del **draw()** possiamo individuare diversi **nuclei di istruzioni:**

1° NUCLEO: Realizzazione della **schermata iniziale** (accessibile quando la variabile ***mod==0***):
Si tratta di istruzioni grafiche per la visualizzazione di titolo, legenda delle palline e rettangolo con i comandi del programma. La visualizzazione della parte interattiva –in cui l'utente sceglie le opzioni della simulazione- ci permette di far comparire un rettangolo solo sull'opzione selezionata dall'utente;
Premendo il tasto ***'TAB'***, ***mod=1***: dalla schermata iniziale si passa alla schermata principale.

2° NUCLEO: Realizzazione dal layout della **schermata principale** (accessibile quando ***mod==1***)
Le istruzioni grafiche permettono la comparsa del grafico e della tabella sottostante, all'interno della quale vengono stampati in ordine (da sx a dx della tabella):

- **Categoria** dell'individuo
- Valore del **contatore** (corrispondente alla categoria)
- **Percentuale** di individui (della stessa categoria) rispetto al totale

3° NUCLEO: I contatori vengono azzerati (così che il conteggio di individui venga effettuato correttamente ad ogni draw). Un ciclo ***for*** scorre l'***ArrayList*** **'balls'** e prende ogni pallina grazie all'istruzione **Ball b= balls.get(j)** dove **j** è il contatore del ciclo.
La lettura del genoma corrispondente (all'interno dello ***StringList*** **'gen'**, nella posizione di indice **j**) permette il display della pallina e l'aggiornamento del contatore corrispondente.
I contatori ora aggiornati permettono la generazione delle **colonne del grafico**, di altezza **4*cx**, dove **cx** indica il contatore di ogni categoria di individui.

4° NUCLEO: Quando viene premuto ***INVIO*** (key== ENTER) un ciclo ***for*** scorre i genomi dei primi **"cont"** individui, modificandoli e permettendo, in tal modo, la comparsa della **mutazione.**

5° NUCLEO: Si tratta del nucleo più complesso; alla pressione del tasto **'g'** viene introdotta la possibilità di **accoppiamento per individui di sesso diverso.** Quando un individuo diventa genitore non potrà più fare figli per il resto della sua vita nello schermo. Molteplici condizioni ci permettono di leggere i genomi degli individui genitori e di generare i genomi dei figli, regolando di conseguenza la loro generazione nella schermata.

6° NUCLEO: Realizzazione del layout della **schermata finale** (accessibile quando ***mod*==2**). Le funzioni ***beginShape(), vertex() ed endShape()*** permettono la realizzazione di **grafici a linee**, uno per ogni categoria di individuo. Le coordinate dei vertici si riferiscono ai valori degli ***IntList 'data'***.

La classe Ball

La classe Ball contiene gli oggetti (le palline) che vengono inseriti nell'**ArrayList *'balls'*.**

Si compone di un costruttore che richiede in input le coordinate del centro e il colore della pallina, e di una serie di **attributi** che ciascuna pallina deve avere:

- **'xpos'** e **'ypos'** rappresentano le coordinate del centro;
- **'col'** è la variabile che indica il colore della pallina;
- **'speedX'** e **'speedY'** contengono le componenti x e y della velocità: la generazione randomica ci consente che le palline possiedano moti diversi nello schermo;
- **'acc'** contiene l'accelerazione della pallina (inizialmente 1);
- **'n'** è un booleano che indica se la pallina ha assunto o meno lo stato di *"genitore";*
- **'morte'** è un booleano che indica se la pallina è ancora in vita;

La classe pallina possiede anche **metodi** che regolano il funzionamento degli oggetti:

- **void update()** regola il moto delle palline (moto uniformemente decelerato) e il rimbalzo;
- **void sano(),void malato(),void portatrice()** controllano il display dei diversi individui;
- **float getx(), float gety()** restituiscono le coordinate del centro della pallina *(xpos e ypos);*
- **boolean nascita()** attribuisce alla pallina lo stato di *"genitore" (n cambia stato);*
- **boolean getbirth(), boolean getdeath()** restituiscono i valori dei booleani *n* e *morte;*

Le funzioni principali: comparsa della mutazione e riproduzione

```
if (keyPressed) {
  if (key == ENTER) {
    for (j=0; j < num_mutati; j++) {
      if ( gen.get(j) == "XX") {
        gen.set(j, "xx");
      }
      if ( gen.get(j) == "XY") {
        gen.set(j, "xY");
      }
    }
  }
}
```

La pressione del tasto 'INVIO' introduce la comparsa della mutazione

num_mutati assume un valore a seguito della scelta iniziale dell'utente

VARIAZIONE DEL GENOMA e COMPARSA DELLA MUTAZIONE

```
if (!a) {
  for (j=0; j < balls.size(); j++) {
    Ball b1 = balls.get(j);
    x1 = b1.getx(x);
    y1 = b1.gety(y);
    n1 = b1.getbirth(n);
    for (i=0; i<balls.size(); i++) {
      Ball b2 = balls.get(i);
      x2 = b2.getx(x);
      y2 = b2.gety(y);
      n2 = b2.getbirth(n);
      d = dist(x1, y1, x2, y2);
      if (d <= 15 && n1 && n2) {
        figli = (int)random(1, 3);
        var = 0;
```

I METODI getx(x) e gety(y) leggono le coordinate del centro (x,y) di ciascuna pallina, attribuendole a x1,y1,x2,y2.

Il metodo getbirth(n) attribuisce a n1 e n2 il valore di n, attributo di ogni pallina. Indica se è attivo lo stato di genitore.

Se la distanza d è minore della somma dei raggi e n1==true (cioè b1 non è genitore) e n2==true (cioè b2 non è genitore), genero un numero di figli…

Progetto a cura di:
Bononi Andrea,
Casadei Mattia,
Checchi Daniele.