SIR Simulation

Lorenzo Manini Nicolò Montalti

A.A. 2019/2020

1 Descrizione del progetto

Il progetto consiste in una libreria dedicata allo sviluppo di simulazioni basate sul modello SIR.

Organizzazione del codice

Il codice è stato sviluppato collaborando su GitHub. La repository è stata resa pubblica ed è liberamente consultabile al link in bibliografia [1].

Il sorgente è suddiviso in tre cartelle: include, src e tests. Le dichiarazioni delle classi sono contenute negli header all'interno di include/, mentre la loro implementazione è nei cpp contenuti in src/. Tutti i file utili al testing sono invece raggruppati in tests/. Infine, una copia di questa relazione è contenuta nella cartella tex/.

Struttura del programma

La popolazione

La simulazione ha come "attori" istanze di *Person*. Una *Person* contiene una posizione, una velocità ed alcune informazioni relative allo stato di infezione. In particolare, la variabile membro *Sub Status* è un enumeratore che definisce una sotto categoria di S, I o R. Ad esempio, gli infetti sono divisi in *incubation*, per le quali il virus è nella fase di incubazione, *infective*, cioè in grado di infettare altre persone, e *quarantined*, cioè in quarantena. Ogni *Person* è contenuta in un std::vector<Person>, che ha come alias *People*. Infine, lo struct *Population*, che rappresenta l'intera popolazione, contiene tre oggetti *People* denominati rispettivamente S, I e R. I *Person* contenuti in S sono considerati sani, quelli contenuti in I infetti e quelli in R recuperati.

La simulazione

La classe che permette all'utente di creare e far avanzare la simulazione è *Simulation*. La classe ha tre variabili membro private, che vengono inizializzate dal costruttore:

- *Simulation_ State* state: contiene tutte le informazioni riguardanti lo stato della simulazione
- *G_ Motion*& motion: referenza al modulo che implementa il movimento della popolazione
- *G_ Infection*& infection: referenza al modulo che implementa l'evoluzione dell'infezione

Simulation State è uno struct che contiene un'istanza di *Population*, la dimensione dello spazio della simulazione e il tempo a cui fa riferimento quello stato. Lo spazio in cui avviene la simulazione è stato modellizzato come un piano cartesiano di estremi 0 e *size* su entrambi gli assi; una persona in un *Simulation State* deve sempre trovarsi all'interno di questo spazio. Il tempo è invece stato discretizzato in ticks interi; i ticks di un *Simulation State* non possono diminuire.

G Motion e *G Infection* sono classi astratte nelle quali è dichiarata una funzione membro pubblica virtuale *update*. Una implementazione di *G Motion*, quando viene chiamato *update(Population& population, int size)*, deve aggiornare la posizione dei *Person* contenuti nella referenza a *Population*. La nostra libreria fornisce *Random Motion*. Una implementazione di *G Infection*, quando viene chiamato *update(Population& population, int ticks)*, deve far progredire l'infezione. Questo avviene spostando i *Person* tra i vari *People* (S, I, R) e aggiornando le variabili membro dedicate dei *Person*. La libreria fornisce *Simple Infection* e *Incubation Infection*.

Prima di inizializzare un'istanza di *Simulation*, l'utente deve creare due istanze di classi che implementano rispettivamente *G Motion* e *G Infection* e un *Simulation State* iniziale. A questo punto è possibile creare un'istanza di *Simulation* passando i tre oggetti come parametri al costruttore.

Simulation ha a sua volta una funzione membro pubblica *update()*, che chiama gli *update()* delle istanze di *G Motion* e *G Infection* e incrementa i ticks. In questo modo l'utente, dopo aver creato l'istanza di *Simulation*, deve solo chiamare *Simulation::update()* per far progredire la simulazione di un tick alla volta. *Simulation* ha inoltre una funzione membro pubblica *is_ over()* che ritorna true se non ci sono più infetti nella popolazione.

Output grafico

La libreria mette a disposizione due classi per visualizzare l'andamento della simulazione. La spiegazione dettagliata degli output, con alcuni esempi, è fornita nella sezione Risultati.

Le due classi che gestiscono la grafica sono *Display* e *Plot*. *Display*, basata sulla libreria grafica SFML, permette di visualizzare in tempo reale i *Person* come cerchi colorati che si muovono in un piano. *Plot*, basata sul framework di ROOT, permette invece di visualizzare in tempo reale un grafico con l'andamento del numero di sani, infetti e recuperati. Tramite la funzione membro *Plot::save()* è inoltre possibile salvare il grafico come RootFile e i dati dell'andamento come csv.

Entrambe le classi funzionano in modo analogo. È necessario passare al costruttore una referenza costante allo stato della simulazione che si vuole visualizzare. Questa è ottenibile tramite la funzione membro *Simulation::get_state()*. Dopodiché, per aggiornare ciò che viene mostrato, è sufficiente chiamare rispettivamente *Display::update()* e *Plot::update()*.

2 Compilazione ed esecuzione

La compilazione è gestita tramite CMake. Per compilare il codice è sufficiente creare una directory di build, generare il Makefile tramite CMake e compilare con make.

```
mkdir build
cd build
cmake ..
make
```

Affinché la compilazione abbia successo, è necessario avere installato sulla propria macchina CMake 3.16 o superiore [2], SFML 2.5 o superiore [3] e ROOT [4]. Il programma è stato testato su Ubuntu 18.04 LTS e Ubuntu 20.04 LTS. Si suggerisce di utilizzare quest'ultima versione, se possibile, per via della maggior compatibilità con SFML 2.5.

L'eseguibile del main viene generato nella cartella build/, mentre i test in build/tests. Per eseguire il programma è quindi sufficiente spostarsi nella directory di build e digitare

```
./SIR_simulation
```

Si apriranno due finestre, una con la schematizzazione delle persone e una con i grafici. Chiudendo la finestre dei grafici si termina l'intera applicazione, mentre chiudendo quella con la popolazione si interrompe la sola simulazione, lasciando la possibilità di interagire con il grafico per eseguire un fit o per salvare i dati.

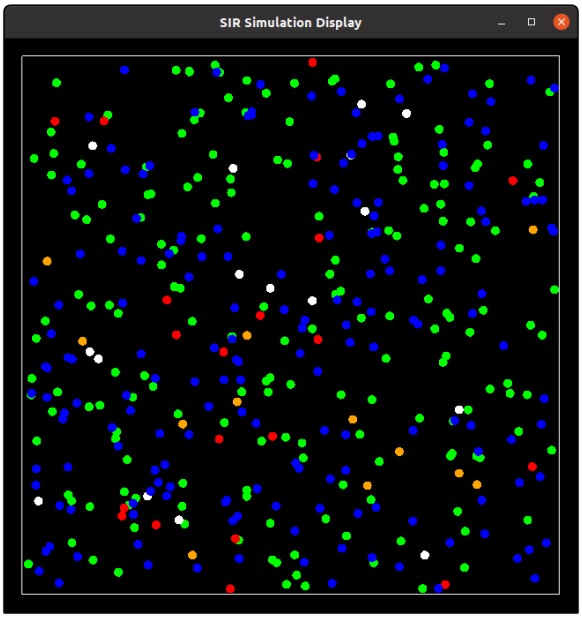


Figura 1: Finestra grafica ottenuta dalla classe Display. Il verde corrisponde ai sani, il rosso agli infettivi, l'arancione al virus in incubazione, il bianco alla quarantena e il blu ai guariti

I test possono essere eseguiti singolarmente, a partire dagli eseguibili generati in build/tests/, o collettivamente con

```
make test
```

Eseguendo il codice con l'address sanitizer abilitato, si è notato che la terminazione della applicazione da parte di ROOT comporta dei memory leak. Si è concluso che essi siano dovuti a librerie esterne, in particolare a libfontconfig. Il problema sembra circoscritto alla classe *Plot*.

3 Risultati

L'applicazione restituisce due tipi di output grafici. Nella prima finestra vengono mostrate le persone, schematizzate come cerchi colorati, che si muovono in uno spazio quadrato. Lo stato delle persone è indicato con colori diversi: verde per i sani, rosso per gli infettivi, arancione se il virus è in incubazione, bianco per la quarantena e blu per i recuperati. Un'immagine d'esempio è mostrata in fig. 1. Contemporaneamente, in una seconda finestra, viene mostrato un grafico aggiornato in tempo reale in cui viene plottato il numero di sani, infetti e recuperati.

Di seguito vengono riportati gli esiti di alcune simulazioni effettuate variando i parametri delle classi *Infection* e *Motion*. Se non diversamente indicato, i parametri dello stato iniziale sono size = 600,

S = 400, I = 10 e R = 0. Si è scelto di iniziare la simulazione con 10 individui infettivi per velocizzarne l'esecuzione. Inoltre si è notato che con un solo infetto può capitare che questo guarisca prima di infettare altre persone. Quest'ultimo caso, sebbene possibile, è stato ritenuto di scarso interesse.

La classe *Random Motion* è stata inizializzata con una deviazione standard pari a 0.2. Si è notato che variare questo parametro influisce poco sulla simulazione. L'unica differenza apprezzabile è nella durata dell'epidemia, che diminuisce all'aumentare della varianza.

La classe *Simple Infection* è stata inizializzata con una distanza critica di 10, una probabilità di infezione di 0.05 e un tempo di recupero di media 200 e deviazione standard 50. Inoltre, alla variante *Incubation Infection*, si sono assegnati un tempo di incubazione di 50 e una probabilità di essere costretti alla quarantena di 0.005.

L'esito di una simulazione con i parametri sopra descritti e la classe *Simple Infection* a gestire l'infezione è riportato in fig. 2a. In fig. 2b si può vedere come aumentando la probabilità di infezione a 0.08 il picco si alzi sensibilmente.

Diminuire la size da 600 a 400, come mostrato in fig- 3a, determina un'epidemia molto più violenta, con un picco alto e la totalità della popolazione che contrae il virus. Aumentarla a 800 (fig. 3b), simulando una sorta di distanziamento sociale, provoca invece l'effetto opposto.

Introducendo un periodo di incubazione si ottiene il grafico in fig. 4a, in cui si può notare come l'intensità dell'epidemia sia più modesta. Infine, aggiungendo la possibilità di essere costretti alla quarantena, si ottiene il grafico in fig. 4b, in cui il numero di infetti è costantemente sotto controllo e il numero finale di individui che non contraggono il virus consistente. Inoltre, il tempo di durata dell'epidemia è più del doppio di quello della versione senza quarantena.

4 Strategia di testing

Per testare la correttezza del programma si è utilizzato Doctest. Si è scelto di testare esplicitamente solo le classi *Motion* e *Infection* e il costruttore di *Simulation State*, dato che costituiscono il cuore del programma e sono le più suscettibili ad errori. Per le classi *Display* e *Plot*, che gestiscono l'output grafico, ci si è limitati a verificarne il corretto funzionamento durante l'esecuzione del programma.

Simulation State

Nel testare il costruttore di *Simulation State* si è verificato che esso generasse persone all'interno dei confini e con velocità nulla. Inoltre, calcolando la media delle posizioni, è stato verificato che la popolazione fosse distribuita uniformemente nello spazio.

Motion

Per testare la classe *Random Motion*, si è creata un'istanza con deviazione standard nulla, così da poter prevedere lo spostamento delle persone. Si è poi generata una popolazione di un solo individuo e si è verificato che la posizione venisse aggiornata correttamente, tenendo conto della velocità iniziale, della presenza dell'attrito e degli urti con le pareti. In seguito si è creata una seconda istanza di *Random Motion* con deviazione standard non nulla e si è controllato che le medie delle posizioni, velocità e accelerazioni della popolazione fossero compatibili con una distribuzione delle accelerazioni di media zero.

Infection

Le due classi che gestiscono l'infezione sono state testate con popolazioni ridotte, senza aggiornare la posizione degli individui. Gli individui sono stati distanziati in modo da poter prevedere i contatti e i parametri probabilistici delle classi sono stati impostati in modo da generare eventi certi o impossibili. Si è poi verificato che i contagi avvenissero come previsto, controllando sia la composizione dei vettori sia il *Sub Status* degli individui.

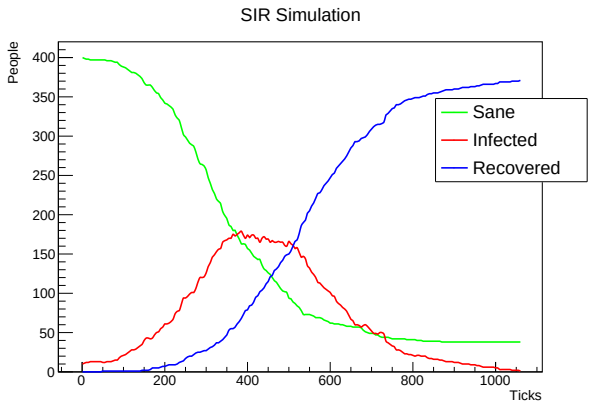
Riferimenti bibliografici

[1] Repository GitHub `https://github.com/lorenzomanini/SIR_simulation`

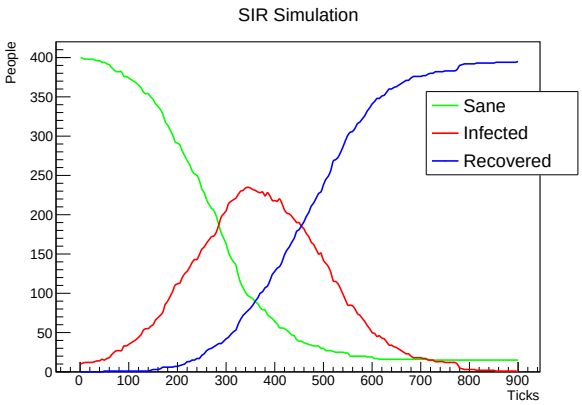
[2] CMake `https://cmake.org/`

[3] SFML: Simple and Fast Multimedia Library `https://www.sfml-dev.org/`

[4] ROOT `https://root.cern.ch/`

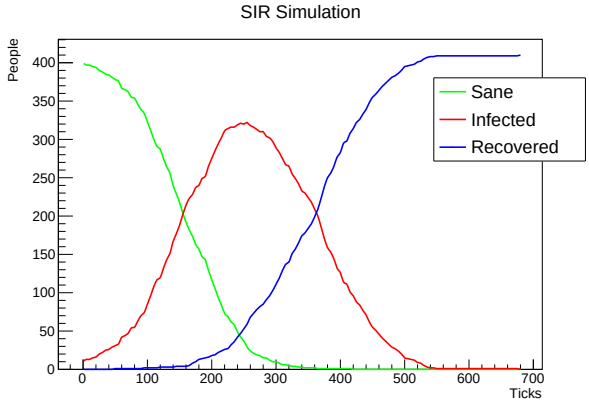


(a) Probabilità: 0.05

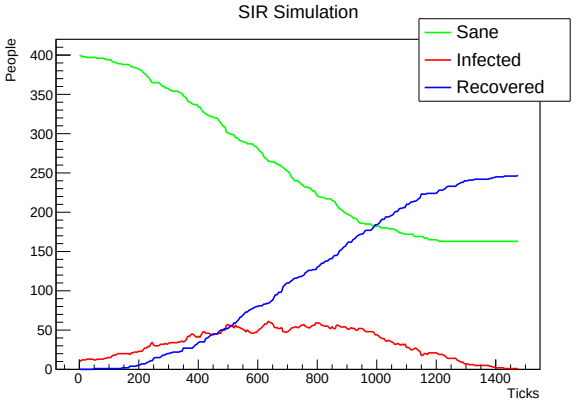


(b) Probabilità: 0.08

Figura 2: Grafici di una simulazione con *Simple Infection* a gestire l'infezione. I parametri della classe differiscono solo per la probabilità di infettarsi venendo a contatto con un individuo infettivo

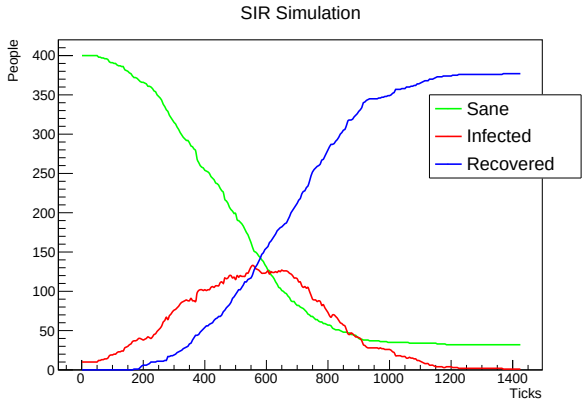


(a) Size: 400

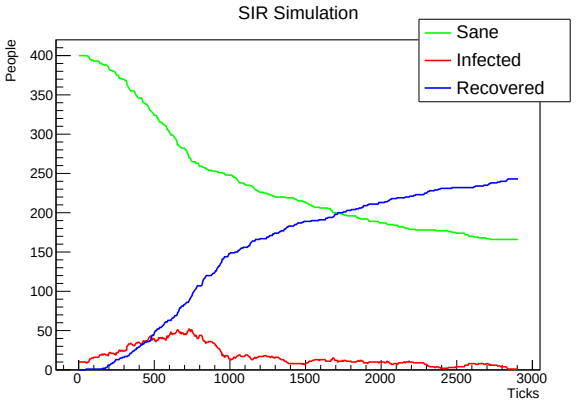


(b) Size: 800

Figura 3: Grafici di una simulazione con *Simple Infection* a gestire l'infezione. I parametri della classe differiscono solo per la dimensione dello spazio a disposizione



(a) Senza quarantena



(b) Con quarantena

Figura 4: Grafici di una simulazione con *Incubation Infection* a gestire l'infezione. I parametri della classe sono gli stessi, ma nel secondo si è aggiunta la possibilità di essere costretti alla quarantena