Progetto Corso Programmazione per la Fisica

A.A. 2020/2021

Gabriele D'Anesio, Andrea Labate, Michael Mancini

27/08/2021

Repository GitHub: https://github.com/stuckk14/progetto

Introduzione

In questo progetto è stata implementata una simulazione di un'epidemia.
Il progetto è diviso in due sezioni:

- la *prima parte* è implementata sulla base del modello *SIR*, un modello matematico differenziale che descrive l'andamento temporale delle tre categorie di popolazione che esso contempla, ossia *suscettibili*, *infetti* e *rimossi*;
- la *seconda parte* simula, su basi probabilistiche, la diffusione dell'epidemia su una griglia rappresentante la Terra.

Prima Parte

La prima parte di questo progetto si basa sulla risoluzione delle equazioni del modello SIR, in una versione "discretizzata".
Il codice è diviso in tre *source files* e due *header files*, rispettivamente `pandemy.cpp`, `graphics.cpp`, `main.cpp` e `pandemy.hpp` e `graphics.hpp`.
`pandemy.cpp` contiene le definizioni delle funzioni per calcolare l'evoluzione dell'epidemia, `graphics.cpp` contiene la gestione dell'output, sia su terminale che via SFML e `main.cpp` gestisce l'input e i relativi controlli; `pandemy.hpp` contiene la definizione della classe `Pandemy`, avente per dati membri i parametri della simulazione, e le dichiarazioni delle sue member functions (definite in `pandemy.cpp`); `graphics.hpp` contiene la definizione di una classe utile alla rappresentazione in forma di grafici e la dichiarazione dei suoi metodi e di altre funzioni libere, definiti poi in `graphics.cpp`.
Ai file menzionati si aggiunge anche un file di test, chiamato `test.cpp`.
All'interno della funzione `main` in *main.cpp* vengono letti da standard input i parametri della simulazione, ossia: il numero totale di elementi della popolazione, il numero di infetti iniziali, la durata in giorni della simulazione, i parametri β e γ delle equazioni del modello e le dimensioni della finestra di visualizzazione (*Fig. 1*). In seguito viene creata un'istanza della classe `Pandemy` e viene chiamata la funzione `window`, passando come parametri la durata in giorni della simulazione, l'istanza di `Pandemy`, il numero totale di persone e le dimensioni della finestra del grafico. Tutta l'elaborazione si svolge all'interno di `window`.
Dentro tale funzione viene creata una finestra di SFML e un'istanza della classe Graphics, vengono disegnati gli assi del grafico, attraverso il metodo `drawAxis` e viene eseguito il for-loop che gestisce l'evoluzione. A ogni iterazione, lo stato delle tre variabili viene stampato in forma tabellare sul terminale e vengono disegnati i tre punti corrispondenti sul grafico. Al termine dell'evoluzione,

vengono aggiunte le etichette agli assi, il titolo del grafico e la legenda e il grafico viene mostrato e il programma rimane in attesa dell'evento di chiusura della finestra di SFML.
Come precedentemente accennato, l'applicazione delle equazioni discretizzate del modello SIR avviene attraverso i metodi della classe `Pandemy`. In particolare, sono presenti un metodo chiamato `evolve`, che si occupa di calcolare l'evoluzione giornaliera mediante la discretizzazione per $\Delta T = 1$ delle equazioni del modello SIR, un metodo `getState`, che permette di accedere al valore giornaliero delle variabili S, I e R, e un metodo `invariant`, che verifica che sia mantenuto l'invariante di classe (e in particolare controlla che la somma di S, I e R sia sempre uguale a N e che queste tre variabili siano sempre positive).

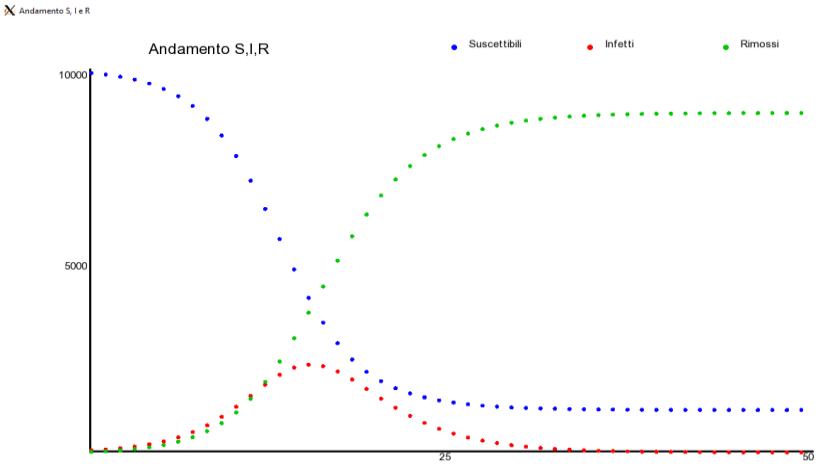
Su standard output viene visualizzata una tabella con i valori dei suscettibili, infetti e rimossi per ogni giorno trascorso (*Fig. 1*). Viene anche aperta una finestra di SFML contenente un grafico dell'andamento dei valori (*Fig. 2*).

Abbiamo usato *Doctest* per eseguire alcuni test, principalmente per verificare che il risultato ottenuto attraverso il programma corrispondesse (entro certi margini tollerabili) con quello ottenuto risolvendo le equazioni discretizzate. Inoltre, tutti i parametri inseriti in input sono controllati, per verificare che rientrino nel range di valori accettabili (*Fig. 3*). Inoltre, la presenza di *assertions*, che valutano l'invariante di classe in esecuzione, aiuta ad assicurare la corretta esecuzione del codice.

```
andrealab@LAPTOP-TNUT3DPR:~/progetto/Parte1/bin$ ./parte1
Inserire il numero di elementi della popolazione
10000
Inserire il numero di infetti iniziali
40
Inserire la durata in giorni della simulazione
50
Inserire i parametri beta e gamma relativi alla simulazione
.7
.3
Inserire le dimensioni della finestra del grafico
1280 720

Tabella riassuntiva
                                STATUS:

DAY:        SUSCEPTIBLES:        INFECTED:          REMOVED:          TOTAL: 10000
==========================================================================================
Failed to enable udev monitor, joystick connections and disconnections won't be notified: -22
Setting vertical sync not supported
1.              9932                56                12
------------------------------------------------------------------------------------------
2.              9893                78                29
------------------------------------------------------------------------------------------
3.              9839                109               52
------------------------------------------------------------------------------------------
4.              9764                151               85
------------------------------------------------------------------------------------------
5.              9661                209               130
------------------------------------------------------------------------------------------
6.              9520                287               193
------------------------------------------------------------------------------------------
7.              9329                392               279
------------------------------------------------------------------------------------------
```




```
andrealab@LAPTOP-TNUT3DPR:~/progetto/Parte1/bin$ ./parte1
Inserire il numero di elementi della popolazione
10000
Inserire il numero di infetti iniziali
40
Inserire la durata in giorni della simulazione
50
Inserire i parametri beta e gamma relativi alla simulazion
1.3
0.3
Beta e' fuori dal range
andrealab@LAPTOP-TNUT3DPR:~/progetto/Parte1/bin$
```

Fig. 1 (in alto a sinistra): l'utente inserisce attraverso standard input i parametri richiesti; se tutti i parametri sono validi, viene stampata su terminale una tabulazione dei valori delle tre categorie del modello SIR (l'immagine si ferma al giorno 7 ma la tabulazione continua, in questo caso, fino al giorno 50).

Fig. 2 (in basso a sinistra): finestra di SFML mostrata contestualmente alla tabulazione su terminale, che esprime in forma grafica i valori sopra menzionati.

Fig. 3 (in alto): uno dei messaggi di errore generati dopo il tentativo di inserimento di un valore di un parametro errato (in questo caso β), in conseguenza del quale l'esecuzione si arresta.

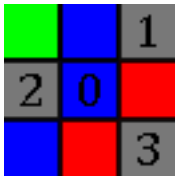
Seconda Parte

La seconda parte del progetto sviluppa un approccio probabilistico per simulare un'epidemia mondiale. La griglia di evoluzione coincide, pertanto, con la superficie terrestre; l'immagine utilizzata nella simulazione è contenuta in `Mondo_grande_porti.psd`. In particolare, vengono simulati suscettibili, infetti e rimossi (divisi in morti e guariti). In aggiunta, sono simulati anche gli spostamenti delle persone verso caselle libere, oltre che porti e aeroporti, per permettere il loro spostamento tra continenti.
Tale sezione si compone di tre *source files* e due *header files* con gli stessi nomi dei file della prima parte, e con finalità simili; a questi si aggiunge un file denominato `input.dat` in cui l'utente può inserire i parametri della simulazione che costituiscono l'input del programma (*nDays, beta, gamma, deathRate, lockdownLimit, daysToDeath, nVaccinated* - il loro significato sarà chiarito a breve).
La posizione e il numero di infetti e celle libere iniziali sul planisfero sono impostati in input dall'utente attraverso il mouse: l'utente attraverso un singolo click o un doppio click su una determinata celletta inizialmente suscettibile può decidere di trasformarla rispettivamente in cella infetta o vuota.
Nella mappa sono anche prese in considerazione zone della Terra con densità di popolazione bassa o nulla. Queste zone non possono essere contagiate e non possono essere attraversate dagli spostamenti delle persone.
Come misure di contrasto all'epidemia, sono simulate vaccinazioni e possibili lockdown.
Le prime sono rappresentate da un numero di individui *nVaccinated* che a partire dal giorno $T = nDays/5$, per una durata di dieci giorni passano automaticamente da suscettibili a guarite. Il parametro *nDays* rappresenta la durata in giorni della simulazione.
Il lockdown, invece, è implementato attraverso una modifica del parametro β, indice della probabilità del contagio: quando il rapporto di infetti rispetto ai suscettibili supera il valore *lockdownLimit*, il parametro viene diminuito di un fattore 2.5. Nel caso in cui suddetto rapporto dovesse scendere al di sotto del 90% di *lockdownLimit*, verrebbe ripristinato il valore iniziale di β e il lockdown cesserebbe.

Tutti gli aspetti dell'evoluzione sono gestiti su base probabilistica e usano il generatore LCG di default del C++ (`default_random_engine`).
La probabilità che una determinata cella suscettibile si infetti è data dalla formula: $prob_{inf} = \beta \frac{N_{inf}}{8}$, dove N_{inf} rappresenta il numero di celle infette adiacenti. Questa probabilità è poi confrontata con il valore assunto da una variabile continua uniforme tra 0 e 1: se tale valore è inferiore a $prob_{inf}$, allora la cella si infetta, altrimenti si mantiene suscettibile.
La probabilità, invece, che una cella infetta guarisca (o muoia) è data da γ. Una volta che un'estrazione simile alla precedente ha deciso che un infetto passerà a rimosso, un'altra analoga estrazione separa i morti dai guariti, usando il parametro *deathRate* fornito in input. Le eventuali celle morte vengono svuotate in un giorno, per simulare la sepoltura o cremazione dei morti.
Per simulare il periodo di decorso della malattia, tutti gli infetti restano tali per un tempo almeno pari a *daysToDeath*.

Lo spostamento di una cella viva verso una cella vuota adiacente è gestito attraverso una generazione uniforme intera, il cui risultato viene usato come indice per scegliere in quale casella adiacente spostarsi (in caso venga estratto 0, non viene eseguito nessuno spostamento). Gli indici sono numerati dal primo in alto a sinistra all'ultimo in basso a destra, assegnando zero alla posizione di partenza.

Esempio di spostamento:
I quadrati grigi rappresentano le celle vuote. Se viene estratto il numero 2, il suscettibile che si trova al centro si sposterà nella cella alla sua sinistra



Un meccanismo simile è sfruttato per lo spostamento tra porti o tra aeroporti: viene creato un vettore contenente tutti i porti o gli aeroporti disponibili (con almeno una cella vuota adiacente) e viene estratto casualmente l'indice di tale vettore.

L'implementazione con SFML si basa su VertexArray di quadrati, introdotti per limitare il numero di chiamate a `sf::RenderWindow::draw`. Esso consiste in un array di punti che a gruppi di quattro rappresentano dei quadrilateri (nel nostro caso, dei quadrati).
Lo stato iniziale della simulazione (in cui tutte le celle disponibili sono suscettibili) viene creato scorrendo un pixel alla volta l'immagine della Terra in formato .psd e controllandone il colore.
Partendo dallo stato iniziale appena impostato, viene creato il VertexArray e il programma attende che l'utente selezioni quali celle vuole rendere inizialmente infette o vuote (cliccandoci sopra con il puntatore), una volta terminata la scelta iniziale e premuto un qualsiasi tasto della tastiera, parte l'elaborazione.
Questa è gestita attraverso il multithreading: mentre il thread principale crea il VertexArray riferito al giorno *n* e lo mostra in output, un thread secondario si occupa di elaborare l'evoluzione del giorno successivo (*n*+ 1). Il fatto che i due thread lavorino su dati diversi e separati assicura che non possano avvenire race conditions.

La visualizzazione impiega diversi colori in base allo stato della singola cella:
- Blu: cella suscettibile
- Rosso: cella infetta
- Verde: cella guarita
- Giallo: cella morta
- Grigio scuro: zone inabitabili
- Grigio chiaro: porti
- Bianco: aeroporti
- Nero: acqua

Come nella prima parte, tutti i parametri letti in input da file sono controllati per verificare che si trovino all'interno del range previsto. In caso contrario, l'esecuzione del programma termina con un errore. Anche in questo caso sono presenti *assertions* per verificare la correttezza dell'esecuzione. In particolare, esse valutano che l'indice di accesso al vettore griglia (`m_grid`) non esca dal range permesso.
In *Fig. 4* è mostrato un esempio di ciò che l'utente visualizza in esecuzione dopo aver impostato i parametri della simulazione da file e aver scelto le celle da infettare o da svuotare sulla mappa.

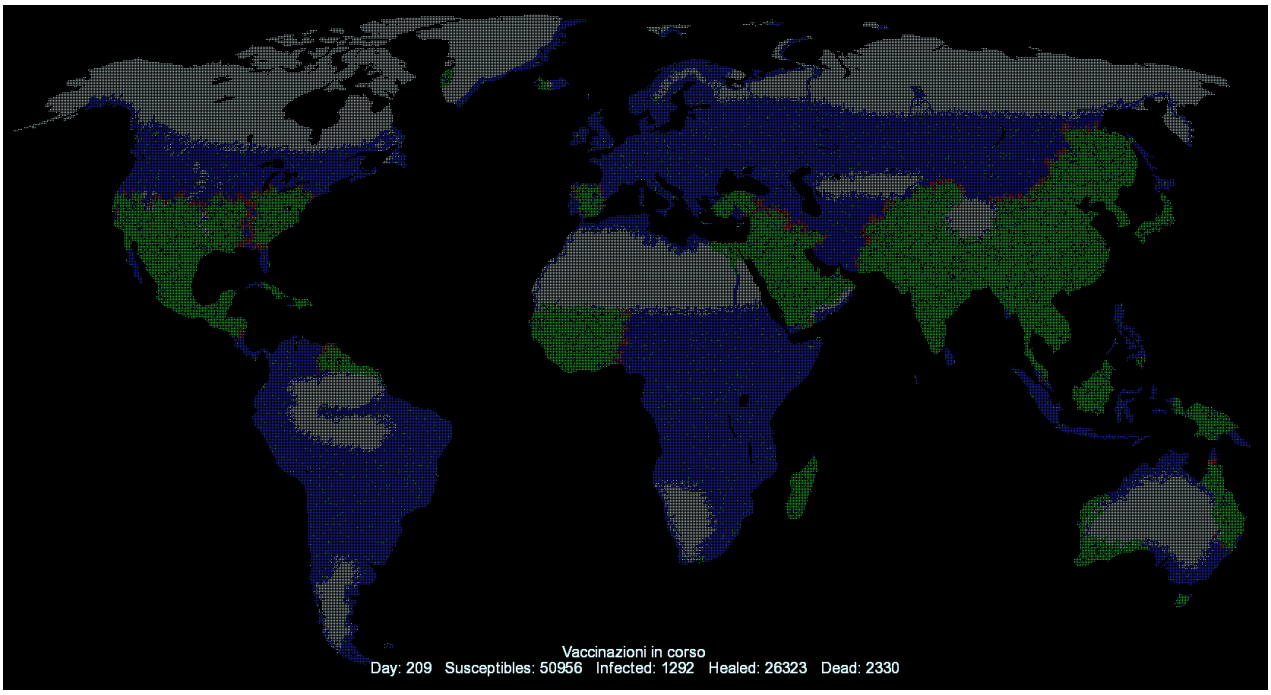


Fig. 4: mappa che mostra l'evoluzione dell'epidemia nel corso del tempo; la riga di testo in basso mostra i valori correnti, mentre la riga immediatamente più in alto viene adoperata per mostrare varie fasi della simulazione (lockdown, vaccinazioni): in questo caso essa mostra che sono in corso le vaccinazioni. I parametri della simulazione in questo specifico caso sono:

nDays = 1000 *deathRate = 0.09* *nVaccinated = 1500*

beta = 0.7 *lockdownLimit = 0.025*

gamma = 0.2 *daysToDeath = 7*

Compilazione

Per compilare il progetto è consigliato l'uso di *CMake*: recarsi nella directory principale e lanciare i seguenti comandi:

```
mkdir build
cd build/
cmake ..
```

Per migliorare le performance si consiglia di compilare in *Release mode*. Per farlo scrivere
`cmake -DCMAKE_BUILD_TYPE=Release ..`

In questo modo, verranno creati tutti i file di configurazione di CMake.
Per compilare, scrivere

`make parte1` Per compilare solo la prima parte

`make parte2` Per compilare solo la seconda parte

`make test`	Per compilare solo l'eseguibile con i test
`make`	Per compilare tutti gli eseguibili

Gli eseguibili si troveranno nelle directory

`progetto/Parte1/bin`	Prima parte
`progetto/Parte2/bin`	Seconda parte
`progetto/Parte1/testbin/`	Test

In caso si volesse compilare con g++, per esempio la prima parte, bisognerà lanciare il comando (in questo specifico caso dalla directory `progetto/Parte1/`)

```
g++ -Wall -Wextra main.cpp graphics.cpp pandemy.cpp -lsfml-graphics -lsfml-system -lsfml-window -fsanitize=address
```

Per compilare in quest'ultimo modo la seconda parte, è conveniente creare prima una sottodirectory di `progetto/Parte2` e generare l'eseguibile in questa sottodirectory. Infatti la funzione *`main`* cerca il file con i parametri di input nella *parent directory* di quella che contiene l'eseguibile finale.
Può capitare che in seguito alla chiusura della finestra di SFML della seconda parte, l'address sanitizer rilevi qualche memory leak: ciò è da imputare a SFML.