Progetto d'esame di Programmazione per la Fisica

Cristina Caprioglio, Luca Morelli

July 5, 2021

Abstract

Poter prevedere l'evoluzione di una pandemia, come gli eventi di quest'ultimo anno ci hanno insegnato, risulta essenziale per l'organizzazione del contenimento di quest'ultima. Tale operazione può essere delegata ad un calcolatore che, sfruttando modelli matematici, è in grado di stimare il numero di infetti e sani di uno specifico giorno. Utilizzando il modello SIR (Suscettibili, Infetti, Rimossi) abbiamo implementato un programma capace di effettuare stime sull'evoluzione di una pandemia giorno per giorno. Abbiamo anche realizzato un programma capace di simulare, tramite un automa cellulare, la diffusione di un virus in una popolazione ristretta.

1 Parte prima: Il modello SIR

1.1 Introduzione

In questa prima parte implementeremo il modello SIR, con alcune variazioni, su codice C++. Si fa uso principalmente di 5 parametri: Beta, che rappresenta la probabilità di infettarsi, Gamma, che rappresenta la probabilità di guarire, S, il numero di persone suscettibili, I in numero di infetti e R, il numero di rimossi. Sono inoltre utilizzati anche altri parametri quali il numero di vaccini effettuati per giorno, dopo quanti giorni iniziano le vaccinazioni e quanto debba durare la simulazione. Il codice necessario per simulare una popolazione tramite modello SIR si compone di 2 header files e 3 files di implementazione vera e propria (tra cui uno con i test).

1.2 Implementazione del modello SIR

Nei files denominati Sir (Sir.cpp, Sir.hpp e Sir.test.cpp) vi è il codice necessario per la simulazione vera e propria, in essi abbiamo utilizzato il namespace Simulation. Nell'header Sir.hpp abbiamo definito una struct Data e una classe Population, con le sue funzioni membro, e abbiamo dichiarato le funzioni Evolve, Simulate e Print, poi definite in Sir.cpp. La struct Data, oltre ai valori relativi al numero di suscettibili (S), infetti (I) e rimossi (R), contiene anche alcune costanti di massimo e minimo necessarie per eventuali controlli eseguiti affinchè la simulazione sia di senso compiuto. La classe Population è composta di una parte privata, ove sono dichiarati una variabile state di tipo Data, due double beta e gamma, il numero di vaccini per giorno e una stringa che indica la presenza o meno della quarantena, e una pubblica ove sono presenti due costruttori e le funzioni membro necessarie per accedere ai dati privati. La presenza di un overload di costruttori è da imputare alla necessità di poter costruire un oggetto di tipo Population anche utilizzando i soli parametri Beta e Gamma cosichè gli altri parametri possano essere valutati in un secondo momento. In ogni costruttore sono presenti dei controlli effettuati tramite assert che verificano che la costruzione di un oggetto avvenga con dati di senso (per esempio con un numero positivo di persone).

Nel file Sir.cpp sono state inizializzate alcune variabili globali costanti utilizzate per semplificare la lettura del codice e contenenti le dimensioni delle varie parti della tabella che viene stampata a terminale. Oltre a queste è presente anche una costante utilizzata per poter consentire l'arrotondamento dei double ad interi durante l'utilizzo degli static cast. La funzione Evolve si occupa di determinare il numero di Suscettibili, Infetti e Rimossi del giorno successivo secondo le formule del modello SIR; questo calcolo implica la conversione da double ad intero dei valori calcolati, tale passaggio avviene con la funzione static cast, al cui argomento viene sommato 0.5 per evitare il troncamento ed effettuare un'approssimazione. Sono inoltre presenti alcuni assert per assicurarsi che S, I e R non abbiano mai valore minore di 0; in tal caso, tramite una serie di if posizionati prima dei controlli, i valori vengono riequilibrati in maniera tale che la simulazione prosegua con senso. Sono stati poi aggiunti dei controlli per correggere la fluttuazione del numero totale di persone causato dagli errori di approssimazione, in modo che esso restasse costante per ogni giorno. Abbiamo anche implementato una simulazione della quarantena che si attiva qualora gli infetti siano maggiori di un terzo della popolazione, in tal caso beta viene dimezzato fino a quando essa resta attiva. Inoltre se i vaccini sono attivi viene sottratto dal numero di suscettibili tante persone quanto il numero di vaccini giornalieri, che vengono poi aggiunte ai

rimossi. In questo caso vi è un controllo sul numero di S tale per cui questo non possa mai essere negativo, mentre nel caso sia minore del numero giornaliero di vaccini si porta S a 0 e si aumenta R delle persone che erano rimaste in S. La funzione Simulate invece si occupa di creare un vettore di tipo Population contenente i risultati dell'intera simulazione, per farlo chiama la funzione Evolve sull'i-esimo elemento del vettore per poter ottenere il successivo. Utilizzando un vettore come output il risultato della simulazione è interamente accessibile da qualsiasi funzione successiva così che sia più semplice per eventuali future implementazioni interagire con i risultati. Anche qui sono presenti alcuni controlli: un primo conclude la simulazione se le variazioni di S,I o R sono troppo piccole da essere eliminate tramite le approssimazioni ad intero, un secondo che, qualora il numero di infetti calasse a zero, essendo in tal caso cessata la pandemia, termina la simulazione e un terzo che si occupa di iniziare a conteggiare i vaccini una volta raggiunto un giorno prestabilito. Infine, la funzione Print si occupa di stampare a terminale una tabella contenente i vari dati giorno per giorno, utilizzando un iteratore che viene spostato tramite un ciclo per tutto il vettore restituito dalla funzione evolve e che consente di accedere tramite i metodi di Population ai dati di ogni giorno. Abbiamo utilizzato la funzione std::setw per poter gestire gli spazi delle varie celle della tabella. Una serie di if si occupano di calcolare il numero di vaccinati totali in base al numero dei vaccini giornalieri e delle persone suscettibili, tale calcolo è dovuto al fatto che i vaccinati non sono conteggiati come una categoria a se ma come rimossi. Il file sir.test.cpp è relativo a questi due files (sir.hpp e sir.cpp) e contiene quattro test cases. Il primo si occupa di testare la struct, il secondo invece testa la classe Population con i relativi operatori e funzioni membro. Il terzo, relativo alla funzione Evolve, racchiude diversi subcases in modo da verificare le varie casistiche: nel caso in cui Beta fosse minore di Gamma e ci fosse la quarantena, si dovrebbe verificare che il giorno seguente I sia aumentato più di R e che Beta si sia dimezzato, mentre nel caso in cui la Quarantena sia attiva e Gamma sia maggiore di Beta, I dovrebbe diminuire e R aumentare, insieme al dimezzamento di Beta. Si sono inoltre testate queste due casistiche quando S fosse praticamente quasi tutta la popolazione: quando questo accade la pandemia non parte a meno che il rapporto Beta/Gamma sia maggiore di 1, come si è verificato con i test. In particolare i subcase sono stati utili per trovare e correggere gli errori sui valori relativi alla fluttuazione del numero di persone nei vari casi. L'ultimo case con i relativi subcases riguarda la funzione Simulate: il primo subcase si occupa di controllare che i numeri di S, I e R, senza l'adozione di vaccini, dopo un numero fissato di giorni corrispondessero, a meno di un'approssimazione, a quelli che si troverebbero calcolandoli con un foglio di calcolo con l'uso del modello SIR, il secondo svolge lo stesso controllo ma con la possibilità di vaccinare dal giorno iniziale, mentre l'ultimo controlla che, se messa come data d'inizio vaccinazioni un giorno diverso dal primo, esse inizino effettivamente in quel giorno, con le relative modifiche ai valori del modello.

1.3 Implementazione grafica

Nei file graph (graph.cpp e graph.hpp) vi è il codice utilizzato per rappresentare graficamente i risulati utilizzando la libreria SFML, tale codice è contenuto nel namespace Display. Nel file graph.hpp sono semplicemente dichiarate le funzioni per la parte grafica. In graph.cpp prima delle funzioni sono state inizializzate alcune variabili globali costanti utilizzate per facilitare la lettura del codice e per specificare alcune dimensioni grafiche. Le funzioni di questa porzione di codice richiedono tutte una referenza ad un oggetto RenderWindow di SFML che non è altro che la finestra grafica su cui vengono mostrati i risultati. Le tre funzioni print_I, print_S e print_R sono molto simili tra loro e stampano per ogni giorno una colonna di larghezza in pixel di un giorno e altezza in pixel dipendente da vari parametri, rispettivamente di colore rosso per I, verde per S e azzurro per R. Fatta eccezione per la prima colonna, la cui altezza è fissa, le altre sono alte 3 più il valore assoluto della differenza tra il numero di persone della categoria in quel giorno e il numero del giorno prima, in modo da ridurre il più possibile gli spazi rendendo il grafico simile ad una linea curva continua. La funzione print_axis invece si occupa di stampare gli assi del grafico con le relative tacche ed etichette. Tutte le dimensioni e le distanze sono calcolate in proporzione al numero di persone e di giorni presi in input rispetto alla dimensione della finestra grafica e della legenda. Poichè in questa porzione di codice occorre stampare caratteri con SFML è necessario fornire al programma un font nella directory di lavoro (font.ttf); tale font potrebbe, secondo la documentazione di SFML, non essere caricato correttamente, in tal caso viene sollevato un Runtime Error che verrà gestito nel main. Infine, la funzione print_caption stampa la legenda relativa al grafico, con etichette e riquadri col bordo nero colorati in base a cosa indicano (S, I o R). Vi sono anche il numero totale di persone e la durata della simulazione. Le posizioni dei box sono calcolate in modo che siano centrati tra loro e rispetto allo spazio della legenda nella finestra grafica, mentre le posizioni delle etichette e delle scritte sono centrate rispetto ai relativi riquadri.

1.4 Il Main

Nel main.cpp sono presenti due variabili globali costanti necessarie per la creazione delle finestre in quanto contengono le loro dimensioni e la dimensione della legenda. Vi è poi una free function Get_parameters che si occupa di prendere in input le variabili per Population e svolgere dei controlli in modo che rispettino i valori

minimi e/o massimi. Alcuni parametri quali la durata della simulazione, il giorno in cui inizia la vaccinazione e il numero di vaccini disponibili per giorno sono chiesti in input esternamente a questa funzione poichè il loro valore è necessario durante tutto il programma e un oggetto Simulation::Population non li contiene. Nella funzione main è presente un try-catch all'interno di un ciclo while il cui scopo è ripetere la richiesta di valori in input fino a quando non rispettano le condizioni. A tale scopo, qualora venisse trovato un errore dovuto all'input, oltre a darne notifica, il catch si occupa di svuotare lo stream in input con apposite funzioni membro di cin. Viene quindi costruito un vector di tipo Simulation::Population che viene riempito da tutti gli stati simulati tramite l'apposita funzione Simulation::Evolve. Tale vettore viene quindi utilizzato per la stampa a terminale da Simulation::Print. Quindi viene creata una finestra grafica con sfondo bianco su cui vengono stampati i grafici, gli assi e la legenda. In ultima battuta si utilizza un ciclo while che si occupa di tenere aperta la finestra fino a quando non viene chiusa. Qualora si dovesse chiudere la finestra grafica, tramite un evento precedentemente dichiarato, la finestra viene chiusa, il ciclo while iniziale termina e con esso anche il programma.

1.5 Interfaccia del programma

Inizialmente è necessario inserire a terminale i parametri della simulazione (Figura 1), che devono rispettare alcune condizioni: Beta e Gamma devono essere compresi tra 0 e 1 inclusi, S, I e R devono essere maggiori di zero ed interi, così come per la durata della simulazione, possono essere invece anche pari a zero il numero di vaccini giornalieri e il giorno in cui inizia la vaccinazione. Una volta inseriti tutti i parametri richiesti dalla simulazione automaticamente verrà visualizzato l'output (Figura 2) con i risultati: questo consta di una tabella stampata a terminale sulla quale sono riportati giorno per giorno i valori di S, I e R con informazioni aggiuntive inerenti la quarantena e le vaccinazioni. In aggiunta è mostrato un grafico che mostra l'andamento funzionale di tutti e tre i parametri S, I e R distinti da opportuni colori a cui si fa riferimento nella legenda.

```
lyokol@MorellisNotebook:~/Progetto/Parte_1/Implementazione_grafica$ ./SIR_Simulation.out
Insert Beta, Gamma, S, I, R, and if people can get vaccinated (yes/no): 0.1 0.08 10000 100 4 yes
Insert number of vaccines per day: 25
Insert number of days of simulation: 10000
Insert from what day they start vaccinating: 300
```

Figure 1: Esempio di interfaccia di Input

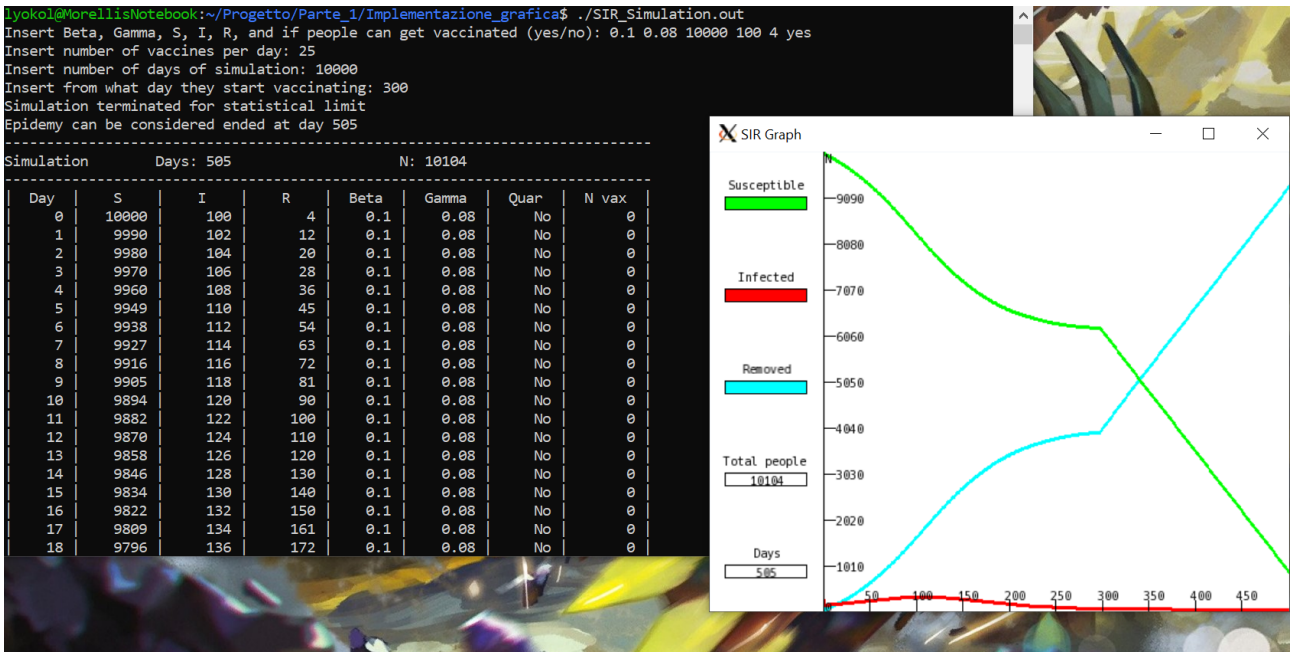

Figure 2: Esempio di interfaccia di Output

1.6 Considerazioni

Il programma realizzato si trova in accordo con il modello SIR, come viene mostrato nei vari test di confronto con calcoli esterni e con la dipendenza dal rapporto β/γ. Inoltre le implementazioni relative quarantena e vaccinazioni generano effettive variazioni dello stato della simulazioni in accordo con il buon senso. (Figura 3) Seppure il codice in sè non generi errori di compilazione, nè con l'uso dell'address sanitizer nè con l'uso delle opzioni -Wall -Wextra, è noto che la libreria SFML è sorgente di memory leaks (Figura 4) che sono presenti nel nostro codice ma che, poichè causati da una libreria esterna, non sono da noi correggibili.

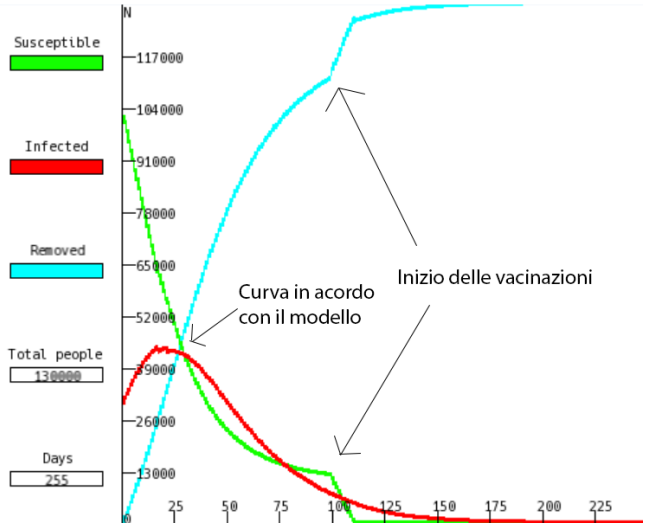

Figure 3: Considerazioni sul grafico

```
=================================================================
==397==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 64 byte(s) in 1 object(s) allocated from:
    #0 0x7fc496713947 in operator new(unsigned long) (/lib/x86_64-linux-gnu/libasan.so.5+0x10f947)
    #1 0x7fc49135f78d  (<unknown module>)

Indirect leak of 56 byte(s) in 1 object(s) allocated from:
    #0 0x7fc496711dc6 in calloc (/lib/x86_64-linux-gnu/libasan.so.5+0x10ddc6)
    #1 0x7fc49160aab5  (<unknown module>)

Indirect leak of 56 byte(s) in 1 object(s) allocated from:
    #0 0x7fc496711dc6 in calloc (/lib/x86_64-linux-gnu/libasan.so.5+0x10ddc6)
    #1 0x7fc49160aa9e  (<unknown module>)

SUMMARY: AddressSanitizer: 176 byte(s) leaked in 3 allocation(s).
```

Figure 4: Esempio di memory leak rilevato

2 Parte seconda: L'automa cellulare

2.1 Introduzione

In questa seconda parte implementeremo un programma capace di simulare una pandemia tramite un automa cellulare. I parametri considerati per tali simulazione sono: la dimensione della griglia su cui avviene la simulazione (il lato), Beta, ossia la probabilità di infettarsi, Gamma, ossia la probabilità di guarire, Theta, ovvero la probabilità di essere vaccinato. Inoltre vi sarà la possibilità di simulare l'uso delle mascherine e del lockdown. Il codice necessario per la simulazione con automa cellulare è distribuito tra più files: il main, 2 header files e 3 files di implementazione, compreso quello con i test.

2.2 Implementazione dell'automa cellulare

La simulazione dell'automa vera e propria avviene nei file plague (plague.hpp, plague.cpp e plague.test.cpp) all'interno del namespace Simulation. L'header file plague.hpp contiene la classe World con gli operatori ad

essa relativi necessari per i test e le dichiarazioni delle 7 free functions poi definite nel relativo file plague.cpp. Inoltre sono definiti una serie di enum class finalizzati al controllo degli stati delle singole celle oppure alle opzioni della simulazione stessa. L'enumerazione Person si riferisce ai vari stati in cui una persona può trovarsi: S per suscettibile, I per infetta, R per rimossa, V per vaccinata ed infine E per empty nel caso la cella sia vuota. Le rimanenti enumerazioni (Mask, Lockdown e Vax) hanno come valori ON e OFF rispettivamente nel caso siano attive le relative misure di contenimento o meno. La parte privata della classe World utilizza Grid come alias per un vettore di tipo Person, che costituisce l'insieme di tutte le persone dell'automa, ed è dichiarato con il nome grid. Inoltre vi sono vari parametri di tipo int o double quali la dimensione del lato della griglia della simulazione, beta, gamma e theta accompagnati da tre variabili di tipo Mask Lockdown e Vax che indicano lo stato di tale misura di contenimento e sono inizializzate ad OFF di default. Infine sono dichiarate due costanti statiche che consentono rispettivamente di avere una referenza ad una cella al di fuori della griglia (di default vuota) e il valore delle coordinate delle celle esterne nel bordo superiore e di sinistra. Nella parte pubblica sono inserite alcune costanti statiche, così che che possano essere chiamate nel resto del programma, le quali restituiscono i valori limite per i parametri in uso. Il costruttore dà origine ad una griglia quadrata in cui tutte le celle vengono inizialmente considerate come delle persone suscettibili e necessita come argomento la dimensione di un lato della griglia e i valori di beta gamma e theta. Il costruttore contiene nel suo scope una serie di assert finalizzati a verificare il senso dei dati utilizzati per la costruzione di un oggetto world (per esempio che le probabilità beta gamma e theta siano comprese tra 0 e 1). Il primo metodo definito consente di accedere allo stato di una cella date le sue coordinate ed è implementato tramite un overload di due funzioni: la prima consente di accedere in sola lettura agli stati delle celle, assegnando lo stato E (empty) alle celle fuori dalla griglia, il che risulta necessario per poter poi valutare l'evoluzione delle persone sul bordo, mentre la seconda permette di modificare lo stato di una persona e pertanto non è presente il caso relativo alle celle fuori dalla griglia. Una serie di assert verifica che le coordinate date siano valide. Sono presenti anche alcune funzioni membro che restituiscono il numero di persone appartenenti ad un determinato stato, semplicemente tramite un conteggio, e altri metodi che restituiscono o il valore o una stringa con lo stato delle varie misure di contenimento: questo è necessario per agevolare la stampa di questi. Successivamente find_E restituisce un vettore con le coordinate di tutte le celle vuote, in tal caso le coordinate sono disposte due a due nel vettore e la free function che chiama questo metodo è progettata sapendo che nel vettore le coordinate x e y sono una di seguito all'altra. Le ultime tre funzioni membro, di tipo void, permettono di modificare gli stati delle misure di contenimento.

All'inizio di plague.cpp sono presenti alcune variabili globali relative al massimo e al minimo che può essere generato randomicamente dalle due distribuzioni di cui si farà uso, e ad altri dati che così dichiarati rendono più comprensibile la lettura del codice. Subito dopo vi sono le definizioni degli oggetti della libreria random necessari per la generazione dei numeri casuali, realizzata attraverso un engine che estrae a caso da due distribuzioni uniformi un valore utilizzato per rendere casuale, nota una certa probabilità che avvenga un evento, la possibilità di infettarsi o guarire o esser vaccinati e lo spostamento in celle vuote di celle piene. Vi è quindi la definizione della funzione di tipo int I_near che attraverso una serie di cicli for controlla le celle limitrofe ad una cella in analisi e ne restituisce il numero di infette. Successivamente vi è la definizione della funzione di tipo Person person_next_status che si occupa di determinare lo stato delle varie celle nel giorno successivo. Prima di tutto si occupa di controllare se sono attive le mascherine, nel qual caso il valore di beta viene dimezzato tramite l'uso di una variabile beta inizializzata con il valore effettivo di questo che poi può essere modificato temporaneamente. In seguito, attraverso uno switch case, in base allo stato attuale di una persona si stabilisce quale debba essere il suo stato successivo tramite l'uso dei generatori di numeri casuali: per una persona S, se essa ha vicino almeno un infetto si verifica randomicamente se questa debba infettarsi, in questo caso tale probabilità dipende da beta e dal numero di celle infette adiacenti, se tale persona non si infetta e sono attive le vaccinazioni allora viene valutata randomicamente la possibilità che questa persona diventi di tipo V (vaccinata), tale probabilità è dettata da theta; nel caso di una persona infetta in funzione di gamma viene stabilito sempre randomicamente se essa possa diventare R (rimossa); il caso di default invece indica un'assenza di modifica e riguarda le persone rimosse, vaccinate o le celle vuote. In tutti questi casi la valutazione di carattere aleatoria è effettuata generando un numero casuale tra 0 e 1 con distribuzione uniforme: si verifica quindi che tale numero generato sia minore della probabilità che si verifichi un determinato evento (per esempio l'infezione di un sano) e solo in tale caso si modifica di conseguenza lo stato della cella. La funzione Evolve, similmente alla sua omonima nella prima parte del progetto, si occupa di cambiare lo stato delle varie celle da quello attuale a quello successivo attraverso due cicli for che controllano persona per persona utilizzando la funzione person_next_status. Le funzioni di tipo void print_intestation e print_terminal si occupano rispettivamente di stampare l'intestazione e i risultati della simulazione in una tabella a terminale in maniera analoga a quanto fatto per il codice della prima parte del progetto, i due processi sono separati per consentire una stampa in tempo reale che possa essere interrotta e ripresa più volte. La funzione swap consente semplicemente di scambiare lo stato di due celle date le loro coordinate, tale funzione è utilizzata per generare un random walk delle celle. La funzione walk si occupa di gestire il random walk e per farlo fa uso del metodo find_E, che restituisce il vettore delle coordinate di tutte le celle vuote, per ognuna di queste genera due numeri casuali compresi tra -1 e 1 e li aggiunge alle coordinate

della cella, così che si scelga casualmente quale cella delle 8 adiacenti possa spostarsi nella cella vuota: tale procedimento avviene tramite la funzione swap. Vi sono inoltre un serie di controlli per evitare che una cella possa uscire dalla griglia così che se si trova adiacente ad un bordo essa potrà spostarsi solamente lungo di esso o allontanandovisi. Nel caso sia attivo il lockdown il movimento e lo scambio tra le celle vengono bloccati tramite un controllo presente nel main. Il file plague.test.cpp contiene i test relativi ai due file precedenti: il primo test verifica che nel caso vi siano solo persone S esse non cambino il loro stato, mentre il secondo e il terzo rispettivamente controllano il funzionamento di beta e gamma, ossia che con tutte persone infette tranne una suscettibile e beta pari a 1, essa diventi infetta a sua volta e che nel caso di una griglia 1x1 con una persona infetta e gamma pari a 1 essa diventi rimossa. Il quarto test si occupa del parametro theta, il quale se vale 1, in caso di vaccinazioni attive in una griglia 1x1, porta la persona in stato V. Il quinto si occupa di verificare il comportamento delle funzioni relative al conteggio di S, I, R e E, mentre il sesto controlla che la funzione find_E funzioni correttamente. Il settimo verifica se effettivamente le funzioni walk e swap causino uno spostamento tra le celle, mentre l'ottavo controlla che le misure di contenimento varino correttamente. In questo caso eseguire i test si è rivelato più complesso in quanto il programma presenta una natura aleatoria, per questo motivo non abbiamo potuto effettuare confronti con dati esterni.

2.3 Implementazione grafica

Il codice utilizzato per poter visualizzare ed interagire con la griglia ed i risultati graficamente è stato implementato tramite l'uso di SFML nei files graph (graph.hpp e graph.cpp) e nella fattispecie nel namespace Display. Tutte le funzioni di questa sezione richiedono come argomento una referenza all'oggetto RenderWindow con cui si deve interagire. All'interno dell'header graph.hpp vengono dichiarate le funzioni poi sviluppate in graph.cpp, dove prima delle varie funzioni vengono prima dichiarate le variabili globali costanti relative alle varie dimensioni degli oggetti grafici. La prima funzione, person_size, calcola la dimensione in pixel di una cella in funzione della dimensione della finestra, per poi poter costruire un oggetto sf::RectangleShape della corretta dimensione per rappresentare una persona. La funzione print stampa la griglia con le celle i cui colori corrispondono allo stato della persona che rappresentano. In particolare, il nero corrisponde ad una cella vuota, il rosso ad una infetta, il verde ad una suscettibile, il blu ad una vaccinata e il bianco ad una rimossa. La funzione set_status permette di modificare con il click del mouse lo stato di una persona (da S a I, da I a R, da R a E e da E a S), convertendo la posizione in pixel del mouse nelle coordinate della cella relativa. Per poter interagire con il mouse è utilizzato il sistema degli eventi di SFML che gestisce automaticamente eventuali immissione multiple. Vi è inoltre un bool che si occupa di controllare che il mouse si trovi dentro la finestra, in modo da non cambiare le celle nel caso di un click fuori dalla griglia, situazione che causerebbe un undefined behavior poichè richiederebbe l'accesso a porzioni del vettore grid non definite. La funzione opt_screen_print invece stampa la finestra che consente di gestire misure di contenimento: ogni opzione (mask, lockdown e vax) dispone di un pulsante cliccabile, tramite l'uso della funzione option, e cambia il suo colore e il suo testo in funzione dello stato delle opzioni. Vi è inoltre un pulsante che si occupa di fermare la simulazione se viene cliccato. La funzione option controlla se vi è un click del mouse in corrispondenza di uno dei pulsanti: affinché questo avvenga correttamente si fa uso degli eventi di SFML combinati con particolari funzioni ed oggetti che consentono di determinare se una posizione data (quella del mouse) si trova all'interno di determinate aree definite tramite una serie di oggetti (mask_box, lockdown_box, vax_box e stop_box), nel qual caso cambia lo stato da ON a OFF o viceversa per le misure di contenimento mentre invece chiude la finestra delle opzioni se si clicca il pulsante STOP, il che, come si vedrà nel paragrafo 2.4, causa la fine della simulazione. Le funzioni print_I, print_R, print_S e print_axis sono analoghe alle loro omonime della parte 1 del progetto, con la differenza che ora vi è anche una funzione print_V relativa ai vaccinati che stampa un grafico di colore viola, e che in generale queste sono state rimaneggiate per poter interagire correttamente con una serie di dati differenti. Infine, print_graph stampa il grafico dell'andamento di S, I, R e V utilizzando le funzioni precedenti.

2.4 Il Main

L'ultimo file del programma è il main.cpp, dove vengono subito dichiarate variabili globali costanti relative le dimensioni delle finestre grafiche e del tempo di attesa che vi è tra ogni generazione. Vi è quindi una funzione di tipo Simulation::World chiamata get_parameter che funziona in modo molto simile alla sua omonima della prima parte del progetto, da cui differisce per i parametri presi in input. Nel main vero e proprio è presente un sistema di try-catch con le stesse finalità di quanto avveniva nel main della prima parte del progetto. Una volta inseriti i dati richiesti viene definita a 0 una variabile int chiamata day che tiene conto dei giorni trascorsi di simulazione, viene dichiarato un vettore wolrd_history, che conterrà tutti gli stati generati perchè di questi possa poi essere stampato il grafico e vengono create due finestre grafiche: su una delle due apparirà il grafico una volta iniziata la simulazione mentre sull'altra è visualizzata la griglia e con il click del mouse è possibile cambiare lo stato delle celle tramite le funzioni definite in plague.cpp. Questa operazione, e quanto segue da qui in poi, è gestito all'interno di un ciclo while che si interrompe quando la finestra con la griglia viene chiusa.

Con l'uso di un if viene quindi iniziata la simulazione solo se viene premuto il tasto invio: la prima cosa che avviene è la stampa dell'intestazione della tabella e dello stato al giorno zero a terminale, successivamente si aggiunge con un push_back al vettore word_history lo stato iniziale e viene creata la finestra da cui gestire le opzioni. A questo punto si apre un secondo ciclo while che continua fino a che la finestra della griglia e quella delle opzioni restano aperte. In questo ciclo avviene la simulazione vera e propria: per prima cosa un if consente il random walk se non è attivo il lockdown, successivamente il giorno viene incrementato di uno per poi passare all'uso della funzione evolve per costruire un Simulation::World chiamato next e contenente lo stato successivo appena generato; questo viene quindi mostrato nella griglia e stampato nella tabella a terminale. Si procede poi ad aggiungere tale stato al vettore world_history e si stampa o ristampa il grafico aggiornato. Infine la vecchia generazione viene sostituita con quella nuova e con l'uso delle funzioni delle librerie standard si mette in pausa il programma per consentire all'utente di percepire la variazione della simulazione in maniera graduale. A questo punto si utilizzano due cicli while per la gestione degli eventi inerenti la chiusura delle finestre e le interazioni con il mouse su di esse. Tale sistema è utilizzato pure nel ciclo più esterno per gestire la chiusura delle finestre.

2.5 Interfaccia del programma

Inizialmente è necessario inserire a terminale i parametri della simulazione (Figura 5) che devono rispettare alcune condizioni: Beta, Gamma e Theta devono essere compresi tra 0 e 1 inclusi e la dimensione del lato del mondo deve essere maggiore di zero. Una volta inseriti tutti i parametri richiesti per il funzionamento della simulazione verrà visualizzata la griglia delle persone accompagnata da una finestra che fino all'avvio della simulazione resterà vuota. Cliccando sulle singole celle è possibile cambiare il loro stato (un click fa diventare una cella da S a I, da I a R, da R a E e da E a S come descritto nella figura 6) e premendo invio si avvia la simulazione. A questo punto viene aperta una finestra con le opzioni e viene mostrato il grafico in tempo reale sulla finestra che inizialmente era vuota (Figura 7) mentre viene stampata la tabella.

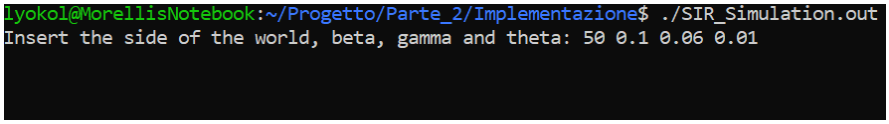

Figure 5: Input da terminale

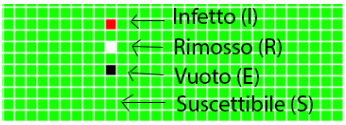

Figure 6: Interfaccia di input

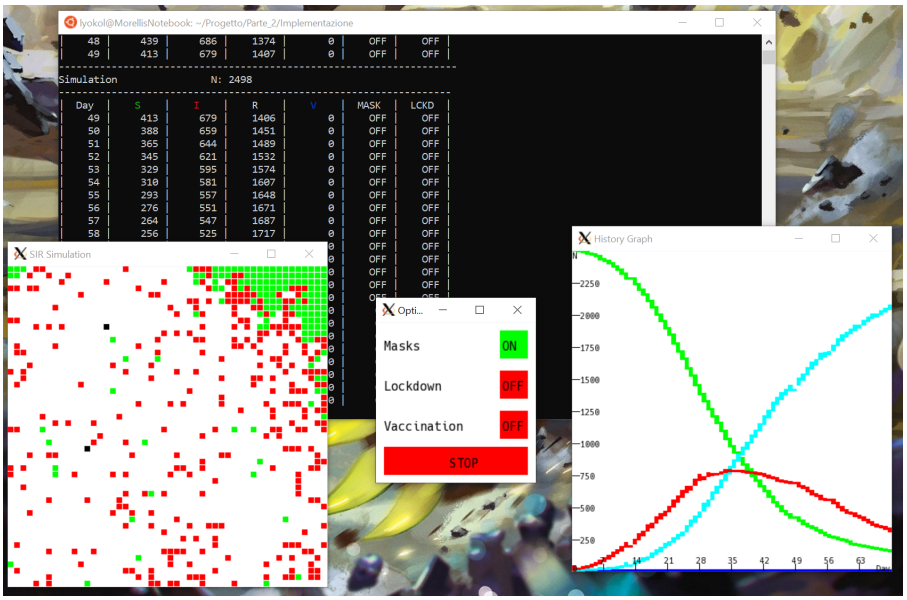

Figure 7: Esempio di output durante una simulazione

Premendo il pulsante stop la simulazione si interrompe ed è possibile modificare lo stato delle celle, in seguito premendo nuovamente invio la simulazione riprende.

2.6 Considerazioni

Il programma realizzato non presenta errori di compilazione o warning in seguito all'uso delle opzioni "-Wall -Wextra", ma presenta una serie di memory leaks imputati alla libreria SFML, esattamente come nella prima

parte del progetto (si veda il paragrafo 1.6). I dati di output sono in linea con quanto ci si potrebbe aspettare a grandi linee da una pandemia e l'andamento grafico dei paramenti S I R è simile a quanto ottenuto con il modello SIR stesso. Inoltre abbiamo osservato che le misure di contenimento effettivamente impattano sull'espansione della pandemia.

3 Istruzioni di compilazione

Per la compilazione consigliamo l'utilizzo di cmake: a tale scopo tra i file sorgente è già stato inserito il file "CMakeList.txt" che contiene le istruzioni per la compilazione, la quale avviene utilizzando le opzioni -Wall -Wextra -g e l'address sanitizer. All'interno di Parte_1 o Parte_2 si crei una directory, è consigliato chiamarla build in quanto in essa si troveranno i build files di cmake. Al suo interno si copi il file font.ttf, che si trova assieme ai file sorgente, e successivamente si lanci il comando "cmake -DCMAKE_BUILD_TYPE=Debug ..". Una volta che i build files sono stati generati sempre nella stessa cartella si lanci il comando "cmake –build .", da cui si otterrà un eseguibile di nome SIR_Simulation.out. Per lanciarlo scrivere "./SIR_Simulation.out" e premere invio. Per la compilazione dei test invece è sufficiente l'uso di g++: per la prima parte è sufficiente compilare sir.cpp e sir.test.cpp con il comando "g++ -Wall -Wextra -g -fsanitize=address -std=c++17 sir.cpp sir.test.cpp", per la seconda parte invece è necessaria la compilazione di plague.cpp e plague.test.cpp con il comando "g++ -Wall -Wextra -g -fsanitize=address -std=c++17 plague.cpp plague.test.cpp", al termine di ogni compilazione si otterrà un file a.out da lanciare con "./a.out".
Vi è la possibilità di consultare la repository utilizzata durante la stesura del codice.

4 Note per l'uso

Qualora si utilizzi un xServer è necessario disattivare il supporto a OpenGL nativo che può causare problemi con la libreria grafica SFML. Nel programma della seconda parte di progetto si è osservato che può presentarsi un dealy nella fase di input qualora si utilizzi un numero molto elevato di celle nella simulazione, tal caso è necessario pazientare per pochi attimi.