Simulazione di un epidemia in linguaggio C++

Matteo Zandi

9 luglio 2021

Contents

1 Introduzione **3**
- 1.1 Modello SIR con equazioni differenziali . . . 3
- 1.2 Modello SIR con automa cellulare . . . 3

2 Prima parte: Equazioni differenziali **4**
- 2.1 SIR.hpp e SIR.cpp . . . 4
  - 2.1.1 struct Population . . . 4
  - 2.1.2 struct Parameter . . . 4
  - 2.1.3 class Virus e funzioni . . . 4
- 2.2 SIR_output.hpp e SIR_output.cpp . . . 5
  - 2.2.1 Funzioni . . . 5
  - 2.2.2 class Graph_display . . . 6
- 2.3 SIR_input.hpp e SIR_input.cpp . . . 8
  - 2.3.1 Funzioni . . . 8
- 2.4 SIR_main.cpp . . . 8
  - 2.4.1 Controllo input . . . 8
  - 2.4.2 Case 1 . . . 8

3 Seconda parte: Automa cellulare **9**
- 3.1 SIR_automa.hpp e SIR_automa.cpp . . . 10
  - 3.1.1 struct Population . . . 10
  - 3.1.2 struct Parameter . . . 10
  - 3.1.3 enum Cell . . . 10
  - 3.1.4 struct Point . . . 10
  - 3.1.5 Classe World . . . 10
  - 3.1.6 Funzioni . . . 10
- 3.2 SIR_automa_output.hpp e SIR_automa_output.cpp . . . 12
  - 3.2.1 class Automa_display . . . 12
- 3.3 SIR_input.hpp e SIR_input.cpp . . . 13
  - 3.3.1 Funzioni . . . 13
- 3.4 SIR_main.cpp . . . 13
  - 3.4.1 Case 2 . . . 13

4 Test **13**
- 4.1 SIR_test.cpp . . . 14

5 Compilazione **14**
- 5.1 Test . . . 14
- 5.2 Programma . . . 14

6 Note **14**

1 Introduzione

Lo scopo della presente relazione è quello di mostrare le scelte di design e di implementazione adottate nel progetto. In questa sezione verranno descritti sommariamente i due modelli utilizzati: quello derivato dalla discretizzazione delle equazioni differenziali (vedi sottosezione 1.1) e quello derivato dal cosiddetto automa cellulare (vedi sottosezione 1.2), mentre nelle successive verrà analizzato il codice (vedi sezioni 2 e 3).

1.1 Modello SIR con equazioni differenziali

Il modello SIR è un'insieme di equazioni differenziali in grado di descrivere lo sviluppo di una epidemia all'interno di una popolazione chiusa e divisa in tre gruppi:

- **S** (suscettibili), ovvero le persone che possono essere infettate;
- **I** (infetti), ovvero le persone infette che possono infettare;
- **R** (rimossi), ovvero le persone che non possono più essere infettate.

Le equazioni che descrivono il passaggio da S a I e da I a R (le uniche direzioni possibili) sono:

$$\frac{dS}{dt} = -\beta \frac{S}{N} I,$$

$$\frac{dI}{dt} = \beta \frac{S}{N} I - \gamma I,$$

$$\frac{dR}{dt} = \gamma I,$$

con i vincoli:

- $N = S + I + R$, essendo la popolazione chiusa;
- $0 < \beta < 1$, che misura la probabilità di contagio tra infetti e suscettibili;
- $0 < \gamma < 1$, che misura l'inverso del tempo medio di guarigione;
- $R_0 = \frac{\beta}{\gamma} > 1$, altrimenti gli infetti guarirebbero prima di infettare e l'epidemia non partirebbe.

Discretizzando con $\Delta t = 1$, si ottiene:

$$S_i = S_{i-1} - \beta \frac{S_{i-1}}{N} I_{i-1}$$

$$I_i = I_{i-1} + \beta \frac{S_{i-1}}{N} I_{i-1} - \gamma I_{i-1}$$

$$R_i = R_{i-1} + \gamma I_{i-1}$$

Queste sono le equazioni utilizzate nella prima parte del programma (vedi sezione 2). Sono di carattere deterministico, ovvero che, dal semplice inserimento dei parametri iniziali, si può prevedere la completa evoluzione dell'epidemia.

1.2 Modello SIR con automa cellulare

Un altro modo per simulare un'epidemia è il cosiddetto automa cellulare, una griglia formata da celle che possono rappresentare una persona o uno spazio vuoto. La persona all'interno della griglia si può muovere in altre celle vuote e, nel caso un suscettibile incontri degli infetti nelle celle adiacenti, può infettarsi. Questo approccio è mostrato nella seconda parte del programma (vedi sezione 3). E' di carattere probabilistico, ovvero non si può prevedere l'evoluzione completa dall'inserimento dei parametri iniziali, essendo regolata da generatori di numeri casuali per le probabilità di contagio, guarigione e spostamento.

2 Prima parte: Equazioni differenziali

In questa sezione verrà mostrato il modo in cui è stato implementato il primo modello (file SIR.hpp e SIR.cpp, vedi sezione 2.1), come sono stati gestiti l'input e i vari output (file SIR_output.hpp e SIR_output.cpp, vedi sezione 2.2, e la prima metà dei file SIR_input.hpp e SIR_input.cpp, vedi sezione 2.3) e infine verrà presentato come le classi e le funzioni definite in questi file sono state utilizzate nel main (SIR_main.cpp, vedi sezione 2.4).
Tutto il codice relativo a questa parte è stato posto sotto il namespace epidemic_SIR.

2.1 SIR.hpp e SIR.cpp

E' stato scelto di raggruppare, in questi file, la parte di codice che regola l'evoluzione dell'epidemia: due struct (Population e Parameter), una classe (Virus) e una funzione (evolve).

2.1.1 struct Population

Population è lo struct che implementa i tre gruppi in cui la popolazione si suddivide. E' formato da 3 double (s, i, r), e non int, per mantenere in fase di calcolo tutte le cifre decimali possibili ed approssimare soltanto in fase di output (vedi funzione round_off, sottosezione 2.2.1).

2.1.2 struct Parameter

Parameter è lo struct che implementa i parametri che regolano il passaggio da un gruppo all'altro. E' formato da 2 double (β, γ).

2.1.3 class Virus e funzioni

Virus è la classe che implementa l'epidemia. E' formata da una parte privata, contenente la popolazione iniziale (m_initial_population) e un vettore di Population (m_data); e da una parte pubblica, contenente i metodi get_data e add_data.
Il vettore m_data ha la funzione di tenere traccia dei tre gruppi in tutti i momenti dell'evoluzione dell'epidemia, dove alla posizione i-esima corrisponde il Population relativo al giorno i-esimo. Il costruttore aggiunge la popolazione iniziale nel primo posto del vettore, corrispondente al giorno 0, mentre per le posizioni successive si è scelto di usare una funzione fuori dalla classe (evolve), modificando o accedendo alle parti private attraverso i suoi metodi (get_data e update_data).

- Virus::**get_data**, quando chiamato, restituisce il vettore m_data.
- Virus::**update_data**, quando chiamato, aggiunge il Population data_to_add (argomento della funzione preso in sola lettura) in fondo al vettore, con il metodo .push_back().
- **evolve**, quando chiamato, permette all'epidemia di progredire per duration giorni, regolata dai parametri Parameter.

  Copia in virus la classe virus_to_evolve. Copia in data il vettore di Population contenuto in virus, con il metodo .get_data(). Copia il contenuto dell'ultima posizione del vettore in population, tramite il metodo .back. Inizializza il vincolo N, come costante, prendendo come riferimento la posizione 0. Itera un ciclo for per duration volte (dove si utilizzano le equazioni differenziali in forma discreta per trovare il Population del giorno successivo attraverso quello del giorno precedente) in cui: definisce delta_i e delta_r, ovvero rispettivamente quanti suscettibili diventano infetti ($\Delta I = \beta \frac{S_{i-1}}{N} I_{i-1}$) e quanti infetti diventano rimossi ($\Delta R = \gamma I_{i-1}$); sottrae delta_i da s e lo aggiunge a i ($S_i = S_{i-1} - \Delta I$ e $I_i = I_{i-1} + \Delta I$); sottrae delta_r da i e lo aggiunge a r ($I_i = I_{i-1} - \Delta R$ e $R_i = R_{i-1} + \Delta R$); aggiorna il vettore m_data della classe virus con il metodo update_data e, una volta uscito dal ciclo, restituisce virus, con il suo vettore aggiornato.

  Sono presenti degli assert in cui: si controlla che il vettore data non sia vuoto (almeno la popolazione iniziale alla posizione 0 è presente), con il metodo .empty; si controlla che delta_i e delta_r non siano negative (da S a I e da I a R sono le uniche direzione possibili); si controlla che i gruppi s, i ed r non siano negativi (non ha senso che un gruppo sia negativo); si controlla che il vincolo N sia rispettato, con std::round (per una maggior precisione nel confrontare una somma di double con un int).

Si è scelto di usare una funzione fuori dalla classe per avere possibilità di simulare più giorni aggiuntivi, partendo non solo dal giorno 0 ma anche da un certo giorno in poi, una volta simulata l'epidemia fino a lì (per esempio chiamando due volte virus = evolve(virus, parameter, 10) si simulano 20 giorni).

2.2 SIR_output.hpp e SIR_output.cpp

E' stato scelto di raggruppare, in questi file, la parte di codice che regola i vari tipi di output: tre funzioni (round_off, print e print_round_off) e una classe (Graph_display).

2.2.1 Funzioni

- **round_off**, quando chiamato, approssima un vettore di Population, formato dai gruppi con numeri decimali, in numeri interi.

  Copia in data il vettore data_to_round_off. Definisce la lunghezza del vettore, con il metodo .size e lo static_cast<int>. Definisce il vincolo N, come costante, prendendo come riferimento la posizione 0. Itera un ciclo for per tutta la lunghezza del vettore in cui: definisce population come il Population presente alla posizione i-esima; calcola la parte intera (s_int, i_int, r_int) e la parte frazionaria (s_fra, i_fra, r_fra) dei tre gruppi, approssimando per difetto (definizione di parte intera) con la funzione std::floor e poi sottraendo al numero decimale la parte intera (definizione di parte frazionaria); sostituisce al valore di population le rispettivi parti intere e poi calcola la somma delle parti frazionarie sum, ovvero quanto manca per arrivare a N sommando tutte le parti intere, usando std::round per una maggior precisione nell'approssimazione; itera un ciclo while per tante volte quanti sum mancano a N, in cui: se la parte frazionaria di s è minore di 0.5, la si aggiunge a i e poi la si pone uguale a zero mentre se invece è maggiore di 0.5, la si pone uguale a 1 e si sottrae alla parte frazionaria di i il valore che aggiungo a s_fra per arrivare a 1; ripete lo stesso procedimento sostituendo i_fra ad s_fra e r_fra ad i_fra; diminuisce sum di 1; una volta usciti dal while (ovvero quando non ci mancheranno più numeri per raggiungere N). le parti frazionarie saranno o 1 o 0, quindi le somma ai rispettivi gruppi della popolazione; sostituisce alla posizione i-esima il population approssimato. Una volta uscito dal ciclo for, restituisce il vettore data, con tutti i Population in tutte le posizioni approssimati.

  Sono presenti degli assert in cui: si controlla che i gruppi s, i ed r non siano negativi; si controlla che il vincolo N sia rispettato, con std::round (per una maggior precisione nel confrontare una somma di double con un int); si controlla che le direzioni di passaggio siano rispettate, ponendo S precedente sempre maggiore o uguale a S successivo e R precedente sempre minore o uguale a R successivo.

  Si è scelto di mantenere i gruppi in double in fase di calcolo per poi approssimarli solo in fase di output, perché se si fosse operato con soli numeri naturali, ci sarebbe voluto un R_0 molto più grande di 1 per far partire l'epidemia, volendoci almeno un delta_i ed un delta_r maggiori di 0.5 (arrotondando con std::round) per far passare una persona da S a I o da I a R. Per esempio con S=999, I=1, R=0, $\beta = 0.4$ e $\gamma = 0.3$, anche se R_0= 1.33 e quindi maggiore di 0, l'epidemia non partirebbe mai essendo al primo giorno S = 998.6 $\approx$ 999, I = 1.09 $\approx$ 1 e R = 0.3 $\approx$ 0, quindi ci sarebbe sempre lo stesso S, I ed R per tutti i giorni e l'epidemia non partirebbe, mentre invece mantenendo i numeri decimali e poi approssimando alla fine, l'epidemia parte. Si è inoltre scelto di far partire il for da 1, perché nella prima posizione ci sarà sempre la popolazione iniziale, già approssimata, e anche perché così è possibile, nell'assert sulle direzioni, confrontare sempre la posizione precedente con quella successiva, altrimenti si avrebbero dei problemi con la posizione 0, essendo la posizione -1 non definita.

- **print**, quando chiamato, stampa l'R_0 e la tabella rispettivamente con i valori decimali.

  Inizializza la larghezza width di una colonna. Inizializza la lunghezza del vettore, con il metodo .size e lo static_cast<int>. Stampa l'R_0, l'intestazione della tabella (Days, S, I, R, N) e, con un ciclo for per tutta la lunghezza del vettore, stampa il giorno come indice i, i valori dei gruppi i-esimi e la somma dei gruppi corrispondente al vincolo N, con il std::cout e l'opzione aggiuntiva std::set(width), che permettere di allargare le colonne della tabella, per non dover andare a capo.

- **print_round_off**, quando chiamato, prende come argomento il vettore di Population (data_to_round_off) e i parametri in parameter, stampa l'R_0 e la tabella rispettivamente con i valori interi approssimati.

E' simile a print, con l'unica differenza che copia il vettore data_to_round_off nel vettore data, utilizzando la funzione round_off per approssimare i valori.

2.2.2 class Graph_display

Graph_display è la classe che implementa la finestra grafica in cui disegnare i grafici. E' formata da una parte privata, contenente la finestra grafica della libreria SFML (m_window), il font (m_font) e la dimensione del lato della finestra (m_display_side); e da una parte pubblica, contenente i metodi count_digits, draw_axes, draw_legend, transform, draw_susceptible, draw_infectious e draw_recovered.
Il costruttore controlla che il font contenuto nel file Roboto.ttf sia disponibile, altrimenti lancia una exception, oltre a definire m_window e m_display_size, presi come argomenti. Nei metodi sono presenti funzioni provenienti dalla libreria SFML (inclusa con <SFML/Graphics.hpp>), che permettono di disegnare le linee e i caratteri per il grafico. count_digit e transform sono due funzioni create ad hoc per, rispettivamente, contare le cifre di un numero e capovolgere il grafico rispetto all'asse delle x.

- Graph_display::**count_digit**, quando chiamato, conta le cifre di un numero.

  Inizializza il contatore result a 0. Itera un ciclo while, contando quante volte si può dividere per 10, finché il numero non è uguale a zero. Restituisce il contatore result.

- Graph_display::**draw_axes**, quando chiamato, disegna le linee, le tacche, i valori e i nomi degli assi del grafico sulla finestra grafica.

  Inizializza un VertexArray di 2 punti x_axis, del tipo linee (ovvero che, dati due punti sullo schermo, ne traccia una linea tra i due). Ad ogni punto viene data una posizione ed un colore tramite .position e .color. Disegna la linea sulla finestra con il metodo .draw. Ripete il procedimento anche per y_axis, con l'unica differenza nella scelta della posizione dei due punti.

  Si è scelto di definire la posizione dei punti in base alle percentuali di distanza dal punto in alto a sinistra della finestra (per esempio x_axis[0].position = sf::Vector2f(m_display_side * 0.15, m_display_side * 0.9); vuol dire che si trova al 15% del lato della finestra grafica di distanza dal lato sinistro e 90% del lato della finestra grafica di distanza dal lato alto).

  Inizializza la metà della distanza tra le tacche, ovvero la lunghezza dell'asse (m_display_size * 0.7) diviso 20. Inizializza un array di 22 punti x_axis_notches, del tipo linee. Itera un ciclo for per tutta la lunghezza dell'array, con passo 2 invece che 1, in cui: ad ogni iterazione fornisce posizione e colore alle tacche; ad ogni punto i-esimo (punto in alto della tacca) viene data una posizione standard, mentre ad ogni punto i+1-esimo (punto in basso della tacca) viene data una posizione in base all'indice (per quelli pari la tacca sarà più lunga, infatti (i+1)/2 determina il numero dell'indice e %2 = 0 controlla se è pari); colora ogni linea con .color e uscito dal ciclo, stampa il vettore con il metodo .draw. Ripete il procedimento anche per y_axis_notches, con la differenza nella scelta della posizione dei due punti, in particolare tutte le tacche della stessa lunghezza;

  Si è scelto di dividere per 20 per il fatto che il for successivo aumenta di 2 in 2, quindi in realtà ogni ciclo va moltiplicato per 2, ritrovando le dieci tacche. Così facendo sarà possibile disegnare i valori adiacenti senza sovrapposizione9, mentre per l'asse y si da un definito valori. Infine si colorano e poi si stampano sulla finestra. Le posizioni sono dipendenti dall'indice del ciclo for, per permettere di disegnare per ogni iterazione del ciclo, invece che mettere a mano dieci valori. questo approccio è valido per tutte le iterazioni for presenti nelle funzioni membro della classe Display_graph. Nello stampare il vettore con più di due punti, collega le linee due a due, ovvero 0 e 1, 2 e 3, ecc.

  Inizializza il nome degli assi x_axis_name come tipo sf::Text. Fornisce il font m_font con setFont. Fornisce la stringa da stampare con setString. Fornisce la dimensione dei caratteri con SetCharacterSize. Colora con SetFillColor. Fornisce la posizione con SetPosition. Ripete il procedimento anche per y_axis_name, con l'unica differenza nella scelta della posizione del text.

  Si è scelto di usare metodi diversi perché precedentemente era stato preferito un array, mentre per stampare dei caratteri è stato usato il singolo text.

Inizializza la distanza tra le tacche, ovvero la lunghezza dell'asse (m_display_size * 0.7) diviso 10. Inizializza il vincolo N. Inizializza il numero di giorni, ovvero la lunghezza dell'array data - 1, togliendo il giorno 0. Inizializza la distanza prima di N e poi di days, tra una tacca e l'altra, ovvero dividendo la quantità per 10. Itera un ciclo for per tutte e 10 le tacche, con passo 1, in cui: inizializza un sf::Text; fornisce il font m_font con setFont; inizializza attraverso count_digits, il numero di cifre del giorno corrispondente alla tacca (ovvero moltiplicando days_range per i + 1); dichiara due std::string: la prima contenente tutte le cifre del giorno corrispondente alla tacca convertite in caratteri tramite std::to_string, mentre la seconda vuota (da riempire con solo quelle da stampare); sceglie due diversi percorsi, in base a quanto è grande days: se days è più piccolo di 1000, itera con un for quante volte sono il numero di cifre + 4 (per avere qualche decimale), in cui prende l'i-esimo carattere a partire dal primo e lo pone in fondo alla stringa approssimata (in questo modo dovrò stampare soltanto i primi caratteri, senza avere troppe cifre dopo la virgola); se invece days è più grande di 1000, potrei avere dei problemi di spazio, quindi si converte in notazione esponenziale, prendendo il primo carattere, aggiungendolo alla stringa approssimata, seguito da un punto e dal secondo carattere, poi si aggiunge *10, seguito dal numero di cifre - 1 (togliendo la cifra significativa); fornisce la stringa da stampare approssimata con setString; fornisce la dimensione dei caratteri con SetCharacterSize; colora con SetFillColor; fornisce la posizione con SetPosition, sempre con le stesse modalità (percentuali di m_display_side) e con la stessa differenza delle tacche di days, ovvero quelli pari un po' più in basso essendo più lunga la tacca; stampa il singolo text, prima che finisca il ciclo. Ripete il procedimento, sostituendo però N a days, e quindi cambiando anche le posizione, essendo sull'asse y, con la lunghezza delle tacche tutte uguali.

- **draw_legend**, quando chiamato, disegna la legenda del grafico, relativa all'ultimo giorno simulato.

  Inizializza un array di 5 vertici, del tipo linestrip (per formare un rettangolo). Fornisce la posizione con .position ai 5 vertici, dove l'ultimo coincide con il primo per avere una linea chiusa. Colora con .color i vertici. Stampa con .draw il rettangolo. Trova l'indice dell'ultima posizione del vettore, con .size e lo static_cast<int> (- 1, per togliere il giorno 0). Dichiara il sf::Text susceptible da stampare. Fornisce il font m_font con .setFont. Inizializza una stringa s in cui pone tra parentesi tonde il valore del gruppo corrispondente all'ultima posizione del vettore. Fornisce la stringa da stampare con setString. Fornisce la dimensione dei caratteri con SetCharacterSize. Colora con SetFillColor. Fornisce la posizione con SetPosition, sempre con le stesse modalità (percentuali di m_display_side). Stampa il testo della legenda con .draw. Inizializza un array di 2 vertici, del tipo lines. Colora con .color. Fornisce la posizione con .position. Stampa la linea colorata della legenda con .draw. Ripete il procedimento per gli altri gruppi, cambiando il double del Population (i e r), mentre per days non stampa la linea colorata.

- **transform**, quando chiamato, permette di capovolgere il grafico sull'asse delle x, mettendo l'origine in basso a sinistra.

  Dichiara un tipo di SFML sf::tranformable. Cambia l'origine del sistema di riferimento per definire la posizione, in (0, m_display_side), con setOrigin, ovvero in basso a sinistra. Capovolge rispetto all'asse delle x mantenendo invariato il resto, scalando di (1, -1), con setScale. Restituisce il tranform del transformable, con il getTranform.

- **draw_susceptible**,**draw_infectious** e **draw_recovered**, quando chiamate, disegnano il grafico effettivo del rispettivo gruppo.

  Dichiarano un array di tanti vertici quanti sono è lungo il vettore data, del tipo linestrip (ovvero che si collegano uno dopo l'altro creando un linea continua). Inizializzano il vincolo N e la lunghezza del vettore data, con .size e lo static_cast<int>. Inizializzano points_distance, la distanza tra due punti sull'asse x, come la lunghezza dell'asse diviso la lunghezza del vettore - 1 (togliendo il giorno 0). Iterano un ciclo for per la lunghezza di tutto il vettore data, in cui: forniscono la posizione con .position di ogni singolo punto, mettendo in x il valore progressivo del punto (ovvero la distanza tra un punto e l'altro moltiplicata per l'indice) e in y il valore corrispondente al gruppo in quel indice diviso N (per scalare il grafico dentro all'asse y) per la lunghezza dell'asse; colorano con .color. Stampano l'array con l'opzione trasform, per non averlo capovolto.

  Si è scelto di dividere in tre funzioni, perché così è possibile stampare separatamente i grafici.

2.3 SIR_input.hpp e SIR_input.cpp

E' stato scelto di raggruppare, nella prima metà di questi file, la parte di codice che implementa l'input relativo al primo modello: quattro funzioni (input_choice, input_days, input_initial_population e input_parameter).

2.3.1 Funzioni

- **input choice**, quando chiamato, regola l'input delle scelte proposte all'interno del main.

  Dichiara un double choice, la variabile che conterrà la scelta. Fornisce un valore con il std::cin (incluso nella libreria <iostream>). Controlla che questo valore sia accettabile (lanciando un'exception in caso contrario). Controlla che il std::cin.fail() non sia true, ovvero che sia stato inserito un numero nell'ultimo std::cin. Controlla che questo numero sia compreso tra il range di valori accettati (inf e sup, argomenti della funzione). Controlla che questo numero sia un intero, verificando che il numero sottratto della sua parte intera non sia diverso da zero (ovvero che abbia parte frazionaria nulla). Restituisce choice, convertito in int con static_cast<int>.

- **input days**, quando chiamato, regola l'input dei giorni da simulare. E' molto simile a input_choice, con la sola differenza che non ha argomenti perchè i giorni devono essere semplicemente non negativi.

- **input_population**, quando chiamato, regola l'input dei tre gruppi in cui la popolazione iniziale è suddivisa. Dichiara un Population initial_population. Fonisce un valore con il std::cin ad un gruppo. Controlla che questo valore sia accettabile (lanciando un'exception in caso contrario). Controlla che il std::cin.fail() non sia true, ovvero che sia stato inserito un numero nell'ultimo std::cin; Controlla che questo numero sia positivo (si noti che il valore di i, deve essere almeno 1, altrimenti senza neanche un infetto, l'epidemia non può iniziare). Controlla che questo numero sia un intero, verificando che il numero sottratto della sua parte intera non sia diverso da zero (ovvero che abbia parte frazionaria nulla). Restituisce initial_population.

- **input_parameter**, quando chiamato, regola l'input dei giorni dei parametri della simulazione. E' molto simile a input_choice, con le seguenti differenze: dichiara un Parameter al posto di un Population; controlla i range entro i quali i parametri devono stare (β e γ devono essere compresi tra 0 e 1); controlla che l'$R_0 = \frac{\beta}{\gamma}$ sia essere maggiore di 1 (altrimenti non parte l'epidemia).

2.4 SIR_main.cpp

E' stato scelto di inserire, in questo file, la parte di codice che implementa il main: la sola funzione int main().
All'inizio del file main.cpp sono inclusi gli header file necessari: SIR.hpp, SIR_automa.hpp, SIR_automa_output.hpp, SIR_input.hpp e SIR _output.hpp.

2.4.1 Controllo input

Per verificare il corretto inserimento degli input, è stato scelto il meccanismo try/catch exception: all'inizio dell'int main() è stato inserito un try e, nell'eventualità che venga lanciata un'exception, alla fine sono stati inseriti due catch: il primo nel caso sia conosciuta (descritta dal metodo .what), mentre il secondo nel caso non lo sia. L'output è del tipo std::cerr ed è indicato il failure del programma con il return EXIT_FAILURE.

Per suddividere i due diversi modi di implementare l'epidemia, è stato inserito uno switch. La scelta su quale modello utilizzare è conservata nella variabile model_choice, presa in input con la funzione input_choice, mentre le opzioni disponibili sono visualizzate in output con std::cout (1 per le equazioni differenziali e 2 per l'automa cellulare).
Nella prossima sezione verrà trattato il case 1 (per il case 2, vedi sezione 3.3).

2.4.2 Case 1

Il case 1 richiede in input la popolazione iniziale (initial_population), con la funzione input_initial_population, e i parametri (parameter), con la funzione input_parameters. Inizializza la classe Virus epidemic, con la popolazione iniziale come argomento. Attraverso 2 do-while, permette all'utente di simulare ulteriori giorni ad epidemia già in corso o di apprezzare diversi output della stessa epidemia.
Nel primo do-while, la scelta di simulare nuovi giorni è conservata nella variabile simulate_more_days_choice,

presa in input con la funzione input_choice, mentre le opzioni disponibili sono visualizzate in output con std::cout (1 per simulare nuovi giorni e 0 per uscire). E' effettuata alla fine del do, prima che il while ne controlli il valore. All'inizio del do, invece, chiede in input il numero di giorni (days), con la funzione input_days; permette all'epidemia di progredire, con la funzione evolve; copia nel vettore data i risultati, con il metodo get_data.
Nel secondo do-while, la scelta su quale output utilizzare è conservata nella variabile output_choice, presa in input con la funzione input_choice, mentre le opzioni disponibili sono visualizzate in output con std::cout (1 per la tabella con i valori decimali, 2 per la tabella con i valori interi approssimati, 3 per i vari tipi di grafico e 0 per uscire). Il meccanismo è lo stesso del precedente, tuttavia è effettuata all'inizio del do, in modo da usare la variabile output_choice anche per uno switch, in cui associare ad ogni case un diverso tipo di output.
Il case 0 non fa nulla, perché semplicemente permetterà di uscire dal ciclo.
Il case 1 stampa i valori della tabella con cifre decimali, usando la funzione print.
Il case 2 stampa i valori della tabella con cifre intere, approssimando una reale epidemia con gruppi di persone descritti da numeri naturali, usando la funzione print_round_off.
Il case 3 utilizza la libreria SFML per disegnare i grafici sulla finestra grafica. Inizializza come costante la variabile display_side, contenente la lunghezza del lato della finestra. La scelta su quale grafico utilizzare è conservata nella variabile graph_choice, presa in input con la funzione input_choice, mentre le opzioni disponibili sono visualizzate in output con std::cout (1 per il grafico con soli suscettibili, 2 per il grafico con soli infetti, 3 per il grafico con soli rimossi, 4 per il grafico con suscettibili e infetti, 5 per il grafico con infetti e rimossi, 6 per il grafico con suscettibili e rimossi e 7 per il grafico con tutti e tre i gruppi). Inoltre la scelta su quale tipologia di grafico utilizzare è conservata nella variabile kind_of_graph_choice, presa in input con la funzione input_choice, mentre le opzioni disponibili sono visualizzate in output con std::cout (1 per il grafico dinamico, ovvero vedere gradualmente tutta l'evoluzione dell'epidemia, 2 per il grafico statico, ovvero vedere direttamente tutta l'evoluzione dell'epidemia). Quest'ultima, è la variabile di uno switch che divide le tue tipologie di grafico, mentre graph_choice servirà più avanti per scegliere quale grafico stampare.
Il case 1 introduce la possibilità di decidere la velocità con cui disegnare il grafico, conservando la scelta nella variabile framerate_choice, presa in input con la funzione input_choice, e visualizzando le opzioni disponibili in output con std::cout (1. per vedere 1 fotogramma al secondo, 2. per vedere 5 fotogrammi al secondo, 3. per vedere 10 fotogrammi al secondo e 4. per vedere 20 fotogrammi al secondo). Utilizza uno switch per cambiare il valore della variabile framerate, inizializzata a 0, in base alla scelta contenuta in framerate_choice. Inizializza la finestra grafica window, con la grandezza della finestra (quadrata di lato display_side) e il nome come argomenti. Imposta il limite massimo di fotogrammi al secondo con il metodo .setFramerateLimit. Inizializza la classe Graph_display, con la finestra grafica window e la sua lunghezza del lato display_side. Inizializza il contatore di giorni, duration, a 0 e la variabile booleana, running, come true. Inizializza il vettore di Population temporary_data, che terrà in memoria tutte le posizioni del vettore data (contenente tutta l'epidemia) fino alla posizione duration. Itera un ciclo while (finché running non diventerà false, ovvero nel caso in cui ci sia l'evento di chiusura della finestra sf::Event::Closed descritto in fondo), in cui: colora lo sfondo di nero, con il metodo .clear; inizializza la lunghezza data_size del vettore data, con il metodo .size e lo static_cast<int>; nel caso duration sia minore della lunghezza del vettore data, aggiunge, in fondo al temporary_data, il Population corrispondente alla posizione duration, con il metodo .push_back, in modo da poter stampare tutta l'epidemia fino a quel momento lì, e aumenta il contatore duration; stampa gli assi e la legenda di temporary_data, con i metodi draw_axes e draw_legend; stampa la linea del grafico, con i metodi .draw_susceptible, .draw_infetious e .draw_recovered, opportunamente chiamati in base alla scelta di graph_choice e implementati nei 3 if; mostra la finestra grafica con .display e, una volta usciti dal ciclo, chiude la finestra con .close.
Il case 2 è molto simile al case 1, tranne il fatto che non sono presenti né la scelta dei fotogrammi al secondo e né il ciclo while, perché stampa direttamente il vettore data, già arrivato alla fine della simulazione, nella stessa modalità di temporary_data. Chiude sempre la finestra sempre con sf::Event::Closed e .close.

3 Seconda parte: Automa cellulare

In questa sezione verrà mostrato il modo in cui è stato implementato il secondo modello (file SIR_automa.hpp e SIR_automa.cpp, vedi sezione 3.1), come sono stati gestiti l'input e i vari output (file SIR_automa_output.hpp e SIR_automa_output.cpp, vedi sezione 3.2, e seconda metà dei file SIR_input.hpp e SIR_input.cpp, vedi sezione 3.3) e infine verrà presentato come le classi e le funzioni definite in questi file sono state utilizzate nel main (SIR_main.cpp, vedi sezione 3.4).

Tutto il codice relativo a questa parte è stato posto sotto il namespace epidemic_SIR_CA.

3.1 SIR_automa.hpp e SIR_automa.cpp

E' stato scelto di raggruppare, in questi file, la parte di codice che regola l'evoluzione dell'epidemia: tre struct (Population, Parameter, Point), una classe (World) e un'enum (Cell).

3.1.1 struct Population

Population è lo stesso presente nel namespace epidemic_SIR.

3.1.2 struct Parameter

Parameter è lo stesso presente nel namespace epidemic_SIR, con l'aggiunta di un terzo double (α), che regola la probabilità di spostamento delle persone nella griglia.

3.1.3 enum Cell

Cell è l'enum che definisce come una cella può essere all'interno della griglia. E' formato da 4 char (Empty, Susceptible, Infectious, Recovered).

3.1.4 struct Point

Point è lo struct che definisce le coordinate di un punto sulla griglia, ovvero in che riga e in che colonna si trova. E' formato da due int (row e column).

3.1.5 Classe World

World è la classe che definisce il mondo in cui l'epidemia si evolve. E' formata da una parte privata, contenente la dimensione del lato della griglia (m_side), un vettore di Cell (m_grid) e un vettore di Population (m_data); e da una parte pubblica, contenente i metodi get_side, get_grid, get_data, add_grid e update_data. Il vettore di Population ha la funzione di tenere traccia di quante persone sono presenti nei tre gruppi, mentre il vettore di Cell ha la funzione di tenere traccia della posizione di ogni cella sulla griglia. Il costruttore prende come argomento il lato della griglia e lo copia in m_side, poi crea m_grid di lunghezza m_side* m_side con in tutte le posizioni un Cell::Empty e infine crea m_data vuoto. Per far evolvere la griglia, si è scelto di usare funzioni fuori dalla classe (find_index, random_int_generator, probability, generate, update_data, find_direction, move, neighbours, evolve_grid e evolve) mentre per tenere traccia dei cambiamenti nei due vettori contenuti nella classe si usano i suoi metodi (get_side, get_grid, get_data, add_grid e update_data).

- World::**get_side**, quando chiamato, restituisce il lato della griglia m_side.
- World::**get_grid**, quando chiamato, restituisce il vettore di Cell, contenente la griglia, m_grid.;
- World::**get_data**, quando chiamato, restituisce il vettore di population, contenente in memoria i tre gruppi per tutto l'arco dell'epidemia, m_data.;
- World::**add_grid**, quando chiamato, sostituisce il vettore di Cell aggiornato grid_to_add (argomento della funzione preso in sola lettura) a quello precedente m_grid
- World::**add_data**, quando chiamato, aggiunge il Population data_to_add (argomento della funzione preso in sola lettura) in fondo al vettore, con il metodo .push_back().

3.1.6 Funzioni

- **find_index**, quando chiamato, restituisce l'indice relativo al vettore griglia in cui si trova il punto, date le coordinate contenute in un Point e la lunghezza del lato della griglia.

  Inizializza come costanti i (j), il resto tra, la somma della riga (colonna) più la lunghezza del lato, e la lunghezza del lato (in condizioni normali corrispondono alla riga e alla colonna, mentre invece se il punto è fuori dalla griglia per esempio a destra con row > side, sarà possibile creare, in stile pac-man, un mondo dove il punto ritornerà fuori a sinistra). Controlla, attraverso un assert, che i e j

siano maggiori di zero e che siano minori di side. Inizializza index, ovvero l'indice in cui si troverà il punto nel vettore e non in coordinate, come la somma tra il prodotto di i per la lunghezza del lato e j. Controlla, attraverso un assert, che l'indice sia maggiore di zero e interno alla griglia, ovvero minore di side * side. Restituisce l'indice.

- **random_int_generator**, quando chiamato, premette di generare un numero casuale del tipo int, tra due valori.

  Inizializza il motore di generazione std::default_random_engine enginestd::random$_device()(inclusonellalibreria <$ $random >).Inizializzaladistribuzioneuniformediinteristd :: uniform\_int\_distribution < int >, nell'intervallotrai$

- **probability**, quando chiamato, restituisce un booleano (true o false) se il numero generato è all'interno dell'intervallo generato dalla probabilità o no.

  Inizializza il motore di generazione std::default_random_engine enginestd::random$_device()(inclusonellalibreria <$ $random >).Inizializzaladistribuzioneuniformedidoublestd :: uniform\_real\_distribution < double >$ $, nell'intervallotra0e1.Inizializzaunnumerodoubleconilgeneratorecasuale.Restituiscetrueseilnumeroèminoreoalpi$

- **update_data**, quando chiamato, conta quanti sono i tre gruppi in quel dato momento sulla griglia e li tiene in mememoria sul vettore di Population.

  Copia in world la classe world_to_generate. Copia in grid il vettore di Cell contenuto in world, con il metodo .get_grid(). Copia in data il vettore di Population contenuto in world, con il metodo .get_data(). Copia in side la lunghezza del lato della griglia contenuta in world, con il metodo .get_side. Inizializza i contatori dei tre gruppi (susceptible_count, infectious_count e recovered_count) uguali a zero. Itera due for, scorrendo tutta la griglia, in cui: inizializza un Point in cui utilizza la row e la column corrispondente agli indici dei for; trova l'indice index del punto con la funzione find_index; se la cella corrispondente all'indice è del tipo Susceptible, aumenta il contatore susceptible_count, se è del tipo Infectious, aumenta il contatore infectious_count oppure se è del tipo Recovered, aumenta il contatore recovered_count. Una volta uscito dal ciclo, aggiunge in fondo al vettore population, definito dai vari contatori dei rispettivi gruppi, con il metodo add_data. Restituisce la classe world.

- **generate**, quando chiamato, trasforma casualmente tante celle sulla griglia quante sono number_of_people da Empty al tipo cell_type.

  Copia in world la classe world_to_generate. Copia in grid il vettore di Cell contenuto in world, con il metodo .get_grid(). Copia in data il vettore di Population contenuto in world, con il metodo .get_data(). Copia in side la lunghezza del lato della griglia contenuta in world, con il metodo .get_side. Itera un for per tante volte quanti sono number_of_people, in cui: genera casualmente un punto sulla griglia, definendo un point standard e trovando due numeri da 0 a side - 1 (che corrispondono alla riga e alla colonna del punto), con la funzione random_int_generator; trova l'indice del punto, con la funzione find_index; se la cella corrispondente è vuota, allora sostituisce il cell_type preso come argomento al suo posto, altrimenti diminuisce l'indice del for di uno, così da poter ritentare un'altra volta. Una volta uscito dal ciclo, aggiorna la nuova griglia della classe world con il metodo add_grid. Restituisce la classe world;

- **find_direction**, quando chiamato, trova una delle 8 direzione possibili in cui il punto può muoversi.

  Genera due numeri causali interi tra 1 e -1, ovvero le possibili direzioni che può compiere (essendo uniformemente distribuita, è possibile anche che non si muova, se escono 0 e 0). Copia in point il punto dove si trova la persona point_to_move. Aggiunge la direzione al point, ovvero di quanto si deve spostare dal punto precedentemente, usando i numeri generati casualmente. Restituisce le coordinate nuove contenute nel punto (è possibile che siano anche negative, ma ciò verrà aggiustato quando verrà chiamata la funzione find_index).

- **move**, quando chiamato, permette di far muovere le persone nella griglia.

  Copia in world la classe world_to_generate. Copia in grid il vettore di Cell contenuto in world, con il metodo .get_grid(). Copia in side la lunghezza del lato della griglia contenuta in world, con il metodo .get_side. Itera con due for, scorrendo tutta la griglia, in cui: inizializza un Point in cui utilizza la row e la column corrispondente agli indici dei for; trova l'indice index del punto con la funzione find_index; se la cella corrispondente all'indice è vuota e la funzione probability, utilizzando come argomento il travel probabilty, fornisce un true (tramite i generatore casuale), si troverà il punto in cui la persona

dovrebbe muoversi, con il corrispondente indice, e soltanto se la cella in cui dovrebbe muoversi è vuota, si sostituisce la cella del nuovo punto con la persona e invece quella vecchia la si pone vuota. Una volta usciti dal ciclo, aggiorna la nuova griglia della classe world con il metodo add_grid. Restituisce la classe world.

- **neighbours**, quando chiamato, conta quante sono le celle adiacenti a point del tipo cell_type.

  Copia in world la classe world_to_generate. Copia in grid il vettore di Cell contenuto in world, con il metodo .get_grid(). Copia in side la lunghezza del lato della griglia contenuta in world, con il metodo .get_side. Inizializza il contatore count a 0. Trova l'indice del Point. Se il punto desiderato è del tipo di cui si sta cercando di contare, verrà diminuito di 1, per non contarlo successivamente. Itera due for, scorrendo per tutti i punti adiacenti a point, in cui: trova l'indice index del punto sommando i due indici i e j al point; se in quella cella di quell'indice è del tipo cercato cell_type, aumenta il contatore di 1. Una volta uscito dal ciclo, restituisce il contatore count.

- **evolve_grid**, quando chiamato, permette di far evolvere l'epidemia, trasformando i suscettibili in infetti e gli infetti in rimossi.

  Copia in world la classe world_to_evolve. Copia in grid il vettore di Cell contenuto in world, con il metodo .get_grid(). Copia in side la lunghezza del lato della griglia contenuta in world, con il metodo .get_side. Itera due for, scorrendo tutta la griglia, in cui: inizializza un Point in cui utilizza la row e la column corrispondente agli indici dei for; trova l'indice index del punto con la funzione find_index; se la cella corrispondente all'indice è Susceptible, si contano, nella costante c, quanti infetti ci sono adiacenti, con la funzione neighbours, e, se c è positivo e la probability, con la probabilità beta moltiplicata per ogni infetto vicino, è true, il suscettibile diventerà infetto; se la cella corrispondente all'indice è Infectious e la probability, con la probabilità gamma, è true, l'infetto diventerà rimosso. Una volta uscito dal ciclo, aggiorna la nuova griglia della classe world con il metodo add_grid. Restituisce la classe world;

- **evolve**, quando chiamato, mette insieme le due funzioni move e evolve_grid, per fornire il nuovo mondo con quelle due implementazioni.

  Copia in world la classe world_to_evolve. Muove le persone nelle celle con la funzione move, usando il parametro α contenuto in parameter. Trasforma i suscettibili in infetti e gli infetti in rimossi, con i parametri contenuti in parameter. Restituisce la classe world.

3.2 SIR_automa_output.hpp e SIR_automa_output.cpp

3.2.1 class Automa_display

Automa_graph è la classe che implementa la finestra grafica su cui disegnare la griglia e i grafici. E' formata da una parte privata, contenente la finestra grafica della libreria SFML (m_window), il font (m_font), il vettore di Population (m_data) e la dimensione del lato della finestra (m_display_side); e da una parte pubblica, contenente i metodi draw_grid, count_digits, draw_axes, draw_legend, transform, draw_graph. Il costruttore controlla che il font contenuto nel file Roboto.ttf sia disponibile, altrimenti lancia una exception, oltre a definire m_window e m_display_size, presi come argomenti. I metodi draw_axes, count_digits, draw_legend, transform, draw_graph sono identici alla classe Graph_display del namespace epidemic_SIR, con l'eccezione di aver creato una finestra grafica rettangolare, lunga due volte m_display_side, con nella metà sinistra la griglia e a destra il grafico (quindi i valori relativi alle ascisse delle posizioni dei punti del grafico aumentati di m_display_side). Il metodo draw_graph è l'unione di draw_susceptible, draw_infectious e draw_recovered. E' presente un separatore tra le due metà, creato con l'array di due vertici, del tipo lines, inserito in draw_legend. L'unico nuovo metodo è draw_grid.

- Automa_display::**draw_grid**, quando chiamato, disegna la griglia sulla finestra grafica.

  Copia in world la classe world_to_draw (argomento della funzione preso in sola lettura). Copia in grid il vettore di Cell contenuto in world, con il metodo .get_grid(). Copia in side la lunghezza del lato della griglia contenuta in world, con il metodo .get_side. Inizializza il lato di una singola cella, dividendo la lunghezza del lato del display per la lunghezza del lato della griglia. Inizializza il numero di celle, ovvero la dimensione dell'array grid, con il metodo .size e il static_cast<int>. Inizializza l'array di 4 volte il numero di celle vertici, del tipo quads (quadrati, definiti da 4 vertici). Inizializza un contatore count uguale a 0, per poter assegnare i valori di 4 in 4, ad ogni ciclo. Itera con due for, scorrendo tutta

la griglia, in cui: inizializza come costanti le coordinate relative al punto in alto a sinistra del quadrato che rappresenta la cella, ovvero x_coord (y_coord) il numero di righe i (colonne j) moltiplicate per la lunghezza della cella; fornisce la posizione dei 4 vertici del quadrato, aggiungendo la lunghezza del lato della cella dove necessario; trova l'indice index del punto relativo alla cella che si trova alla riga i-esima e colonna j-esima, con la funzione find_index; se la cella corrispondente all'indice è Susceptible, la colora di blue, se è Infectious, di rosso, se è Recovered, di verde e se è Empty, di nero; aumenta il contatore di 4, siccome sono necessari quattro vertici per formare il quadrato colora. Una volta uscito dal ciclo, disegna l'array contenente tutte le celle.

3.3 SIR_input.hpp e SIR_input.cpp

E' stato scelto di raggruppare, nella seconda metà di questi file, la parte di codice che implementa l'input relativo al second0 modello: due funzioni (input_initial_population e input_parameter).

3.3.1 Funzioni

- **input_population**, quando chiamato, regola l'input dei tre gruppi in cui la popolazione iniziale è suddivisa. E' identico a input_population del namespace epidemic_SIR, con l'unica differenza nell'aggiunta di un controllo ulteriore: preso come argomento la lunghezza del lato della griglia, controlla che la somma dei tre gruppi non sia superiore al numero di celle grid_side * grid_side.
- **input_parameter**, quando chiamato, regola dei parametri della simulazione. E' identico a input_parameter del namespace epidemic_SIR, con l'unica differenza nell'aggiunta di un altro parametro alpha (sempre compreso tra 0 e 1), relativo alla probabilità che una persona si sposti nella griglia.

3.4 SIR_main.cpp

3.4.1 Case 2

Il case 2 inizializza la lunghezza del lato della finestra grafica display_side. Introduce la possibilità di decidere le dimensioni della griglia, conservando la scelta nella variabile grid_choice, presa in input con la funzione input_choice, e visualizzando le opzioni disponibili in output con std::cout (1 per la griglia 120 x 120, 2 per la griglia 60 x 60 e 3 per la griglia 30 x 30). Utilizza uno switch per cambiare il valore della variabile grid_side, inizializzata a 0, in base alla scelta contenuta in grid_choice. Introduce inoltre la possibilità di decidere la velocità con cui disegnare il grafico, conservando la scelta nella variabile framerate_choice, presa in input con la funzione input_choice, e visualizzando le opzioni disponibili in output con std::cout (1. per vedere 1 fotogramma al secondo, 2. per vedere 5 fotogrammi al secondo, 3. per vedere 10 fotogrammi al secondo e 4. per vedere 20 fotogrammi al secondo). Utilizza uno switch per cambiare il valore della variabile framerate, inizializzata a 0, in base alla scelta contenuta in framerate_choice. Richiede in input la popolazione iniziale (initial_population), con la funzione input_initial_population, e i parametri (parameter), con la funzione input_parameters (vedi sezione). Inizializza la classe World world, con la lunghezza del lato della griglia come argomento. Fornisce una posizione nella griglia alle persone, in base a quante e di che tipo sono state inserite nella popolazione iniziale. Aggiorna il vettore data di world. Inizializza la finestra grafica window, con la grandezza della finestra (rettangolare, di lati 2* display_side e display_side) e il nome come argomenti. Imposta il limite massimo di fotogrammi al secondo con il metodo .setFramerateLimit. Inizializza la classe Automa_display, con la finestra grafica window e la sua lunghezza del lato display_side. Inizializza la variabile booleana, running, come true. Itera un ciclo while (finché running non diventerà false, ovvero nel caso in cui ci sia l'evento di chiusura della finestra sf::Event::Closed descritto in fondo), in cui: colora lo sfondo di nero, con il metodo .clear; permette all'epidemia di progredire, con la funzione evolve; copia nel vettore data i risultati, con il metodo get_data; disegna la griglia di world, gli assi, la legenda e le linee del grafico di data, con i metodi draw_grid, draw_axes, draw_legend e draw_graph; mostra la finestra con .display e, una volta usciti dal ciclo, la chiude con il metodo .close.

4 Test

In questa sezione verrà mostrato il modo in cui sono stati testati i due modelli (file SIR_test.cpp, vedi sezione 5.1)

4.1 SIR_test.cpp

E' stato scelto di raggruppare, in questo file, la parte di codice che regola i test: 5 TEST_CASE. All'inizio del file SIR_test.cpp sono inclusi gli header file necessari: SIR.hpp, SIR_automa.hpp, SIR_output.hpp e doctest.h
I primi tre TEST_CASE sono relativi al modello delle equazioni differenziali. I primi due simulano l'epidemia con popolazione iniziale e parametri differenti, mentre il terzo verifica che simulare nuovi giorni all'epidemia già in corso sia come simulare direttamente tutti i giorni dall'inizio. In tutti e tre è presente anche la verifica dell'arrotondamento.
Gli ultimi due TEST_CASE sono relativi al modello dell'automa cellulare. Il primo verifica che ponendo un suscettibile in una griglia 5 x 5 e un infetto in una cella adiacente, la funzione neighbours conti 1 infetti vicino. Poi se ne aggiunge un altro e si controlla che siano 2. Inoltre verifica anche che la griglia sia continua, infatti è stato messo il suscettibile in alto a sinistra e l'infetto in alto a destra. Il secondo verifica che, mettendo un suscettibile accanto ad un infetto con probabilità di infettare uguale a 1, il suscettibile si infetti e che, mettendo anche la probabilità di trasformarsi in un rimosso uguale a 1, l'infetto diventi rimosso.

5 Compilazione

Non è presente la compilazione con cmake, ma quella con g++.

5.1 Test

La compilazione per i test avviene nel seguente modo: g++ SIR_test.cpp SIR.cpp SIR_output.cpp SIR_automa.cpp -lsfml-graphics -lsfml-window -lsfml-system -Wall -Wextra -g -fsanitize=address -std=c++17.

5.2 Programma

La compilazione del programma avviene nel seguente modo: g++ SIR_main.cpp SIR_input.cpp SIR.cpp SIR_output.cpp SIR_automa.cpp SIR_automa_output.cpp -lsfml-graphics -lsfml-window -lsfml-system -Wall -Wextra -g -fsanitize=address -std=c++17.

6 Note

- Quando si utilizza SFML, è presente un memory leak.
- Per la formattazione, è stato mantenuto di default il .clangformat, con l'unica modifica al valore: ColumnLimit: 100, per una maggiore fluidità nella lettura del codice.