Relazione Progetto di Programmazione

Link alla directory di git: https://github.com/FabioEhr/final-project

Introduzione

Il nostro progetto si articola in tre programmi principali: l'implementazione del modello sir standard (cartella di riferimento **sir_base**), la simulazione di un automa cellulare con la possibilità di movimento da parte delle celle e visualizzazione della griglia in standard output (nella cartella **sir_grid**) e una rivisitazione del modello tradizionale che include la suddivisione della popolazione in tre categorie principali (*Young, Adults, Elders*), la possibilità di attenuare gli effetti della pandemia tramite svariate misure di contenimento, che tuttavia affliggono l'economia ed il morale delle persone, ed eventi casuali, quali mutazioni del virus, che alterano il decorso dell'epidemia (cartella di riferimento **sir_yae**).

Modello SIR standard

1. *Contenuto della cartella sir_base*

La cartella contiene due file .hpp e due file .cpp: il file *sir.hpp* contiene tutte le strutture dati utili all'implementazione del modello classico, mentre *useful_func.hpp* è un file comune a tutte le cartelle che contiene funzioni utilizzate frequentemente, soprattutto per comparazioni fra double e controllo dell'input. Il file *sir.cpp* è il programma principale, che permette la simulazione della pandemia tramite inserimento di valori in standard output, mentre *sir.test.cpp* contiene i vari test effettuati con *Doctest* per verificare l'efficacia del programma.

2. *Contenuto di sir.hpp*

Per motivi di autoconsistenza del codice ed evitare situazioni in cui il numero totale delle persone non si conservi per via di approssimazioni, abbiamo deciso di gestire l'implementazione di entrambi i modelli deterministici tramite percentuali: i programmi non calcoleranno la variazione del numero di infetti o suscettibili ma la variazione della percentuale di infetti e suscettibili.

Come si può notare dalla *struct condition* i valori che caratterizzano la situazione epidemiologica sono esclusivamente la percentuale di *susceptibles*, *infected* e *recovered*, oltre alla durata della pandemia (*int time*). Questi valori varieranno durante l'evoluzione della pandemia secondo i parametri *contagiousness* (Beta) e *recovery_rate* (Gamma) che caratterizzano la *struct virus*. Ovviamente tutte le percentuali sommeranno sempre ad uno. Tutti i valori utili al modello sono contenuti nella *class pandemy*, e le variazioni sono calcolate dal metodo *void evolve()* della stessa, tramite le equazioni previste dal modello standard. Un altro metodo di particolare interesse è *evolveNtimes()* che a differenza di *evolve()* ritorna come valore un *std::vector* contenente valori di tipo *condition* in modo da poter accedere alla situazione di un qualsiasi giorno.

Il file contiene altre due funzioni utilizzate nel file *sir.cpp* incaricate di gestire i valori inseriti da standard input e utilizzano frequentemente funzioni contenute in *useful_func.hpp.*

3. *Contenuto di useful_func.hpp*

Questo header file nasce dall'esigenza di risolvere bug e problemi comuni a tutti e tre i programmi, in particolare il confronto fra double ed il loop infinito che scaturisce quando viene

inserito un *char* o una *string* in input invece di un numero. Infatti ogni volta che è richiesto l'inserimento di un numero naturale, l'utente andrà ad inserire una stringa di caratteri, che verrà poi convertita in *int* dalla funzione *string_to_int(),* che ignorerà tutti i caratteri non numerici attribuendogli come valore 0. Se per esempio l'utente dovesse inserire "-23r45" come valore, la funzione lo trasformerebbe in 023045, quindi il valore effettivamente inserito sarà 23045. Quando è richiesto l'inserimento di valori nel range [0,1], la funzione utilizzata è *string_to_decimal()* in combinazione con *valid_string()* e *d_comp()*; la prima è incaricata di trasformare la stringa inserita in un numero decimale, la seconda controlla che la stringa inserita non contenga caratteri non numerici (escluso il punto decimale) e l'ultima conferma che il valore inserito sia effettivamente fra 0 e 1; infatti *string_to_decimal()* è pensata esclusivamente per la conversione di stringhe in double positivi minori di 1. Le funzioni sono state testate nel file *test.cpp* della cartella *sir_yae.*

4. Contenuto di sir.cpp e sir.test.cpp

Il contenuto di questi due file è molto semplice; *sir.cpp* utilizza la funzione *createVirus(),* contenuta nell'header file *sir.hpp*, per permettere all'utente di inserire i valori utili alla creazione della situazione epidemiologica che si vuole simulare, poi sempre tramite standard input l'utente potrà inserire la durata della simulazione in giorni. I valori inseriti sono utilizzati per chiamare la funzione *evolveNtimes()* e un *for* in combinazione con la funzione *Print(condition),* che prende come argomento i valori del vettore ritornato da *evolveNtimes()*, permette la visualizzazione in standard output dell'evoluzione della pandemia. *sir.test.cpp* è stato utilizzato per verificare la corretta implementazione del modello, tramite *Doctest*, e ha confermato la significatività dei parametri *Contagiousness* e *Recovery_rate*.

La simulazione tramite Automa Cellulare

1. Contenuto della cartella sir_grid

L'implementazione del modello si sviluppa nella sua interezza nel file *siroflife.hpp* e si articola in 4 *struct* con i rispettivi metodi. Il file *print.cpp* gestisce l'input/output per l'inserimento di valori e permette la visualizzazione della griglia giorno dopo giorno, mentre *grid_test.cpp* è il file che tramite *Doctest* ci ha permesso di trovare e correggere eventuali errori nell'implementazione

2. Il modello in breve

La simulazione dell'automa consiste nella creazione di una griglia definita da numero di righe e colonne (*int height e int width)* mentre il movimento delle persone è casuale e parametrizzato da due valori interi: la *mobility* quantifica il numero di spostamenti che ogni persona attua ogni giorno, mentre la *speed* pone un limite superiore al range degli spostamenti lungo le righe e le colonne. Una *speed* di 3 per esempio implica che una persona in una posizione (*r,c),* chiamata la funzione *rndmove(),* potrà trovarsi in una cella qualsiasi inclusa nel quadrato di vertici opposti (*r-3,c-3) (r+3,c+3),* qualora questi valori non siano superiori a *height e width*. Inizialmente potrebbe sembrare che questo parametro sia inversamente proporzionale all'aumento di contagi, poiché non si tratta di un movimento *cell by cell* come avverrebbe con una *speed* di 1, apparentemente riducendo la probabilità di incontro fra cella infetta e suscettibile; tuttavia i test hanno smentito questa ipotesi: infatti una *speed* elevata permette alle celle di uscire dal *cluster*

locale, incontrando persone differenti. Questo effetto è particolarmente evidente su griglie di elevata dimensione.

3. *siroflife.hpp: la struct Virus*
   La *struct Virus* può sembrare molto simile a quella implementata nel modello base, però i parametri *contagiousness* e *recovery_rate* hanno un significato differente: non sono più dei valori che quantificano il passaggio da numero di persone *susceptibles-infected* o *infected-recovered*, ma delle probabilità; ogni persona suscettibile che condivide la posizione con una persona infetta ("respira la stessa aria") avrà una probabilità pari a *contagiousness* di essere contagiata, mentre ogni *infected* avrà probabilità *recovery_rate* di passare a *recovered*. Abbiamo scelto di implementare un passaggio intermedio fra *susceptible* ed *infected*: le persone appena contagiate saranno in uno stadio di incubazione del virus, e non saranno inizialmente infettive ma lo diventeranno dopo un numero pari a *incubation_time* di giorni.

4. *siroflife.hpp:le struct Cell e Person*
   Ogni persona è caratterizzata da un intero che rappresenta la sua condizione, il valore 0 è attribuito ai suscettibili, 1 agli infetti, 2 ai recovered e 3 alle persone infette ma con il virus in uno stadio di incubazione; per tenere traccia del periodo di incubazione del virus abbiamo utilizzato un altro numero naturale (*int incub_day*). L'informazione sulla posizione di ogni persona è contenuta all'interno di una variabile di tipo *Cell* (*P_cell*), che racchiude una coppia di *int* che identificano la riga *(r)* e la colonna (*c)* su cui è posizionato l'individuo. Per la *struct Cell* sono stati definiti vari *operator* che consentono l'utilizzo di *container* e *sorting alghoritms* che migliorano notevolmente l'efficienza. Di particolare interesse è il metodo *rndmove()* della *struct Person*, che modifica la posizione della persona in base ai parametri sopracitati *speed, mobility, height e width.* Il movimento è caratterizzato da numeri casuali estratti da una distribuzione uniforme di numeri interi compresi nell'intervallo *[-speed,speed]*.

5. *siroflife.hpp: la struct Grid*
   Ogni griglia è caratterizzata da numero di righe (*height*), numero di colonne (*width*), dal numero di persone (*population*), che si dividono in *susceptible, infected e recovered* (le persone che stanno incubando il virus vengono conteggiate come infette) e un *std::vector<Person>* nominato *people* vuoto di default; per istanziare un oggetto di tipo *Grid* dovranno essere forniti 5 interi (*height, width, infected, susceptible, recovered*) e il costruttore si occuperà di riempiere correttamente *people* (tramite il metodo *push_back()*), fornendo una posizione iniziale casuale ad ogni persona. Al termine di questo processo la dimensione di *people* sarà uguale a *population*. Il metodo *sus_evolve(Virus)* gestisce il passaggio da *susceptible* a *infected* secondo i seguenti step:
   1) Creazione di un *std::set<Cell> (inf_cells)* contenente la posizione delle celle infette.
   2) Scansione del vettore *people*, utilizzo del metodo *find()* dei *set* per trovare persone suscettibili che condividono la cella con persone infette.
   3) Qualora questa ricerca vada a buon fine, estrazione di un numero casuale da una P.D.F uniforme nel range [0,1], se il numero estratto dovesse essere minore o uguale della *contagiousness* del virus, la persona passerà da *susceptible* ad *incubating infection*.

   Il metodo *move_and_evolve()* gestisce il movimento delle persone e l'infezione utilizzando *rndmove()* e *sus_evolve()*, il passaggio da *infected* a *recovered* in maniera analoga a *susceptible-infected*, e l'aumento di *incub_day* con l'eventuale passaggio da *incubating infection* a *infected*.

6. siroflife.hpp: convenzioni e metodi per visualizzare la griglia

Per la rappresentazione grafica del modello abbiamo stabilito delle convenzioni e gerarchie di priorità per gestire al meglio il fatto che più persone possono essere sulla stessa cella. Queste sono:

1) Un gruppo di *susceptible* viene rappresentato con *char S*
2) Un gruppo di *infected* viene rappresentato con *char I*
3) Un gruppo di persone nello stato *incubating infection* viene rappresentato da *char F*
4) Un gruppo di *recovered* viene rappresentato da *char R*
5) L'incontro tra un gruppo di *infected* e *susceptibles* viene rappresentato da *char !*
6) L'incontro tra *susceptibles* e persone nello stato *incubating infection* è rappresentato da *char #*
7) La priorità in un incontro *infected-incubating infection* viene data ad *infected*, verrà quindi rappresentato da *I*
8) La priorità in un incontro *susceptible-recovered* viene data a *susceptible*, verrà quindi rappresentato da *S*
9) La priorità in un incontro *recovered-incubating infection/infected* viene data ad *incubating infection/infected* verrà quindi rappresentato da *F/I*
10) Gli spazi vuoti vengono rappresentati da *char* –

La funzione *get_map(Grid)* utilizza le convenzioni stabilite per trasformare la griglia in un *std::vector<char>*, che poi viene fornito a *draw_map()*, che stampa il risultato in standard output.

7. Contenuto di print.cpp

Analogamente a *sir.cpp*, questo file gestisce l'input dell'utente, che sarà invitato ad inserire i valori utili al modello da standard inuput, e permette la visualizzazione della griglia, del numero di suscettibili, infetti e guariti del giorno corrente, secondo le convenzioni stabilite. Per far passare un giorno è sufficiente inserire un input qualsiasi.

8. Contenuto di grid_test.cpp

I test effettuati tramite *Doctest* si sono rivelati utili alla verifica della corretta implementazione e soprattutto una maggior comprensione del modello; di notevole interesse è il test sul parametro *speed*, che ha estinto i nostri dubbi sull'efficacia di quest'ultimo. Prima di eseguire i test è consigliabile commentare la stampa in standard output del messaggio di avvenuto contagio nella funzione *sus_evolve()*.

Modello Sir Rivisitato (YAE)

1. Contenuto della cartella sir_yae

Il codice è suddiviso in 6 header file, ognuno dei quali contiene strutture dati utili alla simulazione di aspetti differenti della pandemia:

1) *yae.hpp* contiene le classi e i metodi principali, con le quali si può simulare una città, composta da diverse fasce d'età, in preda ad un'epidemia
2) *decisions.hpp* contiene *free functions* che simulano le varie misure di prevenzione e contenimento dell'epidemia da noi proposte

3) *events.hpp* contiene i vari eventi simbolo di una città in crisi epidemiologica che possono alterare notevolmente la situazione ed una funzione che gestisce l'aspetto probabilistico di questi avvenimenti in base ai parametri della città
4) *interface.hpp* racchiude tutti gli aspetti di interfaccia utilizzati nel programma principale (*game_main.cpp*)
5) *presets.hpp* contiene una serie di presets, ideati per rendere meno tediosa la fase di inizializzazione delle variabili e calibrati secondo i valori da noi attribuiti alle varie misure di contenimento
6) *useful_func.hpp* è il file presente in tutti i programmi e contiene funzioni utilizzate frequentemente

I 3 file *.cpp* sono:

1) *game_main.cpp* contiene una simulazione completa della pandemia secondo i presets, gli eventi e le misure di contenimento da noi proposte, le quali vengono gestite tramite standard input
2) *test.cpp* contiene dei test svolti tramite *Doctest* per verificare il corretto funzionamento di alcuni aspetti del programma
3) *more_tests.cpp* è stato il file di riferimento per la calibrazione e la verifica del corretto funzionamento delle strutture da noi proposte; fornisce un numero elevato di informazioni su valori normalmente non accessibili all'utente e permette la visualizzazione degli effetti delle misure istante per istante. È l'unico file non formattato tramite *.clang-format* per motivi di convenienza.

2. Il modello: contenuto del file yae.hpp

Le fasce d'età: struct Age

Ogni fascia d'età è caratterizzato da:

1) 5 double la cui somma è sempre 1, poiché rappresentano la percentuale di *susceptible*, *infected*, *recovered*, *ospedalizzati (hosp)*, *dead*, all'interno della fascia d'età.
2) Due interi che rappresentano *income* e *morale* di ogni persona che appartiene al gruppo di età, utilizzati per simulare gli effetti delle misure attuate contro la pandemia sull'economia e sulla psiche delle persone.
3) Due modificatori (double) che alterano le probabilità di morte in seguito all'infezione (*d_mod)* e di criticità della malattia (*h_mod),* in maniera additiva.

Abbiamo scelto di rappresentare 3 fasce d'età: *Young* (anni 0-25), *Adults* (anni 26-65), ed *Elders* (anni 65+).

La matrice di mobilità: struct Transmatrix

Avendo suddiviso la popolazione in tre categorie, è stato necessario implementare una modalità di calcolo delle interazioni fra persone di fasce d'età differenti, assumendo che mediamente gli incontri avvengano in maggior numero fra persone della stessa categoria; per esempio, un giovane avrà mediamente più incontri con altri giovani che con anziani.
Inoltre i giovani effettueranno mediamente un numero di contatti superiore rispetto agli anziani.

Per modellare al meglio queste discrepanze nel numero di incontri è utile immaginare una matrice simmetrica 3x3 dove la diagonale principale è occupata da numeri reali che modellano gli incontri in fasce d'età omogenea *Young-Young (yy), Adults-Adults (aa), Elders-Elders (ee).* Gli altri 6 posti sono occupati da 3 numeri reali che modellano gli incontri *Young-Adults (ya), Young-Elders (ye), Adults-Elders (ae),* che evidentemente sono uguali agli incontri *Adults-Young, Elders-Young, Elders-Adults.*

	YOUNG	ADULT	ELDER
YOUNG	YOUNG-YOUNG ENCOUNTER	YOUNG-ADULT ENCOUNTER	YOUNG-ELDER ENCOUNTER
ADULT	ADULT-YOUNG ENCOUNTER	ADULT-ADULT ENCOUNTER	ADULT-ELDER ENCOUNTER
ELDER	ELDER-YOUNG ENCOUNTER	ELDER-ADULT ENCOUNTER	ELDER-ELDER ENCOUNTER

La struct Virus

Abbiamo scelto di caratterizzare il virus in base a tre parametri nel range [0;1]:

Beta: misura la contagiosità del virus (*double b).*

Gamma: misura il recovery rate, la percentuale di infetti che guarisce dal virus ad ogni iterazione (*double g).*

Delta: misura il death rate, la percentuale di infetti che muore dal virus ad ogni iterazione (*double d).*

H: misura il rateo di criticità, la percentuale di infetti che cadono in condizioni critiche, quindi necessitano di essere ospedalizzati (*double h).*

Ricordiamo che la contagiosità effettiva sarà calcolata in base alla mobilità, quindi risulterà più alta fra i giovani, mentre la mortalità e il rateo di criticità effettivi dipenderà dai modificatori interni alla classe *Age.*

Il sistema sanitario: la struct Hospitals

Una percentuale degli infetti, in base alla loro fascia d'età ed il parametro *H* del virus, andrà in condizioni critiche, necessitando quindi di essere ospedalizzati. La s*truct Hospitals* racchiude informazioni volte a caratterizzare il sistema sanitario quali, numero di posti letto totali (int *n_beds),* numero di posti letto occupati (*int patients),* e qualità dei macchinari (*int level),* che

influenza i due modificatori associati alla recovery rate (*double r_chance_mod)* e alla death rate (*double d_chance_mod)* degli ospedalizzati; il loro destino sarà infatti la morte o l'immunità dal virus.

La città: class City

La classe *City* racchiude tutte le informazioni necessarie a simulare una città in preda ad un'epidemia entro i limiti del modello.

Essa infatti è caratterizzata dalla popolazione totale (*population)*, la percentuale di giovani(*y_per)*, adulti (*a_pe*r) e anziani (*e_per)* che sommano ad 1, le caratteristiche delle fasce d'età, rappresentate con 3 variabili del tipo *Age (y,a,e)*, una variabile di tipo *Virus (vir)*, la matrice di mobilità *(Transmatrix mob)*, il sistema sanitario (*Hospitals h)*, il denaro nelle case comunali (*int treasury*), un intero che rappresenta lo stato di avanzamento della ricerca nei confronti di un vaccino (*int know)* e una *state_function (stat)*consistente principalmente da delle variabili booleane che tengono traccia delle chiusure e dei provvedimenti presi per contrastare l'epidemia. Tutta l'evoluzione della pandemia e le scelte riguardanti chiusure e quant'altro sono gestite tramite *free functions* e metodi *void* che vanno ad aggiornare i valori all'interno della classe *City.*

L'evoluzione della pandemia: il metodo evolve()

L'aspetto puramente epidemiologico dell'evoluzione della pandemia è gestito in maniera deterministica, tramite il metodo della classe *City* chiamato *evolve().* Come per il modello classico, abbiamo implementato tutti i calcoli secondo percentuali e variazioni sulle percentuali. Per semplicità e chiarezza riassumiamo brevemente i calcoli svolti da evolve() ad ogni iterazione:

1) *Nuovi contagi:* Calcolo di quanto varia la percentuale di persone infette e suscettibili all'interno di ogni fascia d'età, in base alla mobilità e la contagiosità (*Beta*). Per i giovani, ad esempio, verranno considerati gli incontri *Young-Young, Young-Adult, Young-Elder.* Aggiornamento delle percentuali di infetti (*inf*) e suscettibili (*sus)* all'interno della fascia d'età. Dichiarazione delle variabili *current_infected;* per come abbiamo definito *Gamma, Delta e H,* i calcoli successivi devono essere svolti sulla stessa quantità di infetti

2) *Casi critici:* Calcolo della variazione della percentuale, all'interno di ogni fascia d'età, di persone che necessitano l'ospedalizzazione, in base alla percentuale corrente di infetti, il parametro *H* del virus ed il modificatore associato alla *struct Age (h_mod).* Calcolo della differenza fra posti letto disponibili e persone che necessitano di essere ospedalizzati (*overflow*), aggiornamento delle morti in base all'*overflow*; chiunque non riesca ad entrare in ospedale per via della mancanza di posti letto verrà considerato morto. Aggiornamento dei parametri *hosp* e *inf* in base ai valori calcolati.

3) *Recoveries:* Calcolo variazioni da infetti a guariti in base al recovery rate (*Gamma)*. Calcolo variazioni da ospedalizzati a guariti in base al recovery rate e al modificatore interno alla classe *Hospitals* (*r_chance_mod*). Aggiornamento dei valori per ogni fascia d'età.

4) *Morti:* Calcolo della percentuale di infetti destinata alla morte per ogni fascia d'età, in base al parametro death rate (*Delta)* del virus e il modificatore (*d_mod)* della *struct Age*. Calcolo di morti in ospedale tenendo conto del modificatore *d_chance_mod* della *struct Hospitals*. Aggiornamento delle percentuali per ogni fascia d'età.

Il calcolo dei nuovi infetti è stato modellato in relazione alla probabilità di contagio considerando l'appartenenza di un soggetto ad una determinata fascia d'età e alla mobilità. Questa probabilità è pari alla somma delle probabilità che avvenga un contatto con un giovane/adulto/anziano infetto (stimato sulla base del fattore moltiplicativo relativo alla mobilità), meno la somma delle probabilità che sia contagiato da un giovane e da un adulto, da un giovane e da un anziano, o da un adulto e un anziano, più la probabilità che venga contagiato sia da un giovane, che da un adulto, che da un anziano, moltiplicata per la probabilità di contagio dato il contatto (*Beta)*. In formule, considerando gli eventi contagio da giovane (A), contagio da adulto (B) e contagio da anziano (C), la probabilità di essere contagiato è: $P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C)$. A livello logico, se si considerasse solo la somma delle probabilità dei tre eventi, si commetterebbe un errore sistematico di sovrastima, in quanto si conterebbe due volte il caso in cui una persona venga contagiata prima da un giovane e poi da un adulto, quando in realtà è sempre la stessa persona ad essere contagiata più volte.

```
double p_y_1 = (mob.yy * y.inf + mob.ya * a.inf + mob.ye * e.inf);
double p_y_2 = -mob.yy * y.inf * mob.ya * a.inf - mob.yy * y.inf * mob.ye * e.inf - mob.ya * a.inf * mob.ye * e.inf;
double p_y_3 = mob.yy * mob.ya * mob.ye * y.inf * a.inf * e.inf;
double D_y = vir.b * y.sus * (p_y_1 + p_y_2 + p_y_3);
y.sus -= D_y;
y.inf += D_y;
```

Figura: calcola della variazione (D_y) sulla percentuale di giovani infetti e suscettibili all'interno di evolve(). La variabile mob è la matrice di mobilità (Transmatrix), vir è di tipo Virus, e y di tipo Age

Per quanto riguarda la gestione dei posti letto, abbiamo dichiarato una variabile *double overflow=(total_patients-n_beds)/new_patients* dove *total_patients* tiene conto dei posti letto precedentemente occupati e dell'aumento di persone che necessitano l'ospedalizzazione. Qualora questa variabile sia negativa, quindi i posti letto siano maggiori delle persone che necessitano l'ospedalizzazione, viene resa nulla tramite un semplice *if*, mentre qualora fosse positiva, viene utilizzata per calcolare il numero di persone che possono essere ospedalizzate e il numero di morti per mancanza di posti letto. *Overflow* è sempre normalizzato ad 1, in quanto *total_patients* può essere al massimo maggiore di *n_beds* di una quantità pari a *new_patients* che corrisponde al caso di ospedali già pieni dall'iterazione precedente.

Funzioni per aggiustare i modificatori: mod_fixers()

Nel file sono contenute varie funzioni per controllare che i modificatori inseriti siano validi; trattandosi di modificatori di percentuali è importante che, sommati ai parametri del virus, siano nel range [0,1]. Questi controlli sono effettuati da funzioni nominate *mod_fixers()*, incaricate anche di modificare questi parametri qualora la condizione necessaria non sia verificata.

3. *Le misure di prevenzione e gestione della pandemia: contenuto di decisions.hpp*

Nel file *"decisions.hpp"* sono racchiuse tutte le misure che si possono attuare per attenuare l'evoluzione della pandemia. Si trattano di *free functions* di tipo *void*, che prendono come parametro una referenza ad un oggetto della classe *City*, e vanno a modificarne i valori tramite i metodi della classe. I parametri che vengono modificati principalmente sono il *morale*, e l'*income*, da intendere come quantità di denaro che entra nelle casse comunali, per ogni individuo, ogni turno. I metodi più utilizzati sono:

1) *add_mob(parametri: 6 double):* aggiunge alla matrice di mobilità i valori inseriti come parametri, sommando il primo parametro a *yy* (elemento in alto a sinistra), il secondo ad *aa*, e proseguendo secondo l'ordine *ee*, *ya*, *ye*, *ae.*
2) *multiply_mob(parametri: 6 double):* analogamente ad *add_mob*() moltiplica i parametri della mobilità secondo lo stesso ordine.
3) *add_$(int cost)* modifica la quantità di denaro nelle casse comunali (*treasury*) di una quantità *cost*
4) *Set_ages(parametri: 3 oggetti di tipo Age):* sostituisce nel seguente ordine *Young*, *Adults* e *Elders* interni alla classe *City.*
5) *Set_status(oggetto di tipo State_function):* Modifica la *state_function* della classe *City* per tenere traccia delle misure attuate.

Le decisioni si possono classificare in 5 categorie:

1) *Chiusure ed aperture:* è possibile scegliere se chiudere ristoranti, scuole, chiese, teatri (e cinema) oppure tutto (*lockdown*). Le chiusure diminuiscono il *morale* e l'*income* di una o più fasce di popolazione in base a cosa si è scelto di chiudere: per esempio, chiudere le scuole ha una maggiore influenza sulla mobilità *Young-Young*, mentre la chiusura delle chiese affligge principalmente la mobilità *Elder-Elder*. Ad ogni funzione *close* è associata una funzione simmetrica *open*, che tiene conto dell'effetto "euforia": riaprire un'attività che è stata precedentemente chiusa comporta un aumento netto del morale ma anche della mobilità.
2) *Coprifuoco:* Scelta questa opzione, verrà chiesto di inserire il numero di ore di coprifuoco, fino ad un massimo di 24 ore. Riduce la mobilità e il morale in base alla durata. Alleviare il coprifuoco comporta un aumento netto del morale ma anche della mobilità.
3) *Investimenti:* Comportano una riduzione del *treasury* in favore di vari vantaggi, come la riduzione della mobilità nel caso si acquistino mascherine (*buy_masks()*), miglioramento degli ospedali e aumento di posti letto (*build_beds()*, *modernize_hospitals()*), stimolare la ricerca per il raggiungimento di un vaccino (*invest_in_research()*), oppure migliorare le infrastrutture per ridurre il digital divide (*invest_in_digital)*.
4) *Utilizzo dei media:* Tramite articoli di giornale si può modificare la percezione del pericolo effettivo del virus, aumentando o diminuendo la mobilità e/o il morale (*tranquillize_with_media(), terrorize_with_media()*).
5) *Vaccinazioni:* Raggiunto un livello di *knowledge* sufficientemente alto sarà possibile istituire delle campagne vaccinali, decidendo quali fasce d'età vaccinare prima, diminuendo il numero di *susceptibles* in favore dei *recovered*.

4. *Gli eventi casuali: contenuto del file events.hpp*
   Nel file "*events.hpp*" sono definite alcune funzioni volte a simulare eventi casuali caratterizzanti dello scenario di una città in preda alla pandemia. Secondo noi i principali sono:
   1) *Mutazioni del virus:* modifiche dei parametri *Beta*, *Gamma* e *Delta*, avvengono con maggior probabilità quando il numero di infetti è molto elevato. Abbiamo scelto di simulare solo le mutazioni "negative", ovvero che aumentano *Beta e Delta*, oppure diminuiscono *Gamma*. La mutazione del virus è globale, il nostro modello non prevede la coesistenza di più varianti.
   2) *Proteste e super-diffusori:* quando il *morale* è basso e le chiusure sono state fatte troppo preventivamente agli occhi della popolazione (numero di infetti basso), le persone si riuniranno in piazza per protestare, aumentando la mobilità. Se la percezione del pericolo è bassa, la popolazione organizzerà feste, aumentando notevolmente il numero di contagi mentre, in caso opposto, il panico generale porterà ad una riduzione del *morale* e della mobilità.
   3) *Sequenziamento del virus:* investire nella ricerca aumenta la probabilità che medici e scienziati riescano a sequenziare il virus, aumentando la *knowledge* necessaria al raggiungimento del vaccino contro il virus.
   4) *Posti letto non sufficienti:* l'unico evento non casuale, ma determinato dalla capienza degli ospedali. Diminuisce drasticamente il morale.
      La funzione incaricata di sorteggiare quali eventi avvengono ad ogni ciclo è *rnd_events()*.

5. *Contenuto dei files presets.hpp e interface.hpp*
   Vista l'elevata quantità di variabili necessarie per la simulazione della pandemia, abbiamo definito dei presets che forniscono differenti combinazioni interessanti da analizzare. In particolare proponiamo 3 tipi di *Virus: Flu (*alta contagiosità e recovey rate, bassa mortalità e rateo di criticità), *Covid* (mortalità e rateo di criticità più elevati), *Ebola* (letalità elevata), oltre a 3 città-modello: *Matera* (50.000 abitanti, popolazione prevalentemente anziana, bassa mobilità), *Bologna* (400.000 abitanti, popolazione prevalentemente giovane, mobilità elevata) e *Milano* (1.300.000 abitanti, popolazione prevalentemente adulta, mobilità estrema). La maggior parte di ciò che riguarda lo standard output è gestito nell'header file *interface.hpp;* la funzione di maggior interesse in questo file è *execute(City, char)*, che ha il compito di attuare le scelte dell'utente tramite le funzioni contenute in *decisions.hpp*.

6. *Il game-loop: contenuto del file game_main.cpp*
   Avviando il programma all'utente verrà chiesto di scegliere fra i tre possibili virus e le tre possibili città contenute nell'header file *presets.hpp*. La scelta di usare delle città e virus predefinite è dovuta principalmente al bilanciamento delle varie scelte ed eventi, oltre all'elevato numero di parametri da inserire all'interno della classe *City*, che renderebbe lungo e tedioso il processo di customizzazione. Ad ogni ciclo (corrispondente ad una settimana), verranno fornite all'utente informazioni sull'andamento della pandemia, a cui sarà poi fornita una lista delle possibili decisioni attuabili per contrastare l'epidemia. Il loop si interrompe quando il programma approssimerà l'intero *Omega=population*(per_inf+per_hosp)* (popolazione per percentuale di infetti più percentuale di ospedalizzati) a 0 e l'utente potrà visualizzare dei valori di particolare interesse, quali numero totale di morti e durata della epidemia.

7. *I tests: test.cpp e more_tests.cpp*
   Per la fase di testing e bug fixing abbiamo utilizzato prevalentemente *more_tests.cpp.* Questo file è strutturato in modo da poter modificare rapidamente i valori utilizzati dal modello eliminando i commenti da alcune parti di codice e visualizzare l'effetto sul decorso della pandemia in standard output. Il modello non è pensato per mobilità esageratamente elevate, poiché crolla l'ipotesi principale sul quale è costruita la modellizzazione secondo la probabilità dell'evento intersezione; quando i fattori moltiplicativi sono eccessivamente elevati, è possibile che la variazione della percentuale di infetti risulti essere negativa, anche quando non dovrebbe esserlo. Per questo motivo abbiamo inserito un *if* di controllo dentro ad *evolve()* che, al verificarsi di questo caso limite, fa diventare infetti tutti i suscettibili. Il file *test.cpp* invece utilizza *Doctest* per verificare il funzionamento atteso di varie funzioni.

8. *Commenti finali sul modello SIR YAE*
   Il programma secondo noi svolge un buon lavoro nel mostrare gli effetti delle misure di contenimento e i rischi associati alle riaperture: l'epidemia, infatti, può ripartire velocemente anche con un numero di infetti basso. Parametri come morale ed eventi casuali sono estremamente difficili da modellare, ma forniscono un'approssimazione di quanto accade alla popolazione di una città colpita da pandemia.