Politecnico di Torino
Dipartimento di ingegneria gestionale e della produzione

Corso di Laurea Triennale in Ingegneria Gestionale
Classe L8 – Ingegneria dell'informazione A.A. 2022/2023

Simulatore emergenze stradali

Relatore
Prof. Fulvio Corno

Candidato
Michele Della Mura
283738

Settembre 2023

Capitolo 1: Proposta di progetto

1.1 Studente proponente

s283738 Della Mura Michele

1.2 Titolo della proposta

Simulatore incidenti stradali

1.3 Descrizione del problema proposto

I sinistri sono eventi che richiedono una rapida risposta da parte delle unità di soccorso per fornire assistenza medica tempestiva alle vittime coinvolte. Tutto ciò richiede una serie di azioni coordinate, come: la valutazione della gravità e la localizzazione delle ambulanze più vicine all' incidente, il calcolo del percorso ottimale per raggiungere il luogo del sinistro e il coordinamento delle risorse disponibili.
Il progetto vuole quindi affrontare il problema di gestione di allocazione di veicoli di soccorso negli ospedali situati nei diversi quartieri della città di Los Angeles attraverso una simulazione di un pronto intervento di ambulanze al verificarsi di incidenti stradali.

1.4 Descrizione della rilevanza gestionale del problema

L'obiettivo principale è quello di creare un ambiente simulato che permetta, attraverso l'analisi dei risultati di output, di osservare il numero di persone soccorse in funzione dei parametri di input impostati dall'utente: il numero di distretti che possiede un ospedale e il numero di ambulanze totali che ha la città americana.
Dal punto di vista gestionale risolvere questo problema è di grande importanza per le strutture ospedaliere e per la stessa città di Los Angeles. Infatti, per le prime permette di stimare all'incirca quante ambulanze devono avere per poter riuscire a garantire il loro servizio a più persone possibili; per la città americana permette di valutare il numero ottimale di quartieri che devono avere un ospedale (ipotizzo che un quartiere possa avere al massimo un ospedale).

1.5 Descrizione dei data-set per la valutazione

Il database riflette gli incidenti di collisione stradale nella città di Los Angeles a partire dal 2010 fino al 2023.
In particolare il data-set utilizzato è costituito da una tabella che contiene 594000 righe, dove ogni riga rappresenta un incidente stradale. Ogni incidente è poi rappresentato da distinte caratteristiche, identificate dai nomi delle diverse colonne. Nel progetto in esame vengono utilizzate le seguenti colonne:
DR Number, Date Reported, Area ID, Area Name, Victim Age, Premise Description, Location.

Il data-set è disponibile al seguente link:
data.lacity.org/Public-Safety/Traffic-Collision-Data-from-2010-to-Present/d5tf-ez2w

1.6 Descrizione preliminare degli algoritmi coinvolti

Gli algoritmi coinvolti sono un algoritmo di simulazione e un algoritmo sulla ricerca di cammini minimi tra un nodo sorgente ed uno di destinazione all'interno di un grafo, cioè l'algoritmo di Dijkstra.
La simulazione è impostata come segue:

1. Inizialmente tutte le ambulanze si trovano nell'ospedale del distretto di appartenenza. L'utente imposta un valore come parametro di ingresso per i distretti che devono avere un ospedale (e quindi una/delle ambulanza/e). Se ad esempio immette il valore 3, vengono scelti casualmente tre distretti che possiederanno un ospedale.

2. Successivamente al verificarsi di un evento interviene la croce rossa disponibile dell'ospedale del distretto più vicino al luogo dell'incidente. Nello specifico:

   - se vi è un' ambulanza disponibile nello stesso distretto dove è avvenuto l'evento allora essa ci mette: 5 minuti a raggiungere il luogo dell'incidente, in condizioni normali di traffico; 8 minuti, se c'è ingorgo stradale;

   - se non vi è un' ambulanza disponibile nello stesso distretto dove è avvenuto l'evento (o perché le ambulanze di quel distretto sono tutte occupate o perché quel distretto non ha un ospedale) allora viene scelto il nodo che ha la somma dei pesi degli archi minima e che si collega al vertice dove è avvenuto l'incidente: viene scelto quindi il nodo che è meno distante in termini di chilometraggio dal vertice destinazione. Infatti il peso di ogni arco è rappresentato dai km di distanza che ci sono tra i vari quartieri. Per scegliere il nodo di partenza dell'ambulanza per raggiungere il nodo di destinazione dell'incidente viene utilizzato l'algoritmo di Dijkstra.

   Il tempo di percorrenza viene poi calcolato attraverso i km di distanza tra il nodo sorgente e il nodo destinazione e la velocità dell'ambulanza, che è in funzione delle condizioni di traffico stradale: 40 km/h in condizioni di ingorgo; 70 km/h altrimenti.
   La presenza di traffico è un dato scelto dall'utente in input attraverso l'inserimento di un numero da 0 a 1 che rappresenta la probabilità che l'ingorgo stradale si verifichi.

3. Ogni evento è rappresentato da un codice di emergenza diverso. Il codice è rosso quando la vittima risulta essere una persona con età minore di 18 o maggiore di 70 anni. In ogni altro caso il codice risulta giallo.
   Per codici rossi l'ambulanza deve arrivare entro i 10 minuti rispetto alla data in cui è avvenuto l'incidente (colonna "Reported Date") mentre per codici gialli entro i 20 minuti. Se ciò non accade, la vittima si considera deceduta. Viceversa se la croce

rossa arriva in tempo sul luogo, per prestare soccorso al ferito impiega 15 minuti per codici rossi e 30 minuti per codici gialli.

4. Infine il veicolo di soccorso deve tornare all'ospedale di appartenenza per permettere alla vittima stradale di avere cure più specifiche. Il tempo di percorrenza per tornare al nodo di partenza è pari allo stesso tempo calcolato al punto 2.
   Ritornato al vertice sorgente il veicolo risulta libero per soccorrere altre vittime.

1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software

1. All'avvio del programma viene richiesto all'utente di scegliere da un menù a tendina un numero da 2010 a 2022 che rappresenta l'anno in cui si vuole considerare gli incidenti stradali (non considero i dati del 2023 poiché è l'anno in corso).

2. Successivamente è possibile cliccare sul pulsante "Crea città". Il bottone crea un grafo con i vertici che rappresentano i ventuno distretti della città (rappresentati dalle colonne "Area ID" e "Area Name") di quell'anno considerato mentre il peso degli archi non orientati rappresenta la distanza in km tra un distretto ed un altro. La distanza è calcolata attraverso la latitudine e la longitudine propria di ogni distretto, che si trova nella colonna "Location".
   Dato che la latitudine e la longitudine nel database non rappresentano le coordinate del quartiere della città ma rappresentano la strada specifica di quel distretto dove è avvenuto l'incidente, posso:
   - calcolare la media di tutti i valori di latitudine e di longitudine di quegli incidenti che appartengono allo stesso distretto (stesso valore nella colonna "Area ID");
   - impostare il valore mediano come coordinate geografiche del distretto.

3. Dopo la creazione del grafo viene richiesto all'utente di immettere i parametri della simulazione:

   - un menù a tendina permette di specificare il mese in cui iniziare la simulazione nell'anno selezionato nel punto 1;
   - tre campi di testo permettono di inserire un numero: il primo indica quanti distretti presentano degli ospedali e quindi delle ambulanze (da un minimo di uno ad un massimo di ventuno), il secondo si riferisce al totale di ambulanze presenti nella città mentre il terzo si riferisce alla probabilità che si verifichi un ingorgo stradale (un valore da 0 a 1).
     Si ipotizza che un distretto possa avere al massimo un ospedale.

4. In seguito cliccando sul bottone "Simula" è possibile lanciare la simulazione.

In output saranno visibili in un' area di testo il numero di persone salvate, il numero totale di sinistri che si sono verificati in quell'arco temporale, il numero di ospedali e di ambulanze selezionati come valori di input, le aree della città che presentano un ospedale e i tipi di struttura o luoghi in cui si sono verificati gli incidenti (colonna "Premise Description").

Capitolo 2: Descrizione dettagliata del problema affrontato

2.1 Introduzione

L'allocazione di risorse riveste un ruolo assolutamente cruciale nell'ambito di uno Stato, poiché questa delicata pratica esercita un impatto profondo e diretto su molteplici aspetti della vita cittadina e sulla capacità di affrontare con successo le molteplici sfide connesse all'ambiente urbano in costante evoluzione.

L'allocazione oculata delle risorse si traduce in un uso efficiente e razionale dei finanziamenti, dei servizi e delle infrastrutture disponibili, garantendo che ogni aspetto della vita urbana sia trattato con la giusta attenzione. In altre parole, essa costituisce il meccanismo tramite il quale diversi elementi, dalla salute pubblica all'istruzione, dalla sicurezza all'ambiente, vengono gestiti in maniera ottimale.

Non si può sottovalutare inoltre l'impatto diretto che l'allocazione di risorse ha sulla qualità della vita dei cittadini. Un'allocazione mirata ed equa può portare a infrastrutture più solide, servizi pubblici efficienti e una maggiore coesione sociale. Al contrario, una distribuzione inefficiente delle risorse può compromettere la capacità di reazione alle emergenze e creare tensioni sociali.

2.2 Contesto operativo/statale

Nell'ambito del progetto in esame, si vuole concentrare l'attenzione sul problema di determinare il numero di ospedali da costruire all'interno di una città e la loro collocazione strategica nei vari quartieri, insieme alla determinazione del numero totale di ambulanze; tutto ciò costituisce un aspetto fondamentale della gestione sanitaria e dell' efficace erogazione dei servizi medici.

Per affrontare tali criticità viene effettuata un' analisi attraverso l'utilizzo di dati provenienti da un database che contiene numerosi incidenti stradali avvenuti nel corso degli anni nella città di Los Angeles.

Attraverso questo ambiente simulato sarà così possibile aumentare l'accesso ai servizi medici e ridurre i tempi di percorrenza per le cure.

In sintesi, si crea un sistema sanitario equo ed efficiente che può affrontare meglio le sfide sanitarie e contribuire a migliorare la qualità della vita nella città. La rilevanza di questo sottoproblema si riflette nell' impatto diretto sulla salute e sulla sicurezza dei cittadini, nonché nell'efficienza globale dei servizi sanitari della città.

Durante lo svolgimento della simulazione, sono state apportate le seguenti semplificazioni:

- Ciascun distretto può ospitare al massimo un ospedale.
- I tempi limite per i diversi codici delle vittime sono stati selezionati in modo arbitrario.
- I tempi di cura per le vittime sono stati scelti in maniera arbitraria.
- I tempi di percorrenza tra l'ospedale e il luogo dell'incidente, così come il percorso inverso, sono identici.
- I tempi di percorrenza per le collisioni che si verificano nello stesso distretto sono stati scelti in modo arbitrario.
- La velocità dell'ambulanza è stata assegnata con un valore arbitrario.
- Non è stato considerato il tempo necessario all'ambulanza per prepararsi prima di partire per prestare soccorso.
- Se il distretto dell'ospedale da cui parte l'ambulanza è il medesimo distretto nel quale si verifica la collisione, allora la vittima è automaticamente salva.

Capitolo 3: Descrizione del data-set

Il data-set utilizzato per l'analisi contiene informazioni dettagliate sugli incidenti di collisione stradale verificatisi nella città di Los Angeles nel periodo compreso tra il 2010 e il 2023. Questo insieme di dati è rappresentato da una tabella composta da 594000 voci, ognuna delle quali corrisponde a un singolo evento di incidente stradale. Ogni voce è caratterizzata da una serie di attributi unici, identificati dalle diverse colonne presenti nella tabella. Nel contesto del progetto in esame, ci si concentra sull'utilizzo di alcune colonne specifiche, tra cui "Date Reported" (data di segnalazione dell'incidente), "Area ID" (identificativo dell'area geografica), "Area Name" (nome dell'area geografica), "Victim Age" (età della vittima), "Premise Description" (luogo dell'incidente), "Location" (posizione geografica) e "DR Number" (numero identificativo del sinistro).

Per migliorare la gestione e l'analisi dei dati, sono state effettuate una serie di trasformazioni. Innanzitutto, è stata creata la colonna "Data", con il formato modificato da MM/DD/YYYY, presente nella colonna "Date Reported" a "YYYY-MM-DD", per rendere più agevole l'interpretazione temporale delle informazioni.

In aggiunta, la colonna "Location" è stata scomposta in due nuove colonne separate: "Latitude" (latitudine) e "Longitude" (longitudine), entrambi dati di tipo double. Questo aggiornamento consente di lavorare con le coordinate geografiche associate a ciascun incidente.

Sono state utilizzate le seguenti istruzioni SQL:

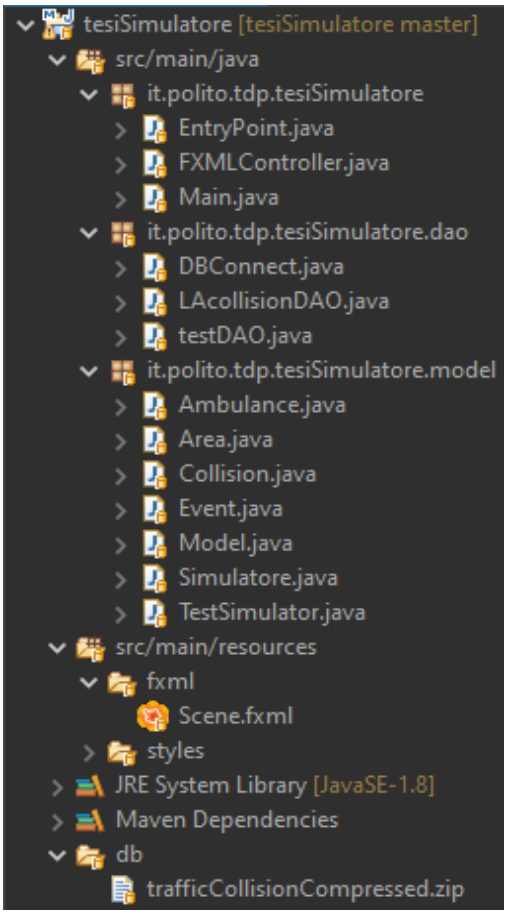
```sql
UPDATE la_traffic_collision
SET Data = DATE_FORMAT(STR_TO_DATE(DateReported, '%m/%d/%Y'), '%Y-%m-%d');
```

```sql
UPDATE la_traffic_collision
SET
    Latitude = CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(Location, ',', 1), '(', -1) AS DECIMAL(9, 6)),
    Longitude = CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(Location, ',', -1), ')', 1) AS DECIMAL(9, 6));
```

Capitolo 4: Descrizione strutture dati e algoritmi coinvolti

4.1 Strutture dati

Il software è stato programmato in Java, usando il pattern **MVC** (Model View Controller) e il pattern **DAO** (Data Access Object). Mediante il software SceneBuilder è stato possibile realizzare l'interfaccia grafica in JavaFX. Come consigliato dai pattern MVC e DAO, il progetto è costituito da tre package per separare il modello, l'accesso al database e l'interfaccia utente.



Il package *it.polito.tdp.tesiSimulatore* contiene la classe principale del progetto, Main, la classe EntryPoint, utilizzata per l'avvio del programma e la classe FXMLController, che collega il Model all'interfaccia grafica.

Il package *it.polito.tdp.tesiSimulatore.dao* contiene le classi necessarie per l'accesso ai dati presenti nel database.

Il package *it.polito.tdp.tesiSimulatore.model* ospita le classi fondamentali dell'applicazione dal punto di vista logico (Model e Simulatore), insieme alle classi degli oggetti ideati per il corretto funzionamento del programma. Questi oggetti includono Ambulance, Area, Collision, Event.

4.2 Algoritmi principali

4.2.1 package *it.polito.tdp.tesiSimulatore.dao*

All'interno della classe LAcollisionDAO sono presenti tre metodi:

- Il metodo *getVertici(Integer year)* estrae le aree all'interno della città di Los Angeles per l'anno specificato, acquisendo inoltre le coordinate geografiche di ciascuna zona presente nel database.
- Il metodo *getCollisionsYearAndMonthSpec(Integer year, Integer month)* estrae gli incidenti stradali avvenuti nella città di Los Angeles nell'anno e dal mese specificati, escludendo le collisioni con vittime di età sconosciuta.

Per poter utilizzare la classe *LatLng*, è stata inclusa la dipendenza nel file *pom.xml*.

```
<dependency>
	<groupId>com.javadocmd</groupId>
	<artifactId>simplelatlng</artifactId>
	<version>RELEASE</version>
</dependency>
```

```java
public List<Area> getVertici(Integer year) {

    String query = "SELECT DISTINCT la.`Area ID`, la.`Area Name`, AVG(la.Latitude) AS avgLat, AVG(la.Longitude) AS avgLon "
            + "FROM la_traffic_collision la "
            + "WHERE YEAR(la.`Data`) = ? "
            + "GROUP BY la.`Area ID`, la.`Area Name`";
    List<Area> result = new ArrayList<Area>();

    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(query);
        st.setInt(1, year);
        ResultSet rs = st.executeQuery();

        while (rs.next()) {
            LatLng coords = new LatLng(rs.getDouble("avgLat"),rs.getDouble("avgLon"));
            result.add(new Area (rs.getInt("Area ID"), rs.getString("Area Name"), coords) );
        }
        conn.close();
        return result;
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Errore connessione al database");
        throw new RuntimeException("Error Connection Database");
    }
}
```

```java
public List<Collision> getCollisionsYearAndMonthSpec(Integer year, Integer month) {

    String query = "SELECT l.`DR Number`, l.`Area ID`, l.`Area Name`, l.`Victim Age`, l.`Premise Description`, l.`Data`, l.Latitude, l.Longitude "
            + "FROM la_traffic_collision l "
            + "WHERE YEAR(l.`Data`) = ? AND MONTH(l.`Data`) >= ? AND l.`Victim Age` > 0 ";
    List<Collision> result = new ArrayList<Collision>();

    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(query);
        st.setInt(1, year);
        st.setInt(2, month);
        ResultSet rs = st.executeQuery();

        while (rs.next()) {
            result.add( new Collision (rs.getInt("DR Number"),
                                       rs.getInt("Area ID"), rs.getString("Area Name"),
                                       rs.getInt("Victim Age"), rs.getString("Premise Description"),
                                       rs.getDate("Data").toLocalDate(), rs.getDouble("Latitude"), rs.getDouble("Longitude")) );
        }
        conn.close();
        return result;
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Errore connessione al database");
        throw new RuntimeException("Error Connection Database");
    }
}
```

4.2.2 package *it.polito.tdp.tesiSimulatore.model*

All'interno della classe Model si trova il metodo *creaGrafo(Integer year)* che crea un grafo con i vertici che rappresentano i distretti della città di quell'anno considerato mentre il peso degli archi non orientati rappresenta la distanza in km tra un distretto ed un altro. In particolare le azioni svolte sono le seguenti:

1. *clearGraph()*: Questo metodo pulisce il grafo esistente, rimuovendo eventuali vertici ed archi precedentemente presenti.

2. Viene creato un nuovo grafo non orientato e pesato, utilizzando il framework JGraphT.

Mediante il metodo *dao.getVertici(year)*, vengono ottenuti i vertici del grafo, che corrispondono alle diverse aree della città nell'anno specificato.

Infine vengono calcolate le distanze in chilometri attraverso un doppio ciclo tra tutte le coppie di aree. Queste distanze vengono calcolate utilizzando il metodo *LatLngTool.distance* del framework LatLng, basato sulle coordinate geografiche delle aree. Le distanze rappresentano il peso degli archi nel grafo.

Il grafo è considerato completo poiché ogni vertice è direttamente collegato a tutti gli altri vertici. Per assicurare l'accuratezza nella costruzione del grafo, è possibile adoperare la formula seguente, la quale consente di calcolare il numero di archi presenti in un grafo completo, basandosi sul numero totale dei vertici n: $\left(\binom{n}{2}\right) = \frac{n(n-1)}{2}$

3. Viene istanziato un oggetto *DijkstraShortestPath* che permette di calcolare i percorsi più brevi nel grafo. Questo oggetto utilizzerà il grafo creato per calcolare i percorsi ottimali tra le aree.

```java
public void creaGrafo(Integer year) {
    clearGraph();
    //costruzione di un nuovo grafo
    this.grafo = new SimpleWeightedGraph<Area, DefaultWeightedEdge>(DefaultWeightedEdge.class);

    //assegnazione dei vertici
    List<Area> vertici = this.dao.getVertici(year);


    // Doppio ciclo per ottenere le distanze in km tra le varie aree della città
    for (int i = 0; i<vertici.size(); i++) {
        for (int j = i+1; j < vertici.size(); j ++) {
            Area area1 = vertici.get(i);
            Area area2 = vertici.get(j);
            double distanzaArea =  LatLngTool.distance(area1.getCoords(), area2.getCoords(), LengthUnit.KILOMETER);
            //aggiungo i vertici e i relativi archi al grafo
            Graphs.addEdgeWithVertices(this.grafo, area1, area2, distanzaArea);
        }
    }

    dijkstra = new DijkstraShortestPath<Area, DefaultWeightedEdge>(grafo);

    }
```

La parte algoritmica di simulazione si trova invece all'interno della classe Simulatore.

In primo luogo vengono stanziati i parametri essenziali per il corretto funzionamento della simulazione.

```java
public class Simulatore {

    //parametri di ingresso
    private Integer year;
    private Integer month;
    private int txtNumberHospital;
    private int txtNumberAmbulance;
    private double txtProbability;
    //parametri
    private LAcollisionDAO dao;
    private List<Area> areas;
    // lista di tutti gli incidenti avvenuti nell'anno e dal mese specificati
    private List<Collision> collisionsYearAndMonthSpecifiedList;
    // mappa on chiave: ID ambulanza, valore: classe Ambulance
    private Map<Integer, Ambulance> ambulancesMap;
    private List<Area> randomAreas;
    private DijkstraShortestPath<Area, DefaultWeightedEdge> dijkstra;
    // l'ambulanza lavora tutto il giorno
    private LocalTime startHour = LocalTime.of(0, 0); //rappresenterebbe mezzanotte 00:00
    private LocalTime stopHour = LocalTime.of(23, 59);
    private long dayHours;

    private Duration timeNoTraffic = Duration.of(5, ChronoUnit.MINUTES);
    private Duration timeTraffic = Duration.of(8, ChronoUnit.MINUTES);


    private Duration timeoutYellow = Duration.of(20, ChronoUnit.MINUTES);
    private Duration timeoutRed = Duration.of(10, ChronoUnit.MINUTES);

    private Duration healYellow = Duration.of(30, ChronoUnit.MINUTES);
    private Duration healRed = Duration.of(15, ChronoUnit.MINUTES);
    //variabili di uscita
    private int collisionsNumber;
    private int peopleSavedNumber;
    private Set<String> placeCollisionSet;
    //coda degli eventi
    private PriorityQueue<Event> queue;
```

Successivamente viene impostato il costruttore, che è invocato quando viene creato un nuovo oggetto di tipo Simulatore. In questa parte di codice vengono scelti casualmente i distretti che avranno un ospedale e vengono assegnate le ambulanze che ogni struttura ospedaliera possiede; queste informazioni vengono salvate in una mappa, chiamata *ambulancesMap* che ha come chiave il numero identificativo del veicolo di soccorso e come valore l'oggetto *Ambulance* (che contiene l'area dell' ospedale di appartenenza come attributo).

Inoltre viene impostato lo stato di tutti i veicoli di soccorso a "FREE" poiché ad inizio simulazione si ipotizza che tutte le ambulanze siano disponibili al pronto intervento.

```java
public Simulatore(Integer year, Integer month, int txtNumberHospital, int txtNumberAmbulance,
        double txtProbability, DijkstraShortestPath<Area, DefaultWeightedEdge> dijkstra) {

    this.year = year;
    this.month = month;
    this.txtNumberHospital = txtNumberHospital;
    this.txtNumberAmbulance = txtNumberAmbulance;
    this.txtProbability = txtProbability;
    this.dijkstra = dijkstra;
    this.dao = new LAcollisionDAO();
    this.collisionsYearAndMonthSpecifiedList = new ArrayList<>();
    this.peopleSavedNumber = 0;
    this.placeCollisionSet = new HashSet<>();
    this.ambulancesMap = new HashMap<>();

    this.dayHours = ChronoUnit.HOURS.between(startHour, stopHour);


    this.areas = this.dao.getVertici(year);


    // scelta casuali dei distretti che presentano un ospedale
    List<Integer> indices = new ArrayList<>();
    randomAreas = new ArrayList<>();

    int z = 0;
    while (z < this.txtNumberHospital) {
        int randomIndex = (int) (Math.random() * areas.size());
        if (!indices.contains(randomIndex)) {
            indices.add(randomIndex);
            randomAreas.add(areas.get(randomIndex));
            z++;
        }
    }
```

```java
// associo il numero di ambulanze totali ad una delle aree scelte casualmente nel ciclo while precedente
int j = 0;
for (int i=0; i< this.txtNumberAmbulance; i++){
    if (j == randomAreas.size())
        j = 0;
    Ambulance a = new Ambulance(i, randomAreas.get(j), null, State.FREE);
    ambulancesMap.put(i, a);
    j++;

}
```

Il metodo *popolaCoda()* poi riempie la coda degli eventi con le informazioni sugli incidenti. Questo processo considera le date, gli orari e le condizioni delle vittime, assegnando un tipo di evento in base all'età della persona coinvolta. Nello specifico, viene assegnato un orario casuale a ciascun incidente stradale, e le collisioni in cui la vittima ha un'età inferiore ai 18 anni o superiore ai 70 anni vengono classificate come eventi “RED”, altrimenti considerate come eventi “YELLOW”. Inoltre, vengono memorizzati tutti i luoghi in cui gli incidenti si sono verificati.

```java
public void popolaCoda() {

    this.queue = new PriorityQueue<Event>();

    Random rand = new Random();


    this.collisionsYearAndMonthSpecifiedList = dao.getCollisionsYearAndMonthSpec(year, month);

    this.collisionsNumber = collisionsYearAndMonthSpecifiedList.size();

    List<Event> events = new ArrayList<>();

    LocalTime ora = startHour;

    for (Collision c : collisionsYearAndMonthSpecifiedList) {
        Event e = null;

        // recupero coordinate geografiche della zona
        int areaIDCollision = c.getAreaID();
        LatLng coords = null;
        for (Area a : this.areas) {
            if (a.getAreaID() == areaIDCollision)
                coords = a.getCoords();
        }
        Area areaIncidente = new Area (c.getAreaID(), c.getAreaName(), coords);
        LocalDate date = c.getDate();
        // attribuisco ad ogni incidente un tempo casuale in cui avviene
        int minutes = rand.nextInt(60 * ((int) dayHours));
        ora = ora.plusMinutes(minutes);

        // aggiungo al set  il tipo di struttura o luogo in cui si è verificato l' incidente
        if (! (c.getLocation().isEmpty()) )
            this.placeCollisionSet.add(c.getLocation());
```

```java
        if (c.getVictimAge() < 18 || c.getVictimAge() > 70) {
            e = new Event(LocalDateTime.of(date, ora), EventType.RED, areaIncidente);
        }
        else {
            e = new Event(LocalDateTime.of(date, ora), EventType.YELLOW, areaIncidente);

        }

        events.add(e);
    }


    for (Event e : events) {
        this.queue.add(e);
    }



}
```

Successivamente, troviamo il metodo *processaEventi()*, il quale esegue un ciclo attraverso gli eventi presenti nella coda. Durante questo processo, si gestisce lo stato delle ambulanze. Nel dettaglio, il metodo stabilisce lo stato "FREE" per tutte le ambulanze che hanno un istante di tempo precedente a quello dell'incidente attualmente considerato. Questo approccio consente di gestire il passaggio delle diverse ambulanze della città dallo stato "OCCUPIED" allo stato "FREE" in modo efficace. Infatti se l'ambulanza i-esima ha un istante precedente all'istante della collisione che si sta considerando, ciò implica che essa sia rientrata in ospedale prima che l'incidente avesse luogo.

Inoltre, il metodo si occupa di trattare gli eventi specifici di tipo rosso o giallo, richiamando il metodo *handleCase()*.

```java
public void processaEventi() {



    while(!queue.isEmpty()) {

        Event e = queue.poll();

        LocalDateTime istantCollision = e.getIstant();

        for (Ambulance a : ambulancesMap.values()) {
            // se l'incidente è avvenuto dopo che l'ambulanza è tornata all'ospedale
            if (a.getIstant() != null && istantCollision.isAfter(a.getIstant())) {

                a.setState(State.FREE);
            }
        }

        switch (e.getType()) {

        case RED:
            handleCase(timeoutRed, healRed, e);
            break;

        case YELLOW:
            handleCase(timeoutYellow, healYellow, e);
            break;


        }

    }

}
```

Infine, si presenta il metodo *handleCase()*, il quale si occupa delle operazioni specifiche per la gestione di un incidente, a seconda del tipo di evento (rosso o giallo). Tale metodo gestisce lo stato delle ambulanze, calcola i tempi di percorrenza e tratta le vittime in base ai parametri forniti. Per essere più precisi:

1. Innanzitutto, verifica la disponibilità di ambulanze utilizzando il metodo *checkFreeAmbulances()*.
2. Nel caso in cui siano disponibili ambulanze nello stesso distretto dell'incidente (verificato tramite *AmbulancesSameDistrictOfCollision())*, ne viene selezionata una e il suo stato viene impostato su "OCCUPIED". Successivamente, viene generato un numero casuale r e, se questo è inferiore alla probabilità specificata (txtProbability), vengono aggiunti minuti, rispetto all'istante dell'incidente stradale, considerando l'andata e il ritorno all'ospedale con ingorgo stradale; in caso contrario, vengono aggiunti minuti senza traffico. Inoltre, viene sommato il tempo necessario per la cura della vittima e si incrementa il conteggio delle persone salvate. È importante notare che quando sono disponibili ambulanze nello stesso distretto dell'incidente, la vittima è sempre considerata in salvo.
3. Nel caso in cui non ci siano ambulanze disponibili nel medesimo distretto, si procede al calcolo dell'area più vicina con ambulanze libere rispetto all'area in cui è avvenuto l'incidente, utilizzando l'algoritmo di Dijkstra. Si itera sulle aree contenenti ambulanze libere, e mediante l'utilizzo di *dijkstra.getPath()*, si determinano le distanze più brevi. Tra le distanze minime ottenute tra i vari nodi di partenza e lo stesso nodo di destinazione, si seleziona il valore minimo (bestKm). Successivamente, si determina il tempo necessario per percorrere tale percorso, considerando la velocità dell'ambulanza (con o senza traffico) e la

distanza (bestKm). Nel caso in cui l'ambulanza raggiunga il luogo dell'incidente in tempo, il suo stato viene impostato su "OCCUPIED", l' attributo istant, della classe Ambulance viene aggiornato, aggiungendo il tempo necessario per il viaggio di andata e ritorno dall'ospedale e per la cura della vittima e si incrementa il conteggio delle persone salvate.

Se, invece, l'ambulanza non riesce a raggiungere il luogo dell'incidente in tempo, il suo stato viene comunque impostato su "OCCUPIED" e l'attributo istant viene aggiornato, ma senza considerare il tempo di cura per la vittima e senza aumentare il conteggio delle persone salvate. Infatti, anche in questo caso, l'ambulanza si dirige sul luogo della collisione per verificare effettivamente che la vittima sia deceduta.

```
private void handleCase(Duration timeout, Duration healingTime, Event eventCollision) {

    Area areaCollision = eventCollision.getAreaCollision();
    LocalDateTime timeStampCollision = eventCollision.getIstant();

    // verifico se vi sono ambulanze disponibili
    if (checkFreeAmbulances(ambulancesMap)) {

            // verifico se delle ambulanze disponibli, ci sono quelle che si trovano nello stesso distretto dell'incidente
            if (AmbulancesSameDistrictOfCollision(ambulancesMap, areaCollision) != null) {
                Ambulance a = AmbulancesSameDistrictOfCollision(ambulancesMap, areaCollision);
                a.setState(State.OCCUPIED);

                // se si verifica ingorgo
                double r = Math.random();
                if (r < txtProbability)
                    // aggiungo minuti di andata e ritorno dell'ambulanza dall'ospedale
                    a.setIstant(timeStampCollision.plus(timeTraffic.multipliedBy(2)));
                else
                    a.setIstant(timeStampCollision.plus(timeNoTraffic.multipliedBy(2)));

                // aggiungo minuti per curare vittima
                a.getIstant().plus(healingTime);
                this.peopleSavedNumber++;
            }
```

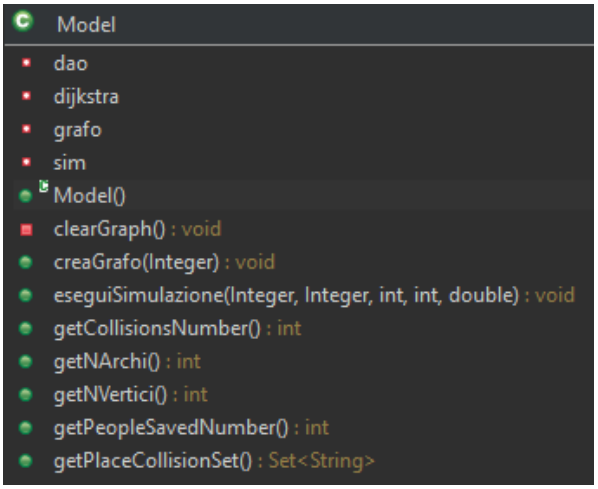
```
// caso in cui non vi sono ambulanze nello stesso distretto della collisione
else {

    // prendo, attraverso dijkstra, i km minori ciclando su tutte le aree in cui
    // risultano disponibili delle ambulanze
    // tra i km minimi ottenuti tra le diversi nodi di origine e il medesimo nodo destinazione
    // prendo il valore minimo
    Set<Area> areaWithFreeAmbulancesSet = getAreaWithFreeAmbulances(ambulancesMap);

    double bestKm = 9999999999.0;
    Area bestArea = null;


    for (Area area : areaWithFreeAmbulancesSet) {
        double kmBetweenSourceTarget = dijkstra.getPath(area, areaCollision).getWeight();
        if (kmBetweenSourceTarget < bestKm) {
            bestKm = kmBetweenSourceTarget;
            bestArea = area;
        }
    }

    Ambulance bestAmbulance = getBestAmbulanceFree(bestArea, ambulancesMap);


    int pathTime = 0;


    double r = Math.random();
```

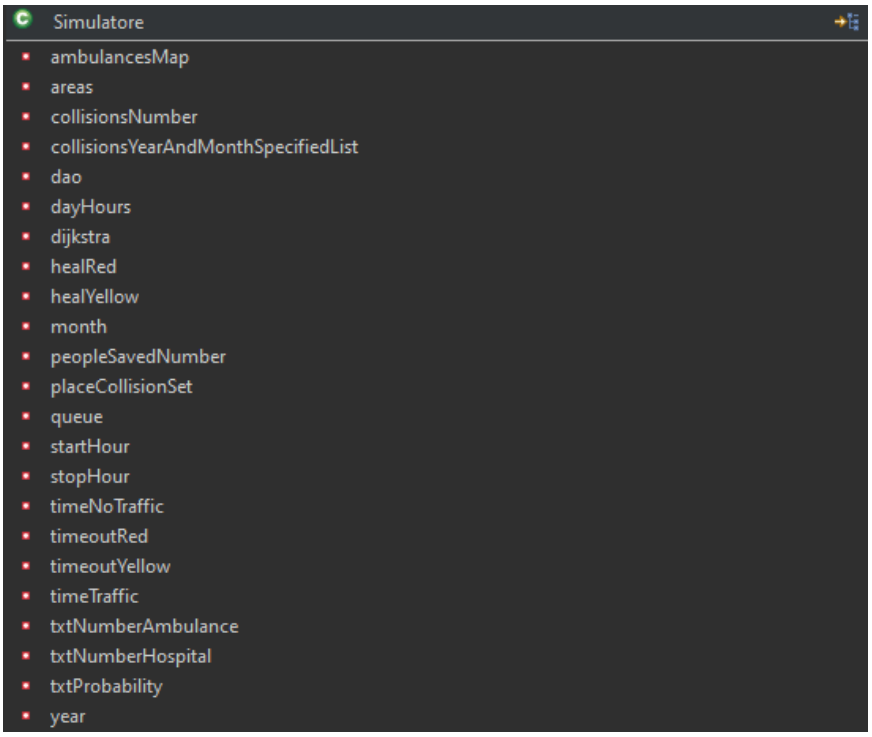
```
            // se si verifica ingorgo
            if (r < txtProbability) {
                // velocità ambulanza con traffico pari a 40 km/h
                pathTime = (int) ((bestKm/40)*60);
            }
            else {
                // velocità ambulanza no traffico pari a 70 km/h
                pathTime =  (int) ((bestKm/70)*60);
            }


            // se l'ambulanza arriva in tempo sul luogo dell'incidente

            Duration pathTimeMinutesConverted = Duration.of(pathTime, ChronoUnit.MINUTES);

            if (pathTimeMinutesConverted.compareTo(timeout) <=0 ) {

                bestAmbulance.setState(State.OCCUPIED);
                // minutaggio minutaggio arrivo sul luogo + cura + ritorno all'ospedale
                bestAmbulance.setIstant(timeStampCollision.plus(pathTimeMinutesConverted.multipliedBy(2)));
                bestAmbulance.getIstant().plus(healingTime);
                this.peopleSavedNumber++;

            }

            // se l'ambulanza NON arriva in tempo sul luogo dell'incidente
            else {
                // verifica che effetivamente la vittima sia deceduta recandosi sul luogo ma non vi è il te
                bestAmbulance.setState(State.OCCUPIED);
                // minutaggio = arrivo sul luogo + ritorno all'ospedale
                bestAmbulance.setIstant(timeStampCollision.plus(pathTimeMinutesConverted.multipliedBy(2)));

            }

        }
}
```

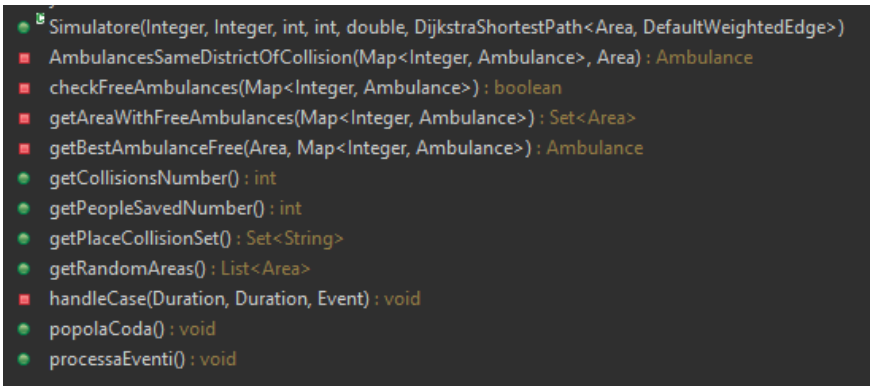
```
private Ambulance getBestAmbulanceFree(Area bestArea, Map<Integer, Ambulance> ambulancesMap) {

    Ambulance result = null;
    for (Ambulance a : ambulancesMap.values()) {
        if (a.getState().equals(State.FREE) && a.getArea().equals(bestArea))
            result = a;
    }
    return result;

}



// metodo che prende solamente le aeree con ambulanze libere
private Set<Area> getAreaWithFreeAmbulances(Map<Integer, Ambulance> ambulancesMap) {

    Set<Area> result = new HashSet<>();
    for (Ambulance a : ambulancesMap.values()) {
        if (a.getState().equals(State.FREE))
            result.add(a.getArea());

    }
    return result;
}
```

```
private Ambulance AmbulancesSameDistrictOfCollision(Map<Integer, Ambulance> map, Area areaCollison) {
    // TODO Auto-generated method stub
    for (Ambulance a : map.values())
        if (a.getArea().equals(areaCollison))
            return a;

    return null;
}



private boolean checkFreeAmbulances(Map<Integer, Ambulance> map) {
    // TODO Auto-generated method stub
    for (Ambulance a : map.values())
        if (a.getState().equals(State.FREE))
            return true;

    return false;
}
```

Capitolo 5: Diagramma delle classi delle parti principali







Capitolo 6: Esempi di utilizzo dell'applicazione

6.1 Videata dell'applicazione

All'avviarsi dell'applicazione, si offre la possibilità di interagire con due componenti principali: un menù a tendina che consente di selezionare l'anno degli eventi da considerare nel database e un pulsante per generare un grafo, chiamato "Crea città". Questo grafo è costituito dai distretti della città, rappresentati come vertici, e dagli archi pesati che indicano la distanza in chilometri tra ciascun distretto.

Una volta creato il grafo, gli altri elementi dell'interfaccia grafica vengono abilitati. Questi elementi sono dedicati alla configurazione e all'avvio della simulazione. Un ulteriore menù a discesa permette di selezionare il mese dal quale partire per estrarre gli eventi per la simulazione. Inoltre, tre campi di testo consentono di inserire valori di input per il numero totale di ospedali, il numero totale di ambulanze che dovrebbero essere presenti nella città di Los Angeles e la probabilità di congestione in caso di incidenti stradali durante la simulazione. Infine, è presente un pulsante che, una volta completata la compilazione dei campi precedenti, avvia la simulazione, chiamato "Simula".

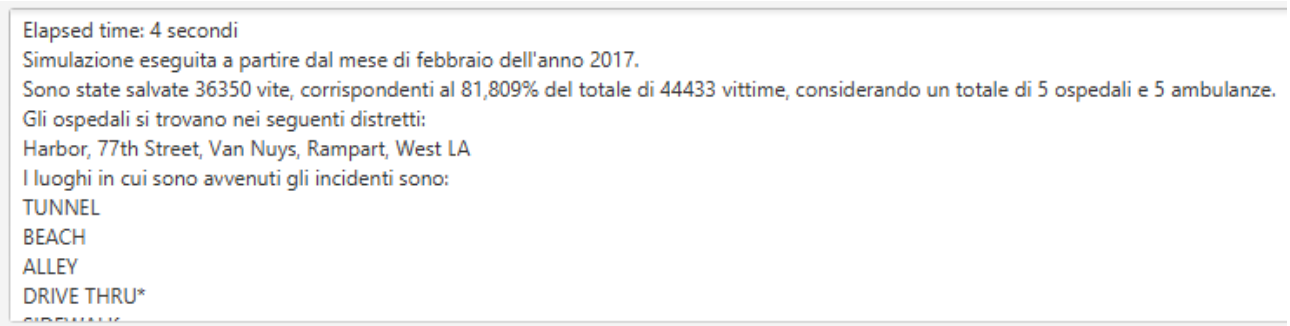
I risultati generati dalla simulazione vengono visualizzati nell'area di testo situata sotto gli elementi descritti in precedenza.

È possibile guardare un video dimostrativo sul funzionamento del programma al seguente link: https://youtu.be/HFDKYQJDHmY

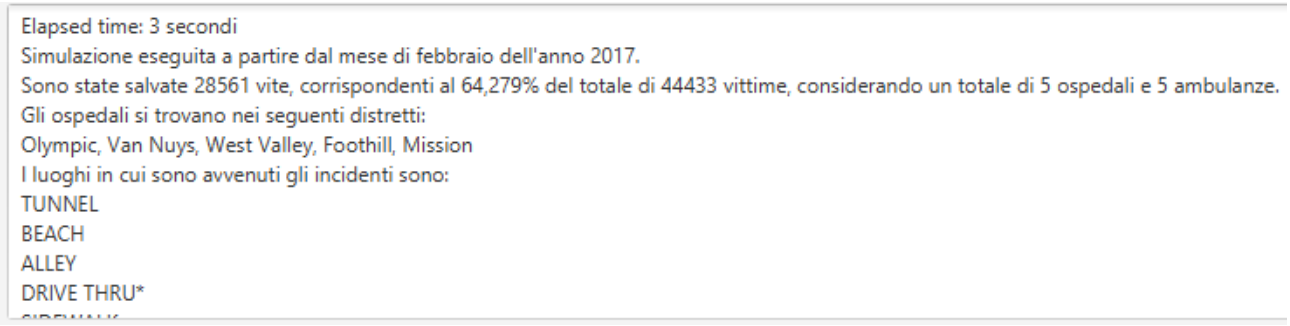
Capitolo 7: Risultati sperimentali ottenuti

Di seguito vengono presentati alcuni dei risultati conseguiti nel corso delle simulazioni.

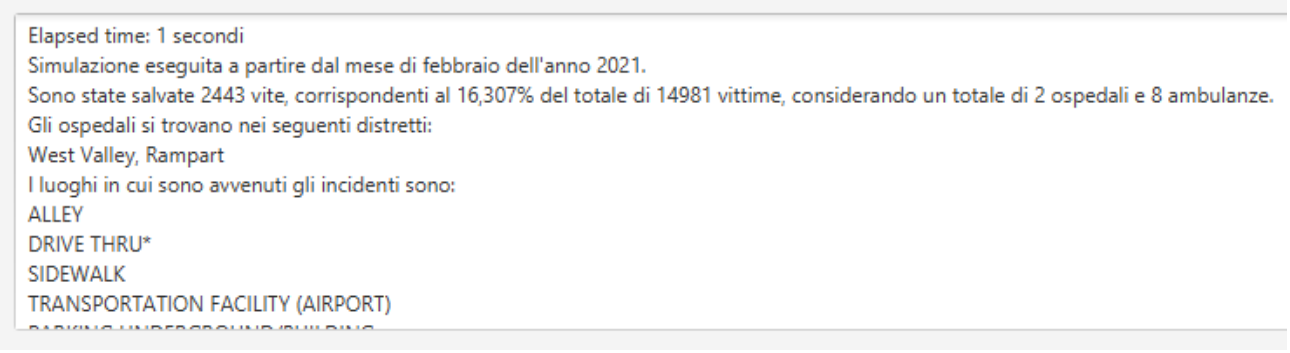
Selezionando come anno *2017*, come mese *febbraio*, come numero di ospedali *5*, come numero di ambulanze *5* e come probabilità di ingorgo *0.5*, i risultati ottenuti sono:



Lasciando invariati i parametri di ingresso impostati dall'utente, i risultati ottenuti sono:



Selezionando come anno *2021*, come mese *febbraio*, come numero di ospedali *2*, come numero di ambulanze *8* e come probabilità di ingorgo *0.9*, i risultati ottenuti sono:



Il tempo di elaborazione delle singole simulazioni varia al variare dei dati immessi dall'utente. Si noti infatti come al diminuire del numero di ospedali diminuisca il tempo di simulazione.

Capitolo 8: Considerazioni finali

8.1 Valutazione dei risultati ottenuti

Le simulazioni producono esiti influenzati dalle variabili di input, dagli istanti temporali in cui gli incidenti si verificano e dai distretti in cui gli ospedali sono ubicati; questi ultimi due elementi selezionati casualmente. Questa casuale selezione potrebbe generare esiti divergenti, anche utilizzando identici dati di input. Questo fenomeno è evidenziato chiaramente dalle prime due figure del capitolo precedente, in cui si osserva una differenza di circa il 17% nel numero di vittime salvate tra le due immagini.

È da notare invece che nell'ultima immagine si registra un minor numero totale di vittime, corrispondenti agli incidenti (poiché ogni vittima rappresenta un incidente), nonostante il mese di partenza della simulazione sia il medesimo delle precedenti immagini. Questo declino può essere facilmente attribuito al periodo di diffusione del Covid. In aggiunta, il numero di vittime salvate risulta inferiore rispetto alle immagini precedenti, ciò è attribuibile alla presenza di un minor numero di ospedali operanti nel territorio statunitense, anche aggravato dalla maggiore probabilità di congestione del traffico.

Inoltre, è importante sottolineare che spesso si registra un'alta percentuale di vittime salvate, nonostante la presenza limitata di ospedali e/o ambulanze. Questo fenomeno può essere attribuito al fatto casuale che le aree in cui si sono verificati gli incidenti nell'ambito della simulazione coincidono con i distretti in cui sono ubicati gli ospedali. Infatti, secondo le impostazioni della simulazione, ogni volta che un incidente si verifica nell'area di un ospedale, la vittima viene automaticamente considerata come salvata. In aggiunta, il fenomeno fa sì che la simulazione sia meno sensibile a variazioni del numero di ambulanze, rispetto a quanto avviene con le variazioni del numero di ospedali.

Nel complesso i risultati ottenuti dall’esecuzione del programma forniscono le informazioni necessarie per definire il numero ottimale complessivo degli ospedali da costruire nella città di Los Angeles, identificando anche la posizione strategica dei distretti in cui collocare tali strutture e stabilendo il totale delle ambulanze necessarie per il territorio urbano.

8.2 Conclusioni

Gli scopi principali del programma possono essere considerati soddisfatti: il software è funzionante e potrebbe essere impiegato per gli scopi delineati.

Possibili soluzioni più dettagliate potrebbero essere introdotte attraverso un ulteriore sviluppo dell’applicazione. Un esempio potrebbe essere consentire la simulazione di più ospedali all'interno dello stesso distretto, oppure offrire la possibilità di configurare dei minuti supplementari tra il verificarsi dell'incidente stradale e la ricezione dell'evento da parte dell'ospedale. Inoltre, per aumentare il realismo, si potrebbe permettere la personalizzazione dei parametri di cura del paziente e delle velocità dell'ambulanza, oppure generarli casualmente con valori plausibili. Anche il tempo di percorrenza di andata e ritorno dal luogo dell'incidente potrebbe variare, caratteristica che invece è mantenuta costante in questa simulazione.

Questa relazione tecnica è rilasciata con licenza Creative Commons BY-NC-SA.

Tu sei libero di:

- Condividere - riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato
- Modificare - remixare, trasformare il materiale e basarti su di esso per le tue opere

Alle seguenti condizioni:

- Attribuzione - Devi riconoscere una menzione di paternità adeguata, fornire un link alla licenza e indicare se sono state effettuate delle modifiche. Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale.
- Non Commerciale - Non puoi utilizzare il materiale per scopi commerciali.
- Stessa Licenza - Se remixi, trasformi il materiale o ti basi su di esso, devi distribuire i tuoi contributi con la stessa licenza del materiale originario.

Per visualizzare una copia di questa licenza, visitare : https://creativecommons.org/licenses/by-nc-sa/4.0/