



**Politecnico
di Torino**

Politecnico

di Torino

Corso di Laurea
A.a. 2021/2022
Sessione di Laurea Ottobre 2020

Simulazione e Gestione di Flussi Migratori

Relatore:
Fulvio Corno

Candidato:
Destiny Jarymaya Okpekpe

Sommario

1 Titolo e Riferimenti

1.1	Studente proponente.....	3
1.2	Titolo della proposta.....	3

2	Proposta di progetto integrale.....	3
2.1	Descrizione del problema proposto.....	3
2.2	Descrizione della rilevanza gestionale del problema.....	3
2.3	Descrizione dei data-set per la valutazione.....	3
2.4	Descrizione preliminare degli algoritmi coinvolti.....	4
2.5	Descrizione preliminare delle funzionalità previste per l'applicazione software.....	4
3	Descrizione dettagliata del problema.....	5
3.1	Descrizione.....	5
4	Descrizione del data-set utilizzato.....	6
4.1	Descrizione generale.....	6
4.2	Data-set country_population.....	6
4.3	Data-set country_borders.....	7
4.4	Criticità nella compatibilità tra data-set.....	8
5	Descrizione di alto livello delle strutture dati e degli algoritmi utilizzati.....	9
5.1	Strutture dati.....	9
5.2	Algoritmi.....	10
6	Diagramma classi.....	15
6.1	Elenco classi.....	15
6.2	Diagramma classi principali.....	16
7	Video dimostrativo.....	17
7.1	Schermate dell'applicazione.....	17
7.2	Video tutorial Youtube.....	18
8	Tabelle con risultati dimostrativi.....	19
8.1	Caso 1.....	19
8.2	Caso 2.....	20
8.3	Caso 3.....	21
9	Conclusioni.....	22
9.1	Punti di forza.....	22
9.2	Punti critici.....	22
9.3	Conclusioni personali.....	22
9.4	Licenze.....	23

1 Titolo e Riferimenti

1.1 Studente proponente

Destiny Jarymaya Okpekpe, matricola s269631 del corso di Ingegneria gestionale L8 presso il Politecnico di Torino.

1.2 Titolo della proposta

Simulazione e gestione di flussi migratori.

2 Proposta di progetto integrale

2.1 Descrizione del problema proposto

In un mondo occidentale sempre più meta di destinazione per la ricerca di un futuro migliore da parte di migranti di stati più in difficoltà, diventa fondamentale essere in grado di gestire i flussi migratori nel migliore dei modi. Per far ciò è importante avere idea di come questi fenomeni evolvano e siano influenzati da fattori come situazione economica, guerre, carestie, epidemie, disastri naturali e ambientali (facendo un confronto con situazioni simili del passato).

2.2 Descrizione della rilevanza gestionale del problema

Un applicativo di questo tipo può supportare le decisioni di chi gestisce il fenomeno, in questo caso un governante o un politico. Esso, in base alle informazioni ottenute, potrà di conseguenza capire quali politiche applicare alla situazione, trovando un bilancio tra spese da sostenere, costi in termini umani e impopolarità delle scelte.

2.3 Descrizione dei data-set per la valutazione

- **Kaggle:** data-set sugli abitanti dei vari stati (in base agli eventi una percentuale di questi migrerà).

3

- **Internet:** per avere altre informazioni come "indice di attrattività di uno stato destinazione", costo gestione migranti, tempi di spostamento di migranti in base alla rotta ecc...

2.4 Descrizione preliminare degli algoritmi coinvolti

Grafi: Disegnare la mappa degli stati e cercare un percorso per raggiungere stati destinazione (Europa).

Simulazione ad eventi: Simulazione dell'andamento migratorio sotto specifici eventi (guerre, carestie, disastri ambientali ecc...).

Ottimizzazione: Una volta simulati i vari eventi, ricerca della soluzione che massimizzi gli interessi dell'utente (spesa minima, popolarità massima, via di mezzo ecc..).

2.5 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'interfaccia permetterà all'utente di:

Selezionare lo stato di interesse di gestione (europeo).

Selezionare vari stati esteri, il tipo di evento che li coinvolge (crisi economica, guerra, disastro ambientale, carestia, pandemia ecc), il livello di gravità del fenomeno (intero da 1 a 3).

Selezionare l'approccio da utilizzare, ovvero se preferire politiche più bloccanti o più permissive.

Questo permetterà di impostare il mondo esterno a cui il simulatore dovrà fare affidamento e ottenere i valori di output di spesa e popolarità.

3 Descrizione dettagliata del problema

3.1 Descrizione

Essendo l'Italia il paese Europeo che più si affaccia sul Mediterraneo e che più è vicino alle coste del continente africano, non ci deve stupire il fatto che

sia uno degli stati europei che più di tutti deve gestire i flussi migratori. Diventa però molto difficile riuscire a farsi un'idea di quale sia il miglior modo per farlo: abbiamo infatti moltissime variabili da tenere in conto.

- Innanzitutto il numero di persone che dovrà essere accolto: questo dipenderà da una miriade di fattori come equilibri geo-politici, guerre, eventi catastrofici imprevedibili, carestie, epidemie ecc...
- Bisogna poi vedere in che direzione andranno questi migranti, se verso altri stati africani, con l'obiettivo di stanziarsi oppure se verso stati europei in cerca di un futuro migliore.
- Infine bisogna fare i conti con le risorse dello stato accogliente: sia in termini di disponibilità economiche che possono essere destinate alla causa, sia alle risorse umane: quante vite siamo disposti a far perdere a queste persone? Siamo disposti ad utilizzare le nostre risorse per degli stranieri? Una volta accolti saremo in grado di costruire un sistema in grado di integrarli?

Tutti questi punti rendono la gestione dei flussi migratori estremamente complessa e divisiva, non a caso è uno degli argomenti che più interessa ad esempio ai partiti politici, in un senso o nell'altro.

Ho notato però come se è vero che è difficile stabilire a priori cosa sia meglio fare, si potrebbe partire utilizzando dei dati solidi: avere cioè ben presente, dati certi input, quali output possiamo aspettarci. In quest'ottica è stato pensato questo software, per quanto semplificato.

4 Descrizione del data-set utilizzato

4.1 Descrizione generale

Per costruire questa applicazione è stato fondamentale avere informazioni sugli stati, sui loro confini e sulle loro caratteristiche, come il numero di

abitanti. Questo proprio per permettere la creazione del grafo su cui la simulazione avrebbe lavorato e per permettere il calcolo di morti, migranti e spesa, in base alle caratteristiche della nazione presa in esame. A questo scopo sono stati utilizzati due data-set.

4.2 Data-set country_population

Questo data-set conteneva informazioni sugli stati del mondo e su alcune loro caratteristiche, in particolare sulla popolazione.

Schema

Country (chiave primaria): Nome dello stato

Region: Zona geografica dello stato (es. EASTERN EUROPE, NORTHERN AFRICA)

Population: Il numero di abitanti di quello stato.

Il data-set è gratuito e a libero utilizzo sul sito Kaggle al link <https://www.kaggle.com/datasets/fernando/countries-of-the-world> (Il data-set è stato modificato per lo stretto utilizzo di questa applicazione).

country_code	country_name	country_border_code	country_border_name
LY	Libya	DZ	Algeria
LY	Libya	TD	Chad
LY	Libya	EG	Egypt
LY	Libya	NE	Niger
LY	Libya	SD	Sudan
LY	Libya	TN	Tunisia
MA	Morocco	DZ	Algeria
MA	Morocco	ES	Spain
MA	Morocco	EH	Western Sahara
MC	Monaco	FR	France

4.3 Data-set country_borders

Questo data-set conteneva informazioni sugli stati ed i loro confini, correlando ad uno ad uno ogni stato con un suo stato confinante.

Schema

Country_code(Chiave primaria combinata): Codice del primo stato.

Country_name: Nome del primo stato.

Country_border_code(Chiave primaria combinata): Codice del secondo stato.

Country_border_name: Nome del secondo stato.

Il dat-set è gratuito e a libero utilizzo dal sito GeoDataSource al link <https://www.geodatasource.com/addon/country-borders>

Country	Region	Population
Afghanistan	ASIA (EX. NEAR EAST)	31,056,997
Albania	EASTERN EUROPE	3,581,655
Algeria	NORTHERN AFRICA	32,930,091
American Samoa	OCEANIA	57,794
Andorra	WESTERN EUROPE	71,201
Angola	SUB-SAHARAN AFRICA	12,127,071
Anguilla	LATIN AMER. & CARIB	13,477
Antigua & Barbuda	LATIN AMER. & CARIB	69,108
Argentina	LATIN AMER. & CARIB	39,921,833
Armenia	C.W. OF IND. STATES	2,976,372
Aruba	LATIN AMER. & CARIB	71,891
Australia	OCEANIA	20,264,082
Austria	WESTERN EUROPE	8,192,880
Azerbaijan	C.W. OF IND. STATES	7,961,619
Bahamas, The	LATIN AMER. & CARIB	303,770
Bahrain	NEAR EAST	698,585
Bangladesh	ASIA (EX. NEAR EAST)	147,365,352
Barbados	LATIN AMER. & CARIB	279,912
Belarus	C.W. OF IND. STATES	10,293,011
Belgium	WESTERN EUROPE	10,379,067
Belize	LATIN AMER. & CARIB	287,730
Benin	SUB-SAHARAN AFRICA	7,862,944
Bermuda	NORTHERN AMERICA	65,773
Bhutan	ASIA (EX. NEAR EAST)	2,279,723
Bolivia	LATIN AMER. & CARIB	8,989,046
Bosnia & Herzegovina	EASTERN EUROPE	4,498,976
Botswana	SUB-SAHARAN AFRICA	1,639,833

4.4 Criticità nella compatibilità tra data-set

Essendo country_borders e country_population due data-set di diversi enti, ci sono stati alcuni problemi di compatibilità: ad esempio in uno gli Stati Uniti erano segnati come “United States” mentre nell’altro “United States of America”. Questo dava problemi nelle operazioni di join, perciò ho dovuto

manualmente collegare alcuni stati, per impedire casi di parti non connesse nel grafo creato.

5 Descrizione di alto livello delle strutture dati e degli algoritmi utilizzati

5.1 Strutture dati

L'applicazione è stata costruita in linguaggio Java, utilizzando il pattern MVC (Model View Controller) ed il pattern DAO (Data Access Object), perciò queste tre parti sono state conseguentemente divise. Per l'interfaccia è stato invece utilizzato JavaFX con il software SceneBuilder.

Abbiamo tre principali package che contengono la nostra applicazione.

- Il package **it.polito.tdp.gestioneFlussiMigratori** che contiene EntryPoint, il Main che farà avviare la nostra applicazione ed il Controller, che si occuperà di visualizzare le informazioni di nostro interesse.
- Il package **it.polito.tdp.gestioneFlussiMigratori.db** che contiene il DBConnect che ci permette di collegarci al nostro database con le nostre credenziali e GestioneFlussiMigratori, il DAO che ci permetterà di ottenere le informazioni che ci servono dal database e convertirle in classi Java che utilizzeremo per i nostri algoritmi.
- Il package **it.polito.tdp.gestioneFlussiMigratori.model** che contiene Il Model che gestirà tutte le classi del package: Country che dà informazioni sullo stato, sul numero di migranti e di morti che conta, Adiacenza che è una classe di supporto per creare il grafo collegando tra loro stati adiacenti, Il simulatore che gestirà gli eventi di nostro interesse (ovvero lo spostamento dei migranti sulla mappa), la classe Evento, la classe Comparatore per ordinare ed infine un TestModel per testare il Model prima di collegarlo al controller.

```

  ▾ 📁 > GestioneFlussiMigratori [repository master]
    ▾ 📁 > src/main/java
      ▾ 📁 > it.polito.tdp.gestioneFlussiMigratori
        > 📄 EntryPoint.java
        > 📄 FXMMLController.java
        > 📄 Main.java
      ▾ 📁 > it.polito.tdp.gestioneFlussiMigratori.db
        > 📄 DBConnect.java
        > 📄 GestioneFlussiMigratoriDAO.java
      ▾ 📁 > it.polito.tdp.gestioneFlussiMigratori.model
        > 📄 Adiacenza.java
        > 📄 ComparatorePerMorti.java
        > 📄 Country.java
        > 📄 Evento.java
        > 📄 > Model.java
        > 📄 > Simulatore.java
        > 📄 TestModel.java

```

5.2 Algoritmi

- **Partiamo dalla creazione del mondo:**

Per creare il grafo su cui lavorerà la simulazione dovremo prima ottenere dal database le informazioni sugli stati e sui confini, per questo utilizziamo i metodi del dao. Utilizzando una idMap riusciamo a non dover ogni volta riempire una nuova lista, ma ci occupiamo di aggiungere solo eventuali stati nuovi (molto utile in caso di database dinamici).

```

public List<Country> getAllAfricanCountries(Map<String, Country> idMap) {
    String sql = "SELECT DISTINCT * "
        + "FROM country_population AS c "
        + "WHERE c.Region LIKE '%AFRICA%' "
        + "ORDER BY c.Country";

    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(sql);
        ResultSet rs = st.executeQuery();
        List<Country> list = new LinkedList<Country>();
        while (rs.next()) {
            if (idMap.get(rs.getString("c.Country")) == null) {
                Country c = new Country(rs.getString("c.Country").trim(), rs.getString("c.Region"), rs.getInt("c.Population"), 0, 0);
                idMap.put(c.getCountry(), c);
                list.add(c);
            } else {
                list.add(idMap.get(rs.getString("c.Country")));
            }
        }
        conn.close();
        return list;
    }
}

```

```

public List<Country> getAllEuropeanCountries(Map<String, Country> idMap) {

    String sql = "SELECT DISTINCT * "
        + "FROM country_population AS c "
        + "WHERE c.Region LIKE '%EUROPE%' "
        + "ORDER BY c.Country";

    try {
        Connection conn = DBConnect.getConnection();

        PreparedStatement st = conn.prepareStatement(sql);

        ResultSet rs = st.executeQuery();

        List<Country> list = new LinkedList<Country>();

        while (rs.next()) {

            if (idMap.get(rs.getString("c.Country")) == null) {

                Country c = new Country(rs.getString("c.Country").trim(), rs.getString("c.Region"), rs.getInt("c.Population"), 0, 0);
                idMap.put(c.getCountry(), c);
                list.add(c);
            } else
                list.add(idMap.get(rs.getString("c.Country")));
        }

        conn.close();

        return list;
    }

}

public List<Adiacenza> getAdiacenza() {

    String sql = "SELECT DISTINCT c.Country, b.country_border_name "
        + "FROM country_population AS c, country_borders AS b "
        + "WHERE c.Country=b.country_name AND (c.Region LIKE '%AFRICA%' OR c.Region LIKE '%EUROPE%') AND c.Country < b.country_ "
        + "ORDER BY c.Country";

    List<Adiacenza> result = new ArrayList<>();

    try {
        Connection conn = DBConnect.getConnection();

        PreparedStatement st = conn.prepareStatement(sql);

        ResultSet rs = st.executeQuery();

        while (rs.next()) {
            result.add(new Adiacenza(rs.getString("c.Country").trim(), rs.getString("b.country_border_name").trim()));
        }

        conn.close();
        return result;
    }

}

```

A questo punto possiamo creare con queste informazioni il nostro grafo con il metodo del model.

```

public void creaGrafo() {

    this.grafo = new SimpleGraph<>(DefaultEdge.class);

    //Vertici
    this.dao.getAllAfricanCountries(this.idMap);
    this.dao.getAllEuropeanCountries(this.idMap);
    Graphs.addAllVertices(this.grafo, this.idMap.values());

    //Archi
    List<Adiacenza> archi = this.dao.getAdiacenza();
    for(Adiacenza a: archi) {
        if(this.idMap.get(a.getCountry1()) != null && this.idMap.get(a.getCountry2()) != null) {
            this.grafo.addEdge(this.idMap.get(a.getCountry1()), this.idMap.get(a.getCountry2()));
        }
    }
    this.grafo.addEdge(this.idMap.get("South Africa"), this.idMap.get("Nigeria"));
    this.grafo.addEdge(this.idMap.get("Congo, Dem. Rep."), this.idMap.get("Nigeria"));
    this.grafo.addEdge(this.idMap.get("Central African Rep."), this.idMap.get("Nigeria"));
    this.grafo.addEdge(this.idMap.get("Cote d'Ivoire"), this.idMap.get("Nigeria"));

    System.out.println("Grafo creato");
    System.out.println("# Vertici: " + this.grafo.vertexSet().size());
    System.out.println("# Archi: " + this.grafo.edgeSet().size());
}

```

- **Facciamo la nostra simulazione:**

Presi dal controller gli input su nazioni, eventi e gravità, facciamo partire una simulazione per ogni stato, inserendo ogni volta degli input di inizializzazione diversi ma utilizzando lo stesso simulatore

```
public void Simula(Country n1, Country n2, Country n3, String e1, String e2, String e3, Integer g1, Integer g2, Integer g3) {
    for(Country c: this.idMap.values()) {
        c.setMorti(0);
        c.setMigranti(0);
    }
    Simulatore sim = new Simulatore(this.grafo, this.idMap);
    if(n1!=null) {
        sim.Init(e1, n1, g1);
        sim.run();
    }
    if(n2!=null) {
        sim.Init(e2, n2, g2);
        sim.run();
    }
    if(n3!=null) {
        sim.Init(e3, n3, g3);
        sim.run();
    }
    this.nNordAfrica = sim.getNTotMigrantiNordAfrica();
    this.nTotMorti = sim.getNTotMorti();
}
```

Il simulatore salva le informazioni di input da caricare, in particolare calcola un “fattore di pericolosità” per capire quale porzione di popolazione prendere dallo stato di partenza.

```
private double creaFattore() {
    double fattore = 0.0;
    switch(this.tipo) {
        case "Epidemia":
            fattore = 0.007;
            break;
        case "Guerra":
            fattore = 0.01;
            break;
        case "Carestia":
            fattore = 0.007;
            break;
        case "Disastro Ambientale":
            fattore = 0.007;
            break;
        case "Persecuzione":
            fattore = 0.005;
            break;
        case "Povertà":
            fattore = 0.005;
            break;
        default:
            fattore = 0;
            break;
    }
    return fattore;
}
```

A questo punto ad ogni passo, uno stato perde un terzo dei propri migranti e li aggiunge ai propri morti. I restanti migranti sono invece pronti ad emigrare verso gli stati adiacenti. Vengono equamente divisi per il numero di stati confinanti e aggiunti ai migranti di quello stato. A questo punto i migrante dello stato da cui siamo partiti ritorna a zero. La simulazione finisce se i migranti giungono in unno stato del nord Africa

```
private void processEvento(Evento e) {
    if(!e.getCountry().getCountry().trim().equals("Algeria") && !e.getCountry().getCountry().equals("Libya") && !e.getCountry().
        int partono = (int)(e.getCountry().getMigranti()/3)*2;
        int morti = e.getCountry().getMigranti()-partono;
        this.mappa.get(e.getCountry().getCountry()).setMorti(this.mappa.get(e.getCountry().getCountry()).getMorti() + morti);
        int statiConfinanti = this.mondo.degreeOf(e.getCountry());
        if(statiConfinanti!=0) {
            int migranti = (int) (partono/statiConfinanti);
            if(migranti>=1) {
                for(Country c: Graphs.neighborListOf(this.mondo, e.getCountry())) {
                    this.mappa.get(c.getCountry()).setMigranti(migranti+this.mappa.get(c.getCountry()).getMigranti());
                    c.setMigranti(c.getMigranti()+migranti);
                    this.queue.add(new Evento(e.getTime()+1, c));
                }
            }
            e.getCountry().setMigranti(0);
        }
    }
}
```

- **Carichiamo i dati nel nostro controller**
- **Gestiamo il flusso migratori**

Una volta ottenuti i migranti giunti nel nord Africa, la nazione ospitante e l'approcci da utilizzare, aggiungiamo un po' di casualità all'approccio (aggiungendo o togliendo dal valore preso, un numero tra -2 e +2). E calcoliamo quanti migranti moriranno, quanti sopravvivranno, la spesa prevista (questa è stata presa da Internet, si stima che per gestire 120mila migranti, lo stato utilizzi intorno ai 1,5 miliardi all'anno, perciò circa 10000-100000 volte il numero di migranti) e il rapporto con la popolazione. Sono stati aggiunti dei controlli sulla positività

```

public List<Integer> gestisci(Country c, double approccio, int migranti){
    double segno1 = Math.random();
    double segno2 = Math.random();
    double v1 = Math.random();
    double v2 = Math.random();

    if(segno1<0.5)
        segno1 = 1;
    else
        segno1 = -1;

    if(segno2<0.5)
        segno2 = 1;
    else
        segno2 = -1;

    double approccioVero = approccio+(segno1*v1)+(segno2*v2);

    List<Integer> lista = new ArrayList<Integer>();
    lista.add(0, (int)((approccioVero/10)*migranti));
    lista.add(1, migranti - (lista.get(0)));
    if(lista.get(1)<0) {
        lista.add(1, ((lista.get(1))*(-1)));
    }
    lista.add(2, 100000*lista.get(1));
    if(lista.get(2)<0) {
        lista.add(2, (lista.get(2))*(-1));
    }
    double num = lista.get(1);
    double den = this.idMap.get(c.getCountry()).getPopulation();
    int f = (int)(den/num);
    if(f<0)
        f = ((f))*(-1));
    lista.add(3, f);
    return lista;
}

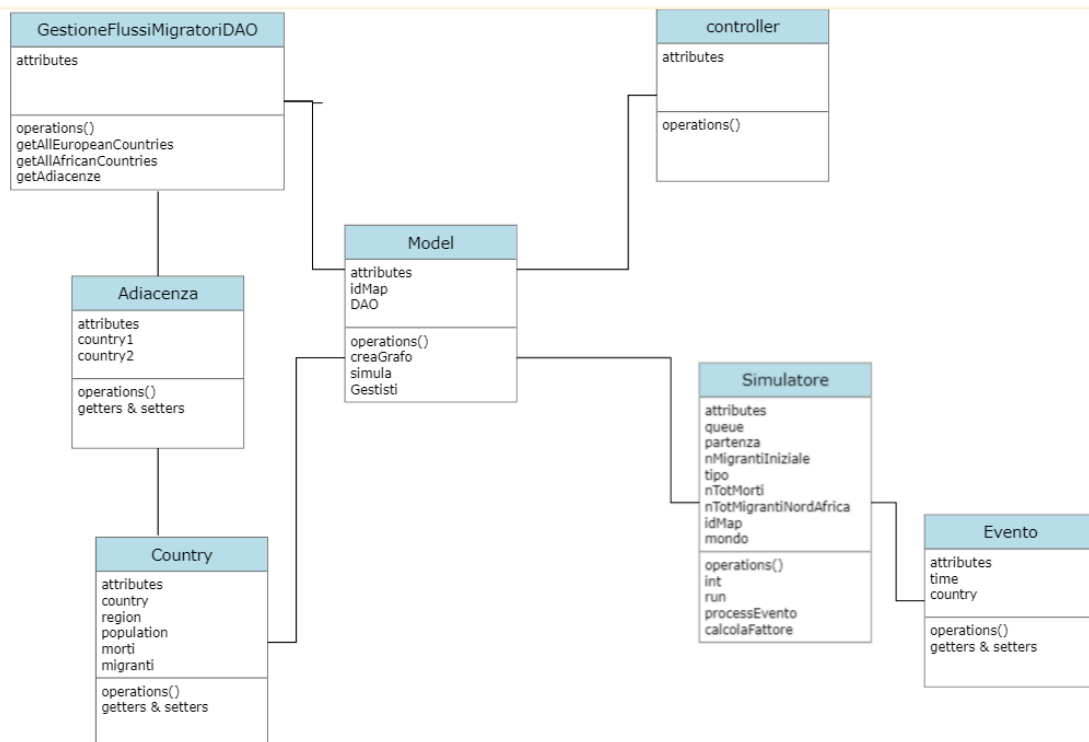
```

6 Diagramma Classi

6.1 Elenco Classi

- **Main**: per avviare l'applicazione
- **EntryPoint**: per costruire e collegare il controller
- **FXMLController**: per gestire la visualizzazione delle informazioni
- **DBConnect**: per collegare l'applicazione al nostro database, inserendo le nostre credenziali
- **GestioneFlussiMigratori**: Il Dao che crea corrispondenza tra le entità del database e le nostre classi Java
- **Model**: Il modello che si occupa della parte algoritmica dell'applicazione
- **Country**: la classe che contiene nome, popolazione, numero migranti e numero morti dello stato
- **ComparatorePerMorti**: ordina i country per numero di morti decrescente
- **Adiacenza**: classe di supporto per creare gli archi del nostro grafo che collegano stati adiacenti
- **Simulatore**: ciò che simula i flussi migratori dati alcuni input
- **Evento**: gli eventi che il simulatore dovrà gestire
- **TestModel**: tester per poter provare il model senza collegarlo al controller

6.2 Diagramma classi principali



7 Video dimostrativo


7.1 Schermate dell'applicazione

Gestione Migrazioni

Simulazione Gestione

Selezione Eventi

Crea Mondo



Nazione Evento Gravità

Mondo Creato!
Ora scegli Nazioni ed Eventi!

Reset Conferma

Nazione	Morti
Nessun contenuto nella tabella	

Totale migranti morti

Totale migranti coste NordAfrica

Gestione Migrazioni

Simulazione Gestione

Gestisci il flusso migratorio

Totale migranti coste NordAfrica

Scegli Nazione

Scegli approccio

Accogliente Respingente

7.2 Link tutorial Youtube

<https://youtu.be/qolsaZMpwQw>

8 Tabelle con risultati dimostrativi

8.1 Caso 1

In questo primo esempio abbiamo inserito questi dati di input

Una guerra di gravità 3 in Nigeria

Una carestia di gravità 1 in Sud Africa

Una crisi di povertà di gravità 2 in Ghana


Date le posizioni degli stati (tutte nella zona centro meridionale) ci aspettiamo che arrivino meno migranti in nord Africa e che ci siano molte morti, soprattutto in quella zona.

Gestione Migrazioni

SimulazioneGestione

Seleziona Eventi

Crea Mondo



Nazione

Evento

Gravità

Nigeria

Guerra

3

South Africa

Carestia

1

Ghana

Povertà

2

Reset

Conferma

Simulazione avvenuta con successo!

Nazione	Morti
Nigeria	17657895
South Africa	3109193
Niger	2286495
Cameroon	2098324
Chad	2078608
Benin	2033723
Central African Rep.	1471487
Congo, Dem. Rep.	1471483
Cote d'Ivoire	1471481
Burkina Faso	1155732
Ghana	1091470
Togo	856843
Botswana	586059
Zimbabwe	586053
Mozambique	543171

Totale migranti morti

119648734

Totale migranti coste NordAfrica

7445816

Possiamo infatti vedere come ci siano stati quasi 2 milioni di morti Nigeria e i primi stati sono tutti della zona centro meridionale dell’Africa, per un totale di quasi 120 milioni di morti. Mentre il numero di migranti arrivati in nord Africa è di circa 8 milioni, più di 10 ordini di grandezza inferiori rispetto ai morti.

8.2 Caso 2

In questo secondo esempio abbiamo utilizzato altri dati di input

Una guerra in Camerun di gravità 2

Una carestia in Burundi di gravità 3

Una persecuzione di gravità 1 nel Niger


Qui abbiamo ottenuto circa 76mila rifugiati

Simulazione

Gestione

Seleziona Eventi

Crea Mondo



Nazione	Evento	Gravità
Cameroon	Guerra	2
Burundi	Carestia	3
Niger	Persecuzione	1

Reset

Conferma

Simulazione avvenuta con successo!

Nazione	Morti
Cameroon	147788
Burundi	75803
Rwanda	57519
Nigeria	40149
Gabon	36940
Equatorial Guinea	36937
Chad	31572
Niger	30867
Uganda	20957
Kenya	8032
Benin	7560
Burkina Faso	5597
Sudan	5003
South Africa	4132
Mali	3985

Totale migranti morti

1364939

Totale migranti coste NordAfrica

75779

Scegliendo uno stato molto grande come la Francia ed un approccio aggressivo, notiamo che il numero di migranti che effettivamente ha raggiunto suolo francese è circa 21 mila (circa un quarto) mentre la spesa ammonta a più di 2 miliardi di euro.

The screenshot shows a web application titled "Gestisci il flusso migratorio". It has two tabs: "Simulazione" and "Gestione". The "Simulazione" tab is active. The interface includes a text input for "Totale migranti coste NordAfrica" with the value "75779". Below it, a dropdown menu for "Scegli Nazione" is set to "France", accompanied by the French flag. To the right, a slider for "Scegli approccio" is positioned between "Accogliente" (2.1) and "Respingente" (7.9), with the slider knob near the "Accogliente" end. At the bottom left are "Reset" and "Conferma" buttons. A text box at the bottom right displays the following data: "Dati raccolti e stime: Sono morti: 53965 migranti, Sono giunti: 21814 migranti, La spesa stimata ammonta a: 2113567296 euro, Rapporto tra popolazione e migranti attesi: 1 migrante ogni 2790 abitanti".

8.3 Caso 3

Qui possiamo notare la casualità nella scelta dell'approccio, utilizzando gli stessi input della simulazione precedente, otteniamo risultati diversi

This screenshot shows the same web application as the previous one, but with different results. The inputs are identical: "Totale migranti coste NordAfrica" is 75779, "Scegli Nazione" is France, and the "Scegli approccio" slider is near the "Accogliente" end. The "Dati raccolti e stime" box now shows: "Sono morti: 51335 migranti, Sono giunti: 24444 migranti, La spesa stimata ammonta a: 1850567296 euro, Rapporto tra popolazione e migranti attesi: 1 migrante ogni 2490 abitanti".

9 Conclusioni

9.1 Punti di forza

Il software costruito, per quanto approssimativo e idealizzato, svolge un lavoro soddisfacente. Riesce con i dati che ha a costruire un modello simile-reale di spostamento di persone in un contesto di migrazione. Anche dal punto di vista dei risultati, ho visto numeri di morti, di rifugiati e di spesa abbastanza vicini a quelli reali. Ciò non toglie che ci siano criticità.

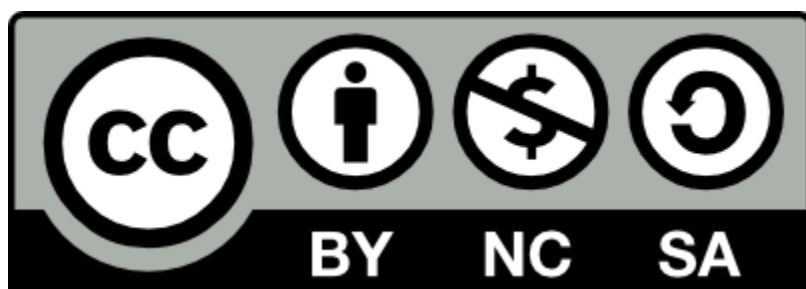
9.2 Punti critici

Il primo punto critico è sicuramente la mappa, come anticipato, essendo i database con alcune incompatibilità intrinseche, il grafo su cui si basa la simulazione riporta alcune falle.

Inoltre sarebbe utile avere altre informazioni sugli stati, di modo da poter costruire un modello ancora più complesso, come la stagionalità delle migrazioni, la capacità di accogliere migranti degli stati intermedi, la distanza tra stati, i rapporti etnici, culturali e religiosi degli stati che possono agevolare o frenare l'avanzata di questi flussi e altro ancora. Questo è il classico problema in cui parametri in più danno risultati esponenzialmente più accurati.

9.3 Conclusioni personali

Svolgendo la prova, ho potuto constatare come quando ci si affaccia ad un problema gestionale, avere davanti numeri a supporto delle proprie tesi e idee può svolgere un ruolo fondamentale nel capire quale sia la scelta giusta da intraprendere. Software e strumenti di questo tipo devono essere presenti per le grandi decisioni



Quest'opera è stata rilasciata con licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale.

Copia della licenza consultabile al sito web:

<http://creativecommons.org/licenses/by-nc-sa/4.0/>