EDITOR PER AUTOMI CELLULARI

DOMENICO BRUZZESE
FABIO CAPIRCHIO
GABRIELE FRARACCI
LEONARDO EMILI

1. Introduzione

Il progetto consiste nella creazione di un editor per automi cellulari e di una GUI. L'intero progetto é fortemente $user-oriented$, l'editor lascia libero l'utente di personalizarne ogni aspetto: dalla personalizzazione grafica fino a quella strutturale che gestisce il flusso della simulazione.
L'idea é quella di eseguire una simulazione che utilizzi i concetti di multithreading, concorrenza nonché i pilastri della OOP.

2. Tecnologie utilizzate

L'intero progetto é stato scritto in $Java$ 9, assieme a: $JavaFx$ utilizzato per creare la GUI, $FXML$ che ha contribuito ha formare la struttura del programma, CSS per le personalizzazioni grafiche $LaTex$ per la stesura di questa relazione.

3. Guida all'uso

Il programma ha un'interfaccia semplice e intuitiva ed é organizzato in 3 stadi logici: un primo stadio dove l'utente sceglie in che modalitá eseguire il programma, un secondo stadio dove vengono recuperate le informazioni necessarie affinché la simulazione possa iniziare, ed infine un ultimo stadio di interazione con il simulatore.

3.1. Schermata di benvenuto.

3.1.1. *Crea nuovo profilo: l'utente viene guidato in una procedura che gli permette di personalizzare i colori associati agli stati e le direttive che regolano lo sviluppo dell'automa. Al termine viene chiesto all'utente se si desidera preservare queste configurazioni per poterle riutilizzare in futuro, in caso contrario verrá creata una "shallow copy" delle configurazioni che verrá poi distrutta al termine dell'esecuzione del programma.*

3.1.2. *Carica profilo: il programma carica le configurazioni utente giá esistenti per essere poi fornite al simulatore.*

3.1.3. *Scegli automa: viene visualizzata una raccolta di automi cellulari con configurazioni e colori di default.*

3.2. **Schermata degli input.** In questa sezione l'utente é libero di scegliere gli stati delle cellule attraverso un'idea piú intuitiva: la scelta dei colori che le cellule dovranno avere, in questo modo si interagisce direttamente con il loro ciclo di vita. É inoltre possibile personalizzare i principi che regolano le transizioni di stato. Queste idee verranno trattate in maniera piú dettagliata nei paragrafi successivi.

3.3. **Schermata del simulatore.** Il protagonista, la tela del quadro é dove avviene la scena e dove é possibile assistere alla simulazione. Questa é circondata ai lati da pulsanti che regolano le configurazioni iniziali, mentre all'estremo inferiore sono localizzati i pulsanti di controllo con i quali é possibile interagire direttamente con la simulazione.

4. Automi cellulari

Nel seguito assumiamo che ciascuna cellula appartenga ad un unico stato, i motivi legati a questo dettaglio sono molteplici e la ragione apparirá piú chiaro nel *paragrafo 6* dove la questione viene trattata piú in dettaglio.
Nel testo viene inoltre utilizzata di frequente la nozione di vicino definita da *Edward Forrest Moore*, pioniere della *teoria degli automi cellulari*: il vicinato di una cella C é composto da 9 celle, la cella centrale C e dalle 8 celle che lo circondano.

4.1. **Game of Life.** L'automa sviluppato dal matematico *John Conway* é uno dei pilastri della collezione, l'apparente semplicitá con cui regole e stati vengono definiti portano all'occhio non pochi risvolti teorici. La scena si svolge su una griglia bidimensionale formata da cellule che possono assumere soltanto due stati: vivo o morto. Le regole di transizione adottate sono le stesse definite in origine dal matematico britannico:

- La cellula che si trova nello stato morto e conta esattamente 3 cellule vive nel suo vicinato transisce nello stato vivo.
- La cellula che si trova nello stato vivo e che ha o un numero di vicini maggiore strettamente di 3 oppure numero di vicini minore strettamente di 2 transisce nello stato morto.
- La cellula che si trova nello stato vivo e conta un numero di vicini pari a 2 o 3 sopravvive alla prossima generazione rimanendo nello stesso stato.

4.2. **Circular Redistribution.** Descrizione e regole che lo definiscono.

4.3. **Cyclic CA.** Descrizione e regole che lo definiscono.

4.4. **Diffusion Aggregation.** Descrizione e regole che lo definiscono.

4.5. **H3 Rule.** Descrizione e regole che lo definiscono.

5. Code Snippets

```
private void comeToMeCA(String className) {
    try {
            Class<?> C = CellularAutomataProgram.class.forName("simulatorWindow.programs."
                  + className);
            CellularAutomataProgram currentProgram = CellularAutomataProgram.class.cast(C.
                  getConstructor().newInstance());  simulator.setProgram(currentProgram);
    } catch (ClassNotFoundException e) {
            System.out.println("Class " + className + " has not been defined.");
    } catch (NoSuchMethodException e) {
            System.out.println("Constructor not found");
    } catch (IllegalAccessException e) {
            System.out.println(e.getMessage());
    } catch (InstantiationException e) {
            System.out.println(e.getMessage());
    } catch (InvocationTargetException e) {
            System.out.println(e.getMessage());
    }
}
```

6. Teoria degli stati e delle transizioni

Lo scenario precedentemente descritto si basa su due concetti principali: stati e regole di transizione.

6.1. **Gli stati.** L'idea di base é che ciasuna cellula al momento della creazione viene associata uno stato che ne regola lo sviluppo. Si noti come in questo modello gli stati non sono solo semplici proprietá di cui ogni cellula dispone, bensí essi sono parte attiva dell'evoluzione dell'automa. Questo implica che in un dato momento τ uno stato S puó avere nel proprio dominio D un numero indeterminato di cellule che oscilla tra lo 0 e l'intera popolazione di cellule. Nello scenario descritto vengono descritti i casi piú estremi, e logico dunque assumere che in un momento τ lo stato S conterá un numero medio di cellule che chiameremo m. In ogni istante ciascuno stato S interroga le proprie cellule e, in base alle regole di transizione definite , quest'ultime sono in grado di abbandonare S per essere integrate in un nuovo stato. Questo meccanismo viene eseguito contemporaneamente da parte di tutti gli stati che sono in grado di applicare le proprie regole senza interferire con cellule estranee al proprio dominio. Al termine della procedura tutte le possibili transizioni sono state eseguite e una volta eseguito il refresh globale della scena l'intero *algoritmo* é pronto a ripartire. Si noti come l'intero processo venga ***virtualizzato*** da parte degli stati: essi applicano regole su ciascuna delle proprie cellule che a loro volta possono mutare il loro stato interno, ma sará solo al termine di ciascun colpo di clock che le cellule renderanno visibile la loro transizione agli altri stati. Questo meccanismo di mutua esclusione tra gli stati impedisce loro di interferire, rendendo cosí possibile la parallelizzazione dei processi su un'unica griglia.

6.2. **Regole di transizione.** Le regole di transizione regolano i processi di mutazione delle cellule in ogni momento. Esse possono essere pensate come particolari *funzioni matematiche biunivoche* definite su un dominio di stati D e su un codominio di stati C. A ciascuno stato é associato, in base regole, un proprio stato di arrivo che puó, eventualmente, coincidere con lo stato di partenza nel caso in cui le regole lo prevedano.
Le regole sono completamente personalizzabili e si basano sui pilastri della **logica booleana**. Ogni complessa equazione booleana puó essere scomposta in proposizioni piú semplici che prendono il nome di *proposizione atomiche*. Secondo il *principio del terzo escluso* della logica booleana ciascuna proposizione atomica ha un unico valore di veritá: vero o falso. Il programma é in grado, attraverso una combinazione di molteplici proposizioni, di stabilire il valore di veritá di una generica regola o di una combinazione di regole ed di applicarle in tal senso. Alcuni dettagli implementativi a riguardo sono trattati nel *Paragrafo 5*.

7. Problemi matematici coinvolti

8. Curiositá