

Indice

0.1	Introduzione	2
0.2	Scopo del programma	3
0.3	Descrizione degli input e degli output	4
0.3.1	Il file Parametri.txt	4
0.3.2	Input da tastiera	6
0.3.3	Ouput: visualizzazione a video	7
0.3.4	Output: file di testo	7
0.4	Descrizione del sorgente	8
0.4.1	File di intestazione	9
0.4.2	Sottoprogrammi	9
0.4.3	Funzione principale: <i>main.c</i>	12
0.5	Esperimenti	14
0.5.1	Esperimento standard	14
0.5.2	Esperimento con scelta casuale in caso di parità di vicini	17
0.5.3	Esperimenti con perturbazioni iniziali	18
0.5.4	Esperimento con diverso valore di soglia	20
0.5.5	Esperimenti con rumore	21

0.1 Introduzione

Il modello LatticeWorld è un'applicazione di un automa cellulare utilizzata per studiare il fenomeno sociale dell'evoluzione degli orientamenti politici all'interno di un insieme di individui di riferimento.

Per automa cellulare si intende un sistema, costituito da un numero finito di entità, che evolvono all'interno di uno spazio diviso in celle discrete e che interagiscono tra di loro nel tempo, che evolve in passi discreti. Ad ogni cella è associata una variabile che può assumere valori discreti; si dice che la cella assume quei valori. Oltre ad essere discreto, l'automata cellulare è un sistema locale: lo stato di una cella al tempo $t + 1$ dipende solo dallo stato al tempo t delle celle che si trovano in un intorno della cella data.

Data la natura discreta di questi modelli, si rendono particolarmente adatti ad essere eseguiti su calcolatore per studiare l'evoluzione di particolari fenomeni naturali e sociali.

Il modello LatticeWorld simula il comportamento di un sistema di individui che formano il proprio parere sulla base delle scelte degli individui con i quali interagiscono più frequentemente data la loro vicinanza. Essi possono dunque cambiare la propria opinione in base alle proprie precedenti esperienze e a quelle degli individui a loro vicini.

Il sistema simulato è bipolare, ossia un sistema in cui gli individui hanno a che fare con una scelta binaria, per cui ogni cella può assumere solamente due valori: 0 o 1 (appartenente all'opinione zero oppure uno). In questo modello il mondo di riferimento è rappresentato da una struttura dello spazio cellulare bidimensionale; ogni cella corrisponde ad un individuo e il valore assunto dalla cella indica l'orientamento politico di quell'individuo. L'aggiornamento delle celle è sincrono: tutte le celle si aggiornano contemporaneamente, a istanti discreti, e il sistema è omogeneo: tutte le celle hanno lo stesso tipo di intorno e la stessa funzione di transizione. Utilizziamo l'intorno di Moore e una topologia toroidale: le celle che si collocano lungo i lati della matrice hanno come vicini le celle poste sul lato opposto. L'evoluzione del sistema è definita dalla funzione di transizione, ossia la regola di cambiamento di stato, che determina lo stato di ogni cella all'istante $t + 1$ a partire dallo stato della cella stessa e delle otto celle che ne costituiscono l'intorno all'istante t .

Non potendo definire a priori l'orientamento di ciascun individuo, si parte dal presupposto che al tempo $t = 0$ esso sia attribuito in maniera casuale. Ad ogni passo della simulazione tutti gli individui valutano il proprio intorno e aggiornano la propria opinione politica. La modalità di variazione delle opinioni politiche degli individui dipende dalla funzione di transizione che, nel modello LatticeWorld, prevede che un individuo si adatti al credo politico dominante all'interno della sua comunità di riferimento, ovvero il suo intorno, o *vicinato*. Se un individuo si trova circondato da cinque o più individui con credo politico uguale a 1, allora il suo stato futuro sarà 1. Se nel vicinato vi sono cinque o più individui con orientamento politico uguale a 0, il suo stato futuro sarà 0. Se ogni opinione politica è sostenuta da quattro vicini, allora l'opinione politica dell'individuo considerato resterà immutata oppure dipenderà da una scelta casuale (a seconda degli esperimenti). La scelta del valore soglia oltre il quale un individuo si adatta all'opinione dominante nel vicinato può

essere modificata: il parametro di *default* di tale valore, chiamato valore di *soglia*, è stato posto uguale a quattro, tuttavia, essendo il programma parametrizzato, è possibile variare tale valore e studiare l'evoluzione del sistema applicando differenti funzioni di transizione. Nelle sezioni Descrizione degli input e degli output e Descrizione del sorgente analizzeremo in dettaglio il funzionamento del programma per comprendere a pieno le conseguenze delle modifiche apportate ai parametri nei diversi esperimenti.

Nella sezione Esperimenti sono descritti alcuni test eseguiti sul modello. Tali esperimenti sono stati svolti per analizzare l'evoluzione delle opinioni politiche degli individui nel corso del tempo. Per avere una visione più ampia possibile del fenomeno in esame, si è intervenuti sul modello variando alcuni parametri come il valore di soglia, oppure introducendo modifiche sostanziali alla funzione di transizione ipotizzando che un individuo scelga a caso il suo orientamento politico in caso di equivalenza delle opinioni nel vicinato. Ulteriori modifiche apportate al modello sono descritte nella sezione Esperimenti.

0.2 Scopo del programma

Gli automi cellulari sono adatti a rappresentare e simulare l'evoluzione globale di fenomeni che dipendono solo da leggi locali.

Il comportamento globale di questi sistemi non è facilmente riconducibile al comportamento locale del sistema stesso, per cui le leggi che regolano il comportamento globale di questi sistemi non sono facilmente deducibili dall'analisi delle singole leggi che controllano ogni sottosistema costituente.

Lo scopo del nostro modello è quello di analizzare come semplici regole a livello locale possano influenzare la dinamica del sistema e la sua configurazione finale globale.

0.3 Descrizione degli input e degli output

Il programma da noi realizzato è di tipo *a console*, ovvero senza interfaccia grafica, e per il suo funzionamento necessita di un file di configurazione che deve chiamarsi “Parametri.txt”. Questo file di tipo testuale contiene una serie di input fondamentali per l’inizializzazione della matrice, il mondo di riferimento, e per definire la funzione di transizione.

Nel programma sono stati definiti dei valori di *default* per questi parametri, quindi, nel caso si vogliano utilizzare tali valori, non è necessario fornire al programma il file *Parametri.txt*. Per questo motivo, il programma fornisce *Runtime* la possibilità di scegliere tra lettura dei valori del file *Parametri.txt* e lettura delle costanti di *default*.

0.3.1 Il file Parametri.txt

Nel caso in cui si scelga di assegnare i valori contenuti all’interno del file *Parametri.txt*, il programma andrà a leggere tale file, che deve essere collocato nella stessa cartella del file eseguibile e deve essere strutturato come in Figura 1

```
Dimensione_Righe 20
Dimensione_Colonne 20
Passi 200
Soglia 4
Attesa 30000000
Proporzione 0.5
Rumore 1
Percentuale_Rumore 0.01
```

Figura 1: Esempio di file *Parametri.txt*.

I parametri necessari per il corretto funzionamento del programma sono:

- **Dimensione_Righe:** (*int*) numero intero che indica il numero di righe della matrice;
- **Dimensione_Colonne:** (*int*) numero che indica il numero di colonne della matrice;
- **Passi:** (*int*) numero che corrisponde al numero di iterazioni da far eseguire al programma;
- **Soglia:** (*int*) numero che corrisponde al numero di vicini di opinione contraria sufficiente a far cambiare orientamento politico all’individuo in considerazione;
- **Attesa:** (*long int*) numero intero che introduce una pausa durante l’esecuzione del programma, necessaria per osservare a video lo stato del sistema ad ogni passo;
- **Proporzione:** (*float*) numero in virgola mobile tra 0 e 1, corrisponde alla proporzione di 0 rispetto a quella di 1 nella matrice iniziale;

- **Rumore:** (*int*) può assumere valore 0 o valore 1. 1 indica l'attivazione della funzione di rumore, 0 indica la non attivazione di tale funzione.
- **Percentuale_Rumore:** (*float*) numero tra 0 e 1; se la funzione di rumore è attiva, indica la probabilità con cui, ad ogni aggiornamento di una cella, eseguire la funzione di rumore .

Il contenuto di questo file deve essere costruito seguendo precise regole:

- i parametri devono susseguirsi in ordine preciso;
- la descrizione del parametro deve essere una parola unica;
- tra la descrizione e il valore del parametro deve esserci uno spazio;
- ogni riga deve contenere una sola coppia di descrizione-valore del parametro;
- il carattere che contrassegna la virgola è il punto “.”.

Nel caso in cui si scelga di utilizzare i valori di *default*, il programma assegna a tali parametri i valori dichiarati come costanti all'interno del file di intestazione *Costanti.h*:

- **Dimensione_Righe:** 25
- **Dimensione_Colonne:** 25
- **Passi:** 200
- **Soglia:** 4
- **Attesa:** 30000000
- **Proporzione:** 0.5
- **Rumore:** 0
- **Percentuale_Rumore:** 0.1

Analizziamo ora in dettaglio i parametri sopra definiti.

I parametri **Dimensione_Righe** e **Dimensione_Colonne** rappresentano rispettivamente il numero di celle orizzontali e il numero di celle verticali della matrice.

Il parametro **passi** indica il numero di passi che il programma deve eseguire.

Per *passo* si intende un'evoluzione totale del sistema, alla fine della quale il tempo incrementa di una unità il suo valore. Quando il sistema evolve dalla configurazione all'istante t alla configurazione all'istante $t + 1$, ha compiuto un passo. L'evoluzione del sistema comprende l'aggiornamento di tutte le celle, la visualizzazione a video, un tempo di attesa, la pulizia del video e l'aggiornamento dei file di output.

Il parametro **soglia** influenza la regola di aggiornamento di ogni cella. Esso definisce il numero dei vicini con valore 1 oltre il quale la cella assume valore 1. Di *default* tale soglia è quattro: se il numero di 1 nell'intorno di una cella è minore di quattro, la cella assumerà valore 0, se è maggiore di quattro, il valore della cella sarà 1. Se il numero di vicini con stato 1 è uguale al valore della soglia, il valore che assumerà la cella dipende dagli esperimenti effettuati: talvolta rimane invariato, talvolta è casuale.

Il parametro **attesa** è utilizzato per aumentare il tempo di esecuzione in modo da agevolare la visualizzazione a video dei risultati. Ciò avviene attraverso l'esecuzione di una struttura di controllo iterativa che non ritorna alcun risultato: nel caso si scelga di utilizzare i valori di *default*, il ciclo *for* itera 30000000 volte.

Il parametro **proporzione** è utilizzato nell'inizializzazione della matrice ed indica la proporzione di 0 all'interno di tale matrice. Se il suo valore è 0.5, la probabilità di zero e uno è uguale; se tale valore è minore di 0.5 la probabilità di uno è maggiore della probabilità di zeri, viceversa se tale valore è maggiore di 0.5.

Il parametro **proporzione** è passato in input ad una funzione che genera casualmente 1 oppure 0. Questa funzione, chiamata *ZeroUnoRandom*, genera un valore casuale tra 0 e 1, lo confronta con il valore del parametro **proporzione** e, se il numero generato è maggiore del valore di **proporzione** restituisce 1, se è minore restituisce 0. Un valore del parametro **proporzione** minore di 0.5 dunque si traduce in una maggior probabilità che la funzione *ZeroUnoRandom* restituisca un 1 piuttosto che uno 0 e vice versa.

Il parametro **rumore** serve ad attivare o a disattivare l'esecuzione di una istruzione del programma attraverso una struttura di controllo decisionale. Se tale valore è 0 l'esecuzione della funzione *Rumore* non avviene, se è 1, l'esecuzione della funzione avviene.

La funzione *Rumore*, ritorna con una certa probabilità, determinata dal parametro **Percentuale Rumore**, un risultato contrario rispetto a quello definito normale. Il comportamento normale dell'aggiornamento delle celle segue infatti la regola della maggioranza sopra descritta: la cella assume stato 1 se nel suo intorno il numero di 1 è maggiore del valore di *soglia*, 0 altrimenti. Nel caso la funzione *Rumore* venga eseguita, con una certa probabilità, la cella non segue tale regola per il suo aggiornamento, ma segue la regola opposta: assume valore 1 se nel suo intorno la maggioranza di celle ha valore 0 e assume valore 0 se la maggior parte delle celle del suo intorno ha valore 1.

0.3.2 Input da tastiera

In fase di esecuzione il programma chiede all'utente, tramite un messaggio visualizzato a video, indicazioni sulla modalità di assegnazione dei valori ai parametri.

SCELTA DEI PARAMETRI:

1 – Leggi i parametri dal file Parametri.txt

2 – Utilizza parametri di default

Digita 1 o 2

Per dare tale indicazione, l'utente è tenuto a digitare il numero che precede la descrizione delle istruzioni. A seconda del numero digitato, i valori dei parametri verranno letti dal file esterno oppure dal file di intestazione contenente le costanti di *default*.

Nella fase immediatamente successiva è richiesto all'utente di dare indicazioni riguardo la modalità di inizializzazione del sistema. Il programma visualizza il seguente messaggio:

INIZIALIZZAZIONE DELLA MATRICE:

1 – Genera una matrice casuale

2 – Utilizza il file *input.txt* per inizializzare il sistema

Digita 1 o 2

Nel primo caso il sistema genera una matrice di dimensioni *Dimensione_Righe* x *Dimensione_Colonne* e la scrive nel file **input.txt** presente nella cartella all'interno della quale è contenuto il file eseguibile. In mancanza del file, lo crea.

Nel secondo caso il sistema legge il file *input.txt* e copia i suoi valori all'interno della matrice iniziale. Tale file deve dunque essere creato dall'utente e salvato nell'apposita cartella prima di iniziare l'esecuzione del programma e dovrà contenere tante righe e tante colonne quante definite nel file *Parametri.txt* nel caso si sia indicato di leggere i parametri dal file, dovrà invece contenere 25 righe e 25 colonne nel caso si sia indicato di utilizzare i parametri di *default*. Il file deve contenere 0 e 1 separati da uno spazio e alla fine di ogni riga è necessario andare a capo digitando il tasto *newline*.

0.3.3 Output: visualizzazione a video

Il programma fornisce due tipi di output: visualizzazione a video e file testuali.

Durante l'esecuzione, il programma visualizza a video l'evoluzione della configurazione della matrice iniziale: ad ogni passo visualizza in sequenza le matrici di stato.

Tra una visualizzazione e l'altra c'è un tempo di attesa, definito dal parametro *attesa* descritto precedentemente, dopo il quale lo schermo viene ripulito tramite un'istruzione che il programma manda al sistema operativo.

0.3.4 Output: file di testo

Alla fine della simulazione, il programma genera un file chiamato **output.txt** contenente lo stato finale del sistema, ossia la matrice di stato all'ultimo passo. Questo file è codificato allo stesso modo del file *input.txt*, ossia con la matrice di stato scritta riga per riga, con gli elementi composti da 0 e 1.

Durante l'inizializzazione del sistema viene generato un file chiamato **storia.txt** che sarà destinato a contenere il numero di 1 presenti in ogni matrice di stato. Ad ogni passo,

e per tutta la durata dell'esecuzione, questo file viene aggiornato con tale numero. Alla fine della simulazione si otterrà un file contenente una sequenza di valori del tipo:

205 207 211 219 226

Un esempio dei file *input.txt* e *output.txt* è riportato in Figura 2.

<pre> 1 1 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 0 0 0 1 0 0 1 0 1 1 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 1 0 1 1 1 1 0 0 0 1 1 0 1 0 1 0 1 1 0 0 1 0 0 0 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 0 0 1 1 0 1 1 0 0 0 1 1 0 0 1 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 </pre>	<pre> 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 </pre>
--	--

Figura 2: A sinistra: esempio di file *input.txt*.

A destra: esempio di file *output.txt*.

0.4 Descrizione del sorgente

Per creare il programma è stato utilizzato l'IDE (Integrated Development Environment) XCode; un ambiente integrato che racchiude una gamma completa di strumenti per la progettazione, lo sviluppo e la realizzazione di un qualsiasi software.

Il sorgente è poi stato compilato per ambiente Windows con BuilderX della Borland.

I diversi file sorgenti sono separati in diversi moduli:

- i file con estensione *.c* sono i cosiddetti *sottoprogrammi*; essi contengono un insieme di istruzioni - una funzione - che hanno un determinato compito, e che sono dunque separabili e riutilizzabili;
- i file con estensione *.h* sono gli *header file*; un file di intestazione contiene dichiarazioni di tipi di dati, dichiarazioni di strutture e prototipi delle funzioni. Sono utilizzati in modo da poter utilizzare funzioni, strutture e tipi di dati anche in altri moduli.

0.4.1 File di intestazione

Costanti.h

In questo modulo definiamo, attraverso l'istruzione `define`, degli identificatori di costanti in corrispondenza dei valori di *default* dei parametri utilizzati.

Parametri.h

File in cui, mediante l'istruzione `struct`, viene dichiarata una struttura dati chiamata `TipoParametri`, dedicata a contenere i parametri per l'esecuzione del programma: è infatti composta da otto membri, uno per ogni parametro descritto nella sezione 0.3.1.

0.4.2 Sottoprogrammi

LeggiFileParametri.c

Questo modulo contiene la funzione `LeggiFileParametri`, la quale legge i parametri dal file *Parametri.txt* e li inserisce nella struttura dati che le è stata passata per riferimento.

È dunque necessario includere, prima della definizione della funzione, il file di intestazione `Parametri.h` attraverso l'istruzione

```
#include "Parametri.h"
```

La definizione della funzione è la seguente:

```
void LeggiFileParametri(struct TipoParametri *tmp_ptr_parametri)
```

e necessita, come parametro di input, un puntatore alla struttura dati `TipoParametri`.

Questa funzione viene chiamata dalla funzione principale nel caso l'utente scelga di utilizzare i parametri contenuti nel file di testo *Parametri.txt*.

LeggiParametriDefault.c

Funzione che legge i parametri dal file di intestazione `Costanti.h`, incluso nelle direttive di compilazione, e assegna i corrispondenti valori ad ogni campo della struttura dati che le è stata passata per riferimento. Questa funzione viene chiamata dalla funzione principale nel caso l'utente scelga di utilizzare i parametri di *default*.

InizializzazioneMatrice.c

La funzione `InizializzazioneMatrice` è di tipo `void`, vuole come parametro di input un puntatore alla struttura dati e viene chiamata dalla funzione principale nel caso l'utente scelga di far inizializzare la matrice al sistema.

La funzione crea il file *input.txt* nel quale scrive la matrice iniziale. Se il file esiste già, la

funzione, che a tale scopo utilizza l'istruzione `fopen(input.txt, w)`; lo sovrascrive. Per la generazione di numeri casuali viene utilizzata la funzione `ZeroUnoRandom`.

ZeroUnoRandom.c

Questo modulo contiene la funzione `ZeroUnoRandom`, la cui dichiarazione è:

```
int ZeroUnoRandom(int flag, struct TipoParametri *tmp_ptr_parametri)
```

Essa restituisce casualmente 1 oppure 0 con una probabilità che dipende dal parametro *proporzione* nel caso in cui la variabile `flag` sia 0, oppure con una probabilità 0.5 nel caso la variabile intera passata come parametro di input sia 1.

La generazione di un numero casuale avviene tramite la chiamata della funzione `rand`, la quale restituisce casualmente un numero intero tra 1 e `RAND_MAX`. Tale numero viene riportato ad un numero tra 0 e 1 tramite la sua divisione per `RAND_MAX` ed una conversione esplicita di tipo da `int` a `float`.

CopiaMatrice.c

Questo modulo contiene la funzione `CopiaMatrice`, la quale è dichiarata nel seguente modo:

```
void CopiaMatrice(int **matrice, struct TipoParametri *tmp_ptr_parametri)
```

Essa copia i valori della matrice contenuta nel file *input.txt* nelle celle corrispondenti della matrice passata per riferimento. Le due matrici, quella all'interno del file e quella passata per riferimento, devono essere della stessa dimensione per cui è opportuno ricordare di controllare bene i parametri prima di lanciare una simulazione. Se il file *input.txt* non esiste, tale funzione visualizza a monitor un messaggio di errore: *File non trovato*.

VisualizzazioneMatrice.c

La funzione `VisualizzazioneMatrice` vuole, come parametro di input, il riferimento alla matrice da visualizzare a video. La funzione visualizza il carattere `*` in corrispondenza di un 1 e il carattere spazio in corrispondenza di uno 0 (Figura 3).

InizializzazioneStorico.c

La funzione `InizializzazioneStorico` conta il numero di 1 presenti nella matrice che le viene passata per riferimento attraverso la funzione `ContaUni` e crea nella cartella del file eseguibile il file *storia.txt* nel quale scrive tale valore. Se il file esiste già lo sovrascrive.

ContaUni.c

La funzione `ContaUni` conta il numero di 1 presenti nella matrice passata per riferimento e restituisce tale numero.

Avanzamento.c

La funzione `Avanzamento` prende in considerazione ogni cella della prima matrice passatale per riferimento, ne calcola l'intorno tramite la funzione `CalcoloVicini`, e riempie le celle corrispondenti della seconda matrice passatale per riferimento a seconda di quello che ritorna la funzione `CalcoloVicini`. L'utilizzo di due matrici consente che l'aggiornamento delle celle avvenga in modo sincrono.

CalcoloVicini.c

La funzione `CalcoloVicini` calcola la quantità di numeri 1 nell'intorno di una determinata cella della matrice passatale per riferimento.

La funzione ritorna 1 o 0 a seconda che il numero di 1 nell'intorno della cella sia maggiore o minore del parametro *soglia* passato alla funzione attraverso un riferimento alla struttura `dati`. Nel caso di uguaglianza alla soglia, lo stato futuro della cella è scelto a caso chiamando la funzione `ZeroUnoRandom` con parametro di input intero 0.

Nel caso in cui sia attivata la funzione `Rumore` (quindi nel caso in cui il parametro *rumore* sia 1), la funzione `CalcoloVicini` utilizza la funzione `ZeroUnoRandom` per generare casualmente un numero tra 0 e 1. Nel caso in cui questo numero sia minore del parametro *percentuale_rumore*, chiama la funzione `Rumore`.

Rumore.c

La funzione `Rumore` ritorna 1 se il numero di 1 nell'intorno di una cella è minore della soglia, 0 altrimenti. Nel caso di uguaglianza alla soglia, lo stato è scelto casualmente chiamando la funzione `ZeroUnoRandom` con parametro di input intero 0.

Attesa.c

La funzione `Attesa` ha come scopo il rallentamento dell'esecuzione del programma attraverso l'esecuzione di una struttura di controllo iterativa: un ciclo `for` - che non esegue alcun calcolo - che va da 0 al valore del parametro *attesa*, passatole per riferimento alla struttura `dati`. In questo modo permette una buona visualizzazione di ogni matrice di stato: un tempo di attesa fra la visualizzazione di due matrici successive.

InversioneMatrici.c

Funzione che inverte i valori di due matrici che le vengono passate per riferimento. Le due matrici devono avere la stessa dimensione.

AggiornamentoStorico.c

Funzione che apre il file *storia.txt* in modalità *append* e aggiunge in coda il numero di 1 presenti nella matrice passata per riferimento

ControlloUguaglianza.c

Funzione che confronta due matrici passate per riferimento; nel caso esse siano diverse ritorna 0, nel caso siano uguali visualizza un messaggio a video, e ritorna 0, nel caso siano uguali e i loro elementi siano tutti 1 o tutti 0, ritorna 1.

StampaOutput.c

Funzione che stampa la matrice passata per riferimento sul file *output.txt*; se il file non esiste lo crea, se esiste lo sovrascrive.

Deallocazione.c

Funzione che dealloca gli oggetti dinamici inizialmente allocati: le due matrici. Utilizza la funzione *free* per liberare lo spazio di memoria occupato dagli oggetti dinamici.

0.4.3 Funzione principale: *main.c*

La funzione principale, l'unica obbligatoria nel linguaggio di programmazione C, contiene lo scheletro di tutto il programma. Grazie alla scomposizione del programma in moduli, la funzione *main* risulta di facile lettura. Come ogni funzione principale, inizia con le direttive di compilazione: l'inclusione dei riferimenti alle librerie (nel nostro caso le librerie di sistema *stdio*, *stdlib*, *time* e *conio*) e ai file di intestazione, e l'elenco delle dichiarazioni delle funzioni create dal programmatore (i prototipi).

Nel corpo del programma troviamo un parte dichiarativa in cui vengono dichiarate le variabili utilizzate nella parte esecutiva.

Nella parte esecutiva troviamo le seguenti istruzioni che vengono eseguite in sequenza:

- generazione del seme random attraverso le funzioni *srand*, che vuole in input un intero senza segno (*unsigned*), e *time*, che restituisce un *unsigned* che rappresenta l'ora corrente del giorno espressa in secondi;
- inizializzazione della variabile puntatore alla struttura;
- lettura dei parametri;
- eventuale creazione del file *input.txt*;
- allocazione dinamica della memoria, tramite la funzione *calloc*, per le due matrici (quella principale e quella di supporto) che servono per l'aggiornamento;

- inizializzazione della matrice;
- visualizzazione della matrice iniziale;
- creazione del file *storia.txt*;

Successivamente troviamo una struttura iterativa che esegue un ciclo `for` che termina quando raggiunge i passi definiti da parametro. Ad ogni passo il programma esegue le seguenti istruzioni:

- stampa a video il passo corrente;
- aggiornamento delle celle della matrice;
- visualizzazione a video della matrice aggiornata;
- attesa;
- inversione degli elementi tra le due matrici;
- aggiornamento del file *storia.txt*;
- confronto tra le due matrici: se sono identiche e le celle sono tutte 1 o 0, creazione del file *output.txt*, deallocazione dello spazio di memoria ed uscita dal programma;
- nel caso l'utente prema "invio", il programma esce dal ciclo.
 Risultato ottenuto utilizzando le funzioni `kbhit` e `getch`: la prima appartiene alla libreria `conio`, che non è una libreria standard, quindi non fa parte dei 24 *header file* della libreria C ANSI, ma è una libreria della Borland e quindi è presente e riconosciuta solo nei suoi ambienti di programmazione. La funzione `kbhit` controlla se è stato premuto qualche tasto dalla tastiera e ritorna un valore diverso da 0 in caso affermativo, uguale a 0 in caso negativo. In caso positivo la funzione `getch` ritorna il codice ASCII del tasto premuto. Assegnando il risultato restituito da questa funzione ad un intero e confrontandolo con il codice ASCII del carattere "invio" (13), facciamo terminare l'esecuzione del ciclo tramite l'istruzione `break`.

Infine, nel caso in cui la funzione `ControlloUguaglianza` ritorni 1, quindi alla fine dei passi da effettuare, le ultime istruzioni del programma sono:

- creazione del file *output.txt*;
- deallocazione esplicita degli oggetti dinamici;
- esci.

0.5 Esperimenti

Per analizzare l'evoluzione delle opinioni politiche degli individui nel corso del tempo sono stati effettuati alcuni esperimenti.

In questa sezione descriviamo tali esperimenti e riportiamo i rispettivi risultati.

Ad ogni tipo di esperimento corrisponde una differente configurazione dei parametri o una differente modalità di esecuzione del modello. In alcuni casi abbiamo modificato solo i parametri che definiscono i valori elementari del modello (numero di righe della matrice o valore di soglia), in altri casi abbiamo introdotto modifiche alla funzione di transizione, facendo alcune ipotesi di base come, per esempio, che un individuo scelga a caso il suo orientamento politico in caso di equivalenza di opinioni nel suo vicinato, oppure introducendo la funzione di *Rumore* che comporta una modalità di aggiornamento dello stato di una cella probabilisticamente contraria a quella normale. In questo ultimo caso la funzione di transizione non prescrive il nuovo stato delle celle ma una distribuzione di probabilità dei nuovi stati: l'automa cellulare non è più deterministico ma stocastico.

0.5.1 Esperimento standard

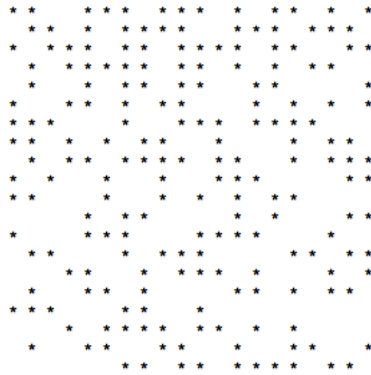
Esperimento effettuato con i seguenti parametri:

- numero di righe: 20
- numero di colonne: 20
- soglia: 4
- passi: 100
- tempo di attesa: 30000000;
- proporzione di uno: 0.5;
- rumore 0.

Specifiche:

- le le celle seguono la *regola della maggioranza*;
- se il numero di 1 nell'intorno di una cella è pari alla soglia, lo stato della cella rimane invariato;
- il numero iniziale di zeri ed uni è uguale; 200 uni e 200 zeri;
- nessuna generazione di numeri casuali.

In Figura 3 è riportato un esempio di matrice iniziale:



*Figura 3: Esempio di matrice iniziale con equal numero di zeri e di uno.
Gli 0 sono rappresentati da uno spazio, gli 1 dal carattere asterisco.*

Partendo da una situazione iniziale come quella riportata in Figura 3, durante l'evoluzione del sistema si nota che vengono a formarsi delle isole di zeri e di uno, le cui definizioni dei confini vengono a stabilizzarsi dopo pochi passi. Raggiunta questa stabilità finale, il sistema non cambia più la sua configurazione e la mantiene costante nel tempo.

Per questo motivo, solo per questo ciclo di simulazioni, abbiamo implementato nel programma una funzione che verifica l'uguaglianza tra le matrici di due stadi successivi, e che termina l'esecuzione del programma stesso nel caso le due matrici siano uguali, stampando a video un messaggio che ci informa del fatto che il sistema ha raggiunto il suo stato finale permanente. In Figura 4 è riportato un esempio di matrice finale:



Figura 4: Matrice finale ottenuta partendo dalla matrice iniziale riportata in Figura 3

In realtà già dopo i primi tre passi si inizia a intravedere lo stato finale, mentre nei passi successivi si definiscono bene i confini. La Figura 5 riporta il terzo passo della simulazione effettuata con la matrice iniziale di Figura 3:



Figura 5: Matrice al passo 3 ottenuta partendo dalla matrice iniziale riportata in Figura 3

Come si può notare le isole si sono già formate, ma i loro confini non sono ancora definiti. Tale comportamento si è verificato in tutte le dieci simulazioni lanciate con diverso seme random, dunque possiamo definirlo *caratteristico*.

Conclusione: il sistema raggiunge sempre uno stato asintotico in media entro i primi 12 passi. I primi passi li impiega per accorpere gli uni e gli zeri e a separarli, gli altri li impiega per definire i confini tra le isole formate. È un sistema deterministico: lo stato di ogni cella al tempo $t + 1$ è determinato solo dallo stato all'istante t delle celle del suo intorno; per cui partendo sempre dallo stesso stato iniziale, si arriva sempre allo stesso stato finale.

Variazione dell'esperimento standard

Esperimento effettuato con gli stessi parametri e con le stesse specifiche dell'esperimento standard, ma con generazione di numeri casuali in fase di inizializzazione delle matrici.

In questi esperimenti la matrice da cui il sistema parte non è stata fornita da noi e non ha necessariamente lo stesso numero di zeri e di uno. Essa è inizializzata dal sistema attraverso la generazione di numeri casuali con la stessa distribuzione di probabilità di zeri e uno. La dinamica del sistema non è differente da quella dell'esperimento precedente ma, effettuando una media sulle dieci simulazioni, risulta che il tempo impiegato dal sistema per raggiungere uno stato asintotico è inferiore di tre time step. Questo risultato può essere determinato dal fatto che non avendo lo stesso numero di uni e di zeri, la *diversità* all'interno della matrice iniziale è inferiore rispetto alla *diversità* all'interno di una matrice con ugual numero di zeri e uno.

Aumentando poi le dimensioni della matrice, e lanciando dieci volte ad ogni aumento, si nota un aumento del tempo impiegato dal sistema per raggiungere uno stato asintotico. La Tabella 1è una rappresentazione di questi risultati.

Tabella 1: passi effettuati dal sistema prima di giungere ad uno stato asintotico relativi a matrici di dimensione differente.

Dimensione della matrice	Time Step
20 x 20	9
25 x 25	13
30 x 30	15
40 x 40	17
50 x 50	19
60 x 60	20
70 x 70	23

Questo risultato potrebbe essere dato dal fatto che aumentando la dimensione della matrice aumenta lo spazio a disposizione e il numero di isole di zeri e di uno che si vanno a formare è maggiore e quindi è di più anche il tempo necessario a definire i bordi di tali isole.

0.5.2 Esperimento con scelta casuale in caso di parità di vicini

In questo esperimento manteniamo i parametri standard, modificando una specifica:

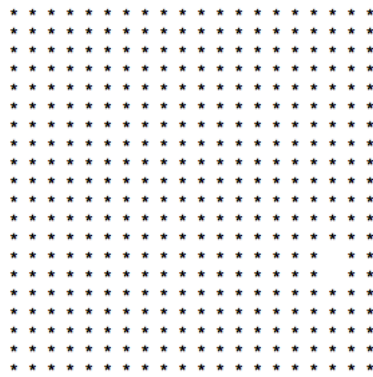
- se il numero di 1 nell'intorno di una cella è uguale al numero di 0, e dunque pari alla soglia, lo stato della cella è scelto a caso con uguale probabilità fra zero e uno.

Effettuando 50 simulazioni arriviamo alla conclusione che il sistema, a queste condizioni, può arrivare a due stadi finali differenti:

1. *Corrosione*: osservando a video l'evoluzione del sistema, sembra che nel corso dei time step, i numeri uno corrodano gli zero o viceversa. Avviene dunque un costante aumento dei numeri uno che alla fine predomina sugli zero, arrivando ad una matrice finale composta totalmente di uno (o viceversa). Il sistema raggiunge dunque uno stato asintotico omogeneo, nel quale un'opinione predomina sull'altra diffondendosi in tutta la popolazione. In questi casi il sistema raggiunge lo stato asintotico in media al passo 48. La Figura 6 mostra due passi di una simulazione con esito corrosione.
2. *Loose Attractor*: attrattore che implica il susseguirsi di stati in una certa zona dello spazio delle fasi, senza un particolare ordine. Il sistema non arriva dunque a raggiungere uno stato asintotico, ma raggiunge una configurazione stabile ed evolve rimanendo nelle vicinanze di quella configurazione. Tale stabilità consiste di due o più differenti configurazioni della matrice di output, le quali si alternano nei passi successivi. Un esempio di questo stato finale è mostrato in Figura 7.



Step 53



Step 199



TERM environment variable not set.
Step 200



TERM environment variable not set.

Figura 6: un possibile stato finale del sistema: corrosione. Figura 7: loose attractor

Effettuando lo stesso tipo di esperimento generando la matrice iniziale con la stessa probabilità fra zeri e uno, gli esiti ottenuti sono gli stessi.

Modificando le dimensioni della matrice il risultato rimane invariato. L'unica differenza è che il sistema, in caso raggiunga uno stato asintotico omogeneo, impiega in media 63 passi.

0.5.3 Esperimenti con perturbazioni iniziali

Questo esperimento consiste nell'effettuare simulazioni con matrici iniziali con differenti proporzioni di zero e uno. Per farlo abbiamo immesso diversi valori relativi al parametro *proporzione* nel file *Parametri.txt* e abbiamo fatto inizializzare la matrice al sistema.

Maggioranza di uno: parametro *proporzione* minore di 0.5

I parametri utilizzati sono i parametri dell'esperimento standard e nel caso in cui una cella abbia lo stesso numero di vicini 1 e di vicini 0, il suo stato futuro è scelto a caso con uguale probabilità di 0 e 1.

Abbiamo modificato il parametro *proporzione* da 0 a 0.49 e abbiamo effettuato dieci lanci per ogni valore. Riportiamo di seguito i risultati delle simulazioni:

- ***proporzione 0:*** tutte le celle della matrice iniziale sono 1; quindi il sistema non cambia perchè già al primo passo la matrice di stato è identica a quella iniziale.
- ***proporzione 0.1:*** La matrice iniziale è composta per la maggior parte da 1. In media tale matrice contiene 356 numeri uno su 400, che significa l'89 % di 1 rispetto all'11% di 0. Il sistema è arrivato ad uno stato asintotico omogeneo (tutte le celle con valore 1) al passo 1 in sei simulazioni. Nelle altre quattro simulazioni vi ha impiegato 2 passi. Esaminando il file *storia.txt* abbiamo osservato che nelle simulazioni in cui il sistema impiega un time step in più per raggiungere il suo stato finale, il numero di 1 nella matrice iniziale non è minore di quello delle altre simulazioni. Analizzando la matrice iniziale e confrontandola con quella finale siamo giunti alla conclusione che la motivazione per cui il sistema ha impiegato maggior tempo è da imputarsi al fatto che nella configurazione della matrice iniziale vi erano quattro o più 0 vicini.
- ***proporzione 0.2:*** la percentuale di 1 nella matrice iniziale è dell'80.5%. Il sistema arriva ad uno stato asintotico omogeneo dopo 2 passi (in cinque simulazioni), 3 passi (in quattro simulazioni) e 4 passi (in una simulazione).
- ***proporzione 0.3:*** 69% di 1 nella matrice iniziale. Dalle simulazioni effettuate risulta che il sistema raggiunge uno stato asintotico omogeneo tra i 3 e i 9 passi.
- ***proporzione 0.4:*** la percentuale di 1 nella matrice iniziale è del 60%. Anche in queste simulazioni l'unico esito è stato il raggiungimento di uno stato asintotico omogeneo nel quale tutte le celle hanno valore 1. Il sistema ha mostrato raggiungere tale stato fra il passo 5 e il passo 19.
- ***proporzione 0.45:*** Il 55% delle celle della matrice iniziale è 1. In nove simulazioni su dieci il sistema ha raggiunto uno stato asintotico omogeneo entro i primi 47 passi. In un solo caso il sistema ha raggiunto un loose attractor.
- ***proporzione 0.49:*** Il 50.75% delle celle della matrice iniziale è 1. In queste simulazioni l'esito di un loose attractor si è verificato con più frequenza rispetto alle simulazioni precedenti: in sei delle dieci effettuate. In due simulazioni l'esito è stato il raggiungimento del sistema, entro i 38 passi, di uno stato asintotico omogeneo (tutti 1), e in una simulazione il sistema ha raggiunto in 27 passi uno stato asintotico in cui tutte le celle hanno assunto valore 0. Nell'ultima simulazione il sistema ha raggiunto al passo 92 uno stato asintotico caratterizzato da due isole nettamente separate.

Maggioranza di zero: parametro *proporzione* maggiore di 0.5

In queste simulazioni il risultato ottenuto è esattamente speculare a quello ottenuto nelle simulazioni in cui il valore del parametro *proporzione* è minore di 0.5.

Riassumendo abbiamo notato che per valori del parametro che vanno da 1 a 0.6, l'unico esito delle simulazioni è stato il raggiungimento di uno stato asintotico omogeneo in cui tutte le celle hanno assunto valore 0. Il tempo impiegato dal sistema per raggiungere tale configurazione aumenta con il diminuire della proporzione di 0 rispetto agli 1; ossia aumenta al diminuire del parametro *proporzione*.

Gli esperimenti con valore del parametro pari a 0.55 hanno due esiti possibili: il raggiungimento del sistema di uno stato asintotico omogeneo (tutti 0) e il raggiungimento di un loose attractor. Questo ultimo risultato si verifica maggiormente se fissiamo il valore del parametro a 0.51. Con quest'ultimo parametro, talvolta il sistema arriva ad una configurazione finale in cui tutte le celle hanno valore 1.

Possiamo concludere che con *proporzione* 0.49 o 0.51, le piccole perturbazioni iniziali non si propagano nel sistema e non vi è alcuna differenza di esito tra la condizioni iniziale normale (*proporzione* 0.5) e la stessa condizione leggermente modificata.

Applicando perturbazioni iniziali più consistenti (*proporzione* da 0.1 a 0.4 e da 0.6 a 1), esse si propagano in tutto il sistema.

Esiste infine una condizione iniziale di media perturbazione (*percentuale* 0.45 o 0.55) in cui le differenze a volte si propagano per tutto il sistema, a volte non si propagano e si può supporre che talvolta si propagano ma, rimanendo localizzate, non influenzano tutto il sistema.

0.5.4 Esperimento con diverso valore di soglia

Per effettuare questi esperimenti abbiamo modificato il valore del parametro *soglia* nel file *Parametri.txt*. I valori dei parametri sono quelli dell'esperimento standard e, nel caso in cui il numero di 1 nell'intorno della cella sia uguale al valore della soglia, l'aggiornamento della cella avviene casualmente. L'inizializzazione della matrice iniziale è fatta dal sistema con uguale probabilità di zero e uno (il parametro *proporzione* è infatti pari a 0.5)

Soglia 3

Il sistema raggiunge uno stato asintotico omogeneo in cui tutte le celle hanno valore 1, in media al passo 4. Dalle venti simulazioni è risultato che raggiunge la sua configurazione finale tra il passo 3 e il passo 5. Infatti con soglia inferiore alla metà dell'intorno le celle con valore 1 hanno maggior facilità di diffondersi nel sistema.

Soglia 5

Dalle venti simulazioni effettuate risulta che il sistema raggiunge uno stato asintotico omogeneo in cui tutte le celle hanno valore 0 in media in 4 passi (tra i 3 e i 5 passi). Infatti le celle con valore 1 possono conservarsi soltanto se, nella matrice iniziale, sono "agglomerate" all'interno di isole di celle simili, il che è molto raro data l'inizializzazione casuale della matrice.

0.5.5 Esperimenti con rumore

Per effettuare questi esperimenti abbiamo modificato il valore del parametro *rumore* che fino a questo momento era settato a 0. Settandolo a 1 nel file *Parametri.txt*, abbiamo attivato la funzione *Rumore*. In questi esperimenti dunque la cella, con una certa probabilità (determinata dal parametro *percentuale_rumore*), non segue la regola della maggioranza, ma esegue l'aggiornamento opposto: 0 in caso di maggioranza di uno e 1 in caso di maggioranza di zero.

La matrice iniziale è inizializzata dal sistema con uguale probabilità di zero e uno.

In caso il numero di 1 nel vicinato di una cella sia uguale al valore del parametro *soglia*, l'aggiornamento di tale cella avviene in modo casuale.

percentuale_rumore minore di 1

Settando il valore del parametro *percentuale_rumore* a 0.01 ed osservando l'evoluzione del sistema a video, si vede chiaramente che il sistema evolve verso le configurazioni di stabilità descritte nei precedenti esperimenti; tuttavia, ad ogni step, ci sono da una a sette *opinioni contrarie*, ossia celle che non seguono la regola della maggioranza per l'aggiornamento. Durante i primi passi della simulazione, durante la "stabilizzazione" del sistema, non è chiaro quanti siano le opinioni contrarie. Dopo che il sistema ha raggiunto la "stabilità", invece, è facile vedere che ad ogni passo compaiono pochi individui che esprimono opinione contraria. La media di tali opinioni ad ogni step è 4; questo conferma il corretto funzionamento del programma infatti 4 su 400 corrisponde esattamente allo 0.01%.

Lo stesso accade nel caso in cui il valore del parametro *percentuale_rumore* sia 0.1, con la differenza che i le opinioni contrarie sono fra le 30 e le 50 ad ogni passo; in media 41.

I grafici sottostanti mostrano il numero di 1 ad ogni passo della simulazione: il grafico riportato in Figura 8 è relativo ad una delle simulazioni con parametro *percentuale_rumore* pari a 0.01, il grafico in Figura 9 con parametro *percentuale_rumore* 0.1. In entrambi i casi le opinioni contrarie non sono sufficienti a cambiare le opinioni dei vicini; le perturbazioni, anche se continue, non sono in grado di creare isole con individui con opinione contraria a quella definita come vincente dalla "stabilità" raggiunta dal sistema.

Aumentando il valore relativo alla percentuale di rumore ed effettuando diverse simulazioni settandolo a 0.2, 0.3, 0.4, 0.5, 0.6 e 0.7, il sistema sembra non raggiungere alcuna configurazione stabile: le matrici di stato che si susseguono nel tempo sono tutte simili alla matrice iniziale. Il numero di 1 varia nel tempo: con percentuale di rumore 0.2 esso tende ad allontanarsi dal valor medio; con percentuale di rumore 0.3 e 0.7 il numero di 1 è variabile ma rimane più vicino al valor medio 200; con percentuali di rumore 0.4, 0.5, 0.6 le oscillazioni si allontanano meno dalla media, come mostrato in Figura 10 e in Figura 11.

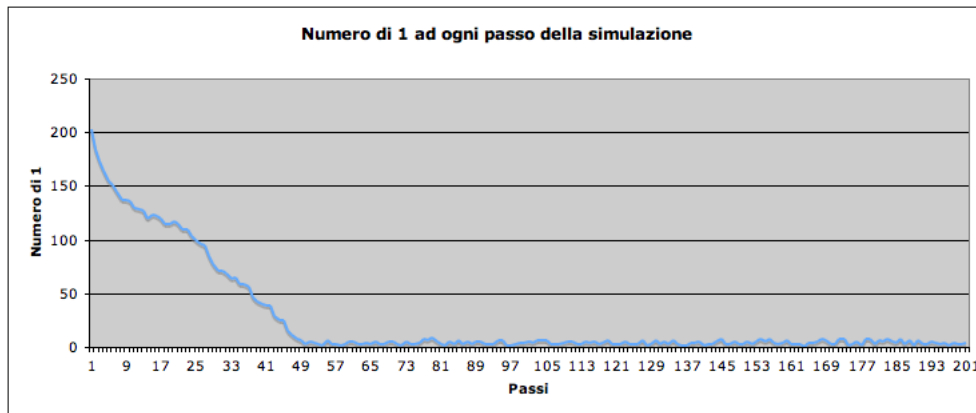


Figura 8: grafico del file storia.txt. Simulazione con percentuale _rumore 0.01

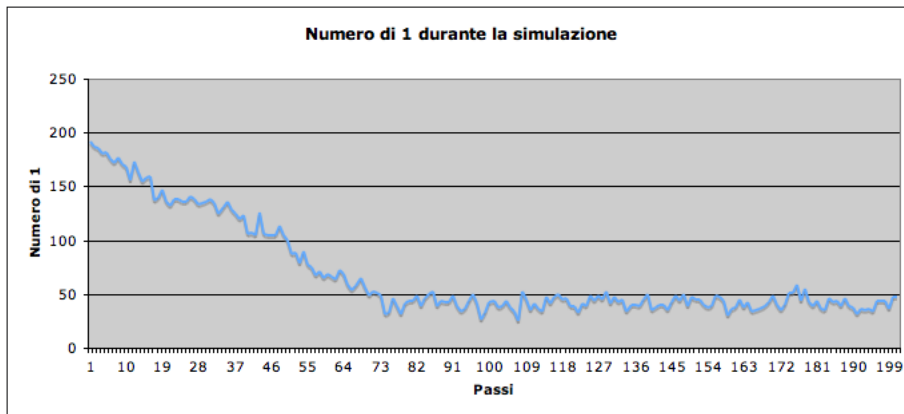


Figura 9: grafico del file storia.txt. Simulazione con percentuale _rumore 0.1

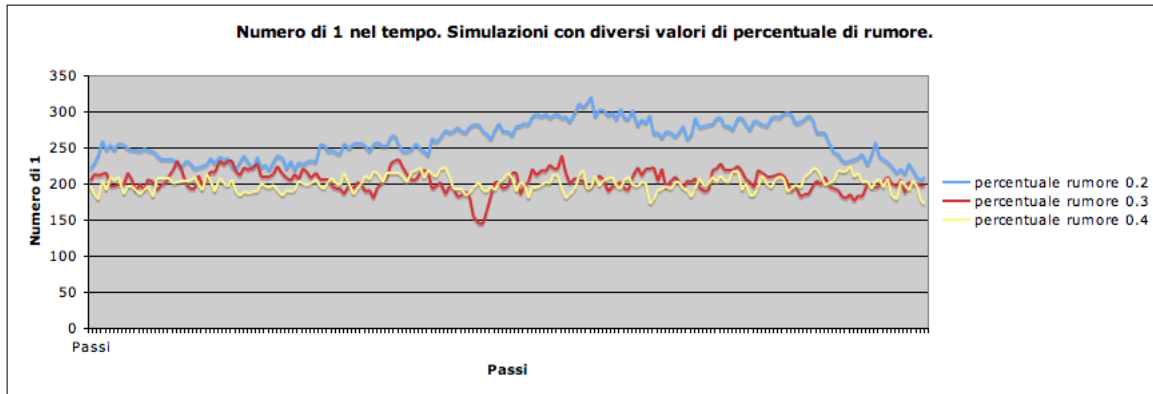


Figura 10: grafico del file storia.txt. Simulazioni con bassi valori di percentuale_rumore.

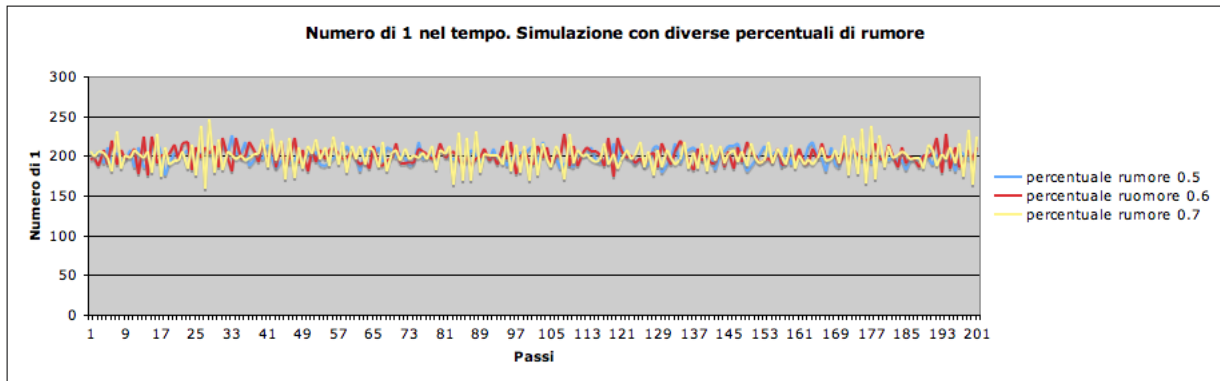


Figura 11: grafico del file storia.txt. Simulazioni con alti valori di percentuale_rumore.

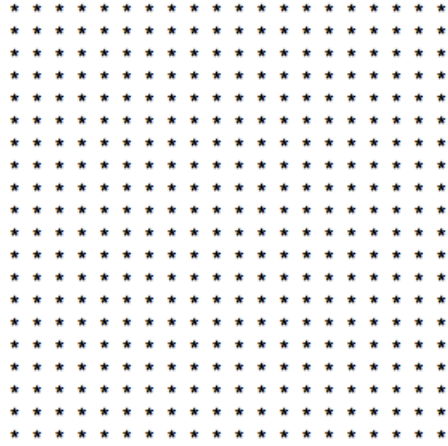
Settando il parametro a 0.8 e 0.9 inizia a verificarsi un fenomeno particolare in cui, dopo un certo numero di passi, due matrici di stato successive appaiono una l'opposto dell'altra. In realtà le due matrici non sono esattamente opposte, ma questa particolarità preannuncia il comportamento del sistema con *percentuale_rumore* 1.

***percentuale_rumore* uguale a 1**

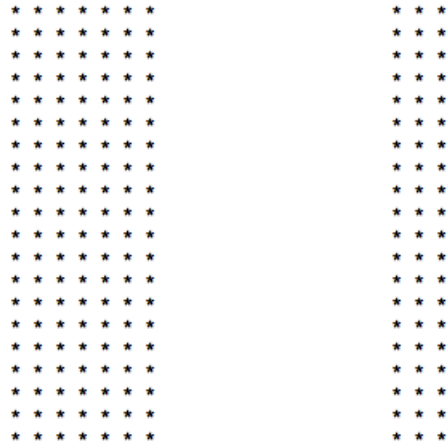
In queste simulazioni facciamo in modo che il comportamento contrario sia la regola che determina l'evoluzione del sistema. I risultati che otteniamo sono i seguenti:

- il sistema arriva ad una configurazione finale caratterizzata da un stato oscillatorio di periodo 2, come in Figura 12.

- il sistema arriva ad una configurazione finale caratterizzata da un loose attractor, come mostrato in Figura 13.

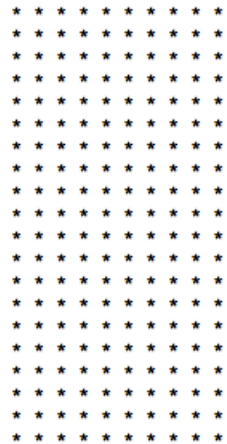


Step 124



Step 200

Step 125



*Figura 12: due esempi di stato oscillatorio con periodo 2.
 Simulazione con percentuale_rumore 1.*

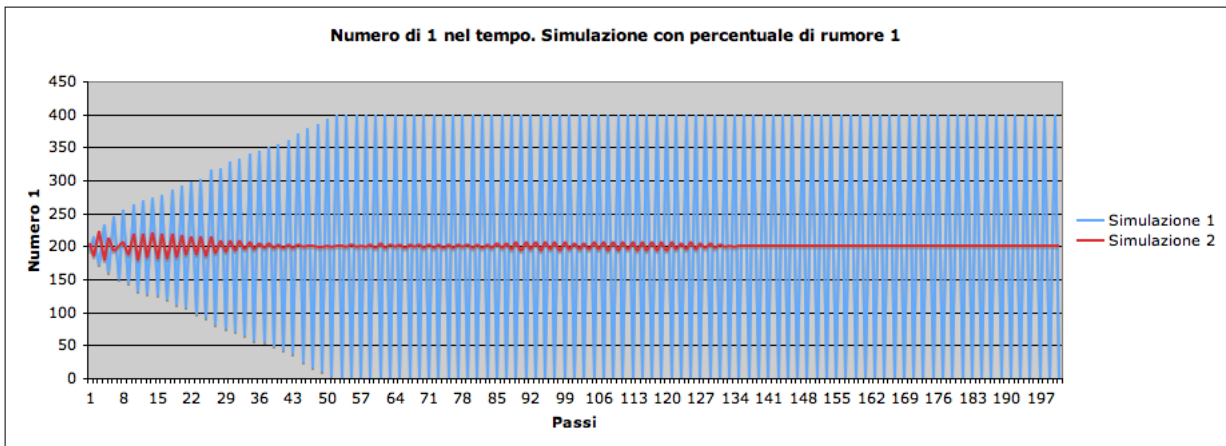


Figura 14: numero di 1 nel tempo. Simulazioni con percentuale_rumore 1.