Ambienti per Automi Cellulari (AC)

- Parecchi ambienti ad AC sono stati implementati su calcolatori sequenziali (PC, workstations)
- Due approcci possono essere individuati: via Hardware e via Software
- Il CAM (Cellular Automata Machine) rappresenta il più famoso esempio di architettura hardware dedicato per lo studio di AC (efficiente ma pochi stati…)
- Ambienti Software: P-CAM, PECANS, StarLogo, CAPE, CAMEL, CAMELot

Caratteristiche di Sistemi Paralleli AC

- Le prinicipli caratteristiche di un ambiente software parallelo dovrebbero essere:
  - Un layer di programmazione ad alto livello per il design di modello computazionali, indipendente dall'architettura parallela sottostante (e.g. CARPET)
  - Un'interfaccia GUI che permette la completa visualizzazione dell'evoluzione del fenomeno ed il display dei valori numerici associati alla simulazione
  - Strumenti di *tuning* e controllo che permettono lo *steering* (controllo globale) dell'evoluzione del fenomeno
  - Scalabilità per permettere una esecuzione efficiente del fenomeno su calcolatori paralleli (non ottenibile su calcolatori sequenziali!)

L'Ambiente CAMELOT

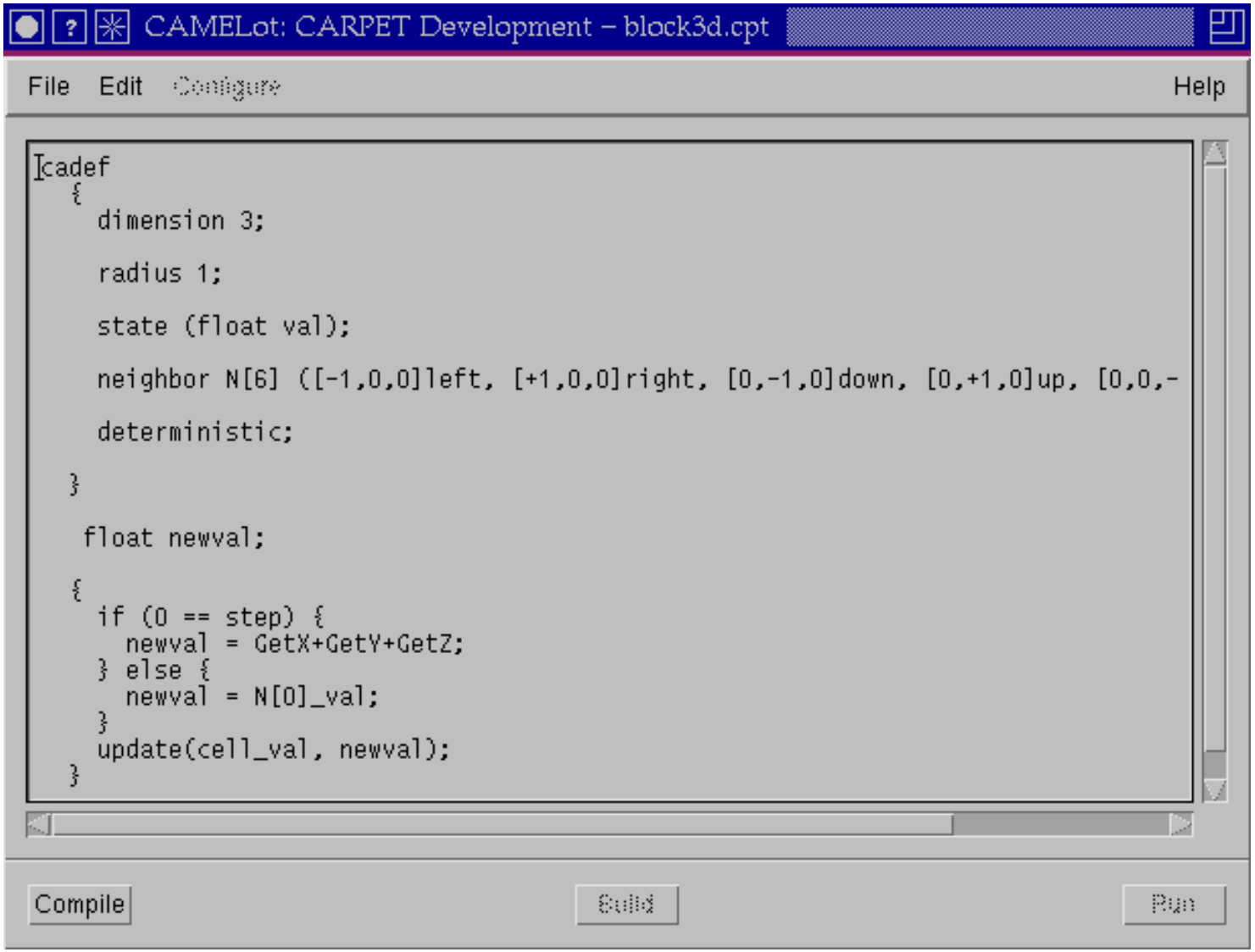
L'ambiente Camelot è un ambiente software per la programmazione ed esecuzione di Automi Cellulari

Camelot è composto da una interfaccia grafica, un linguaggio di supporto (CARPET) e di una finestra per il controllo della esecuzione dell'AC

Il parallelismo risulta invisibile all'utente

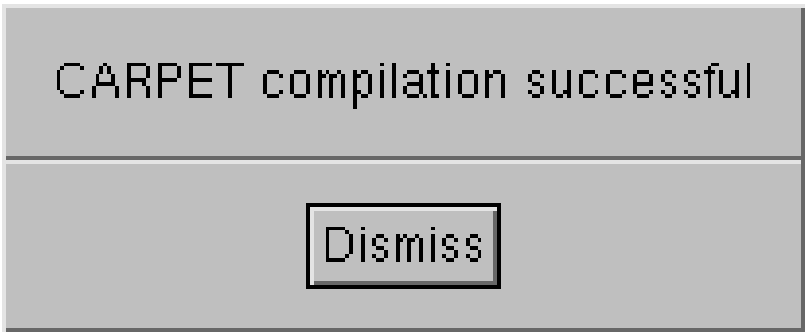
L'utente deve solamente specificare la funzione di transizione di una singola cella tramite un linguaggio ad alto livello (e.g. C, CARPET)

Finestra principale - Editor



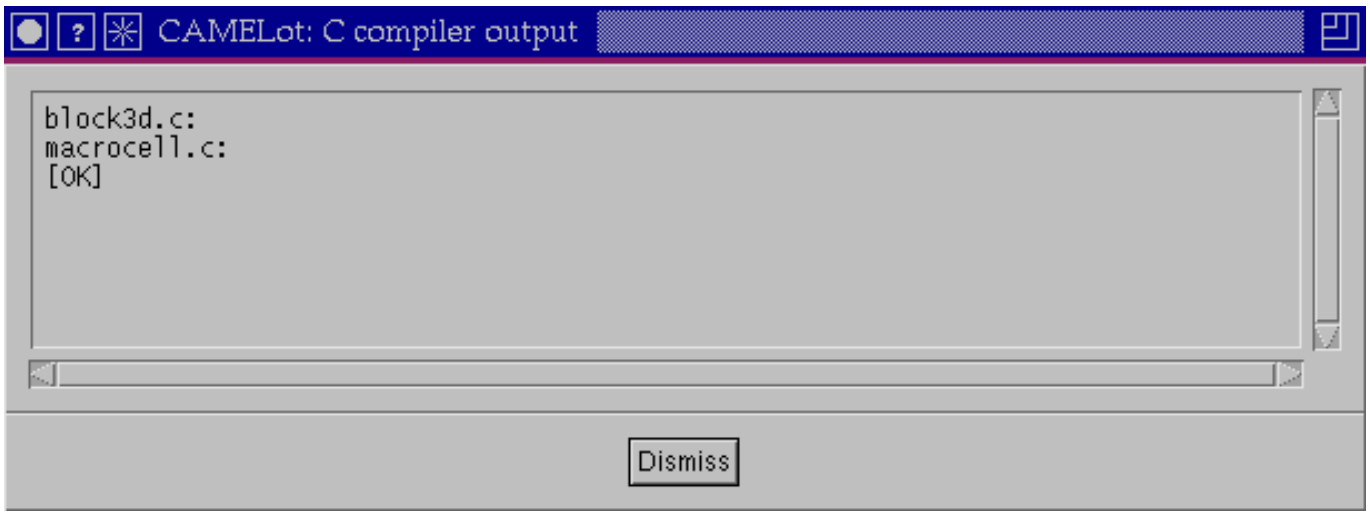
Compilazione

- Dopo aver scritto il programma in Carpet (*.cpt) bisogna compilarlo. Camelot genera un file *.c per essere successivamente linkato con varie libreria (MPI, etc) per ottenere un eseguibile.



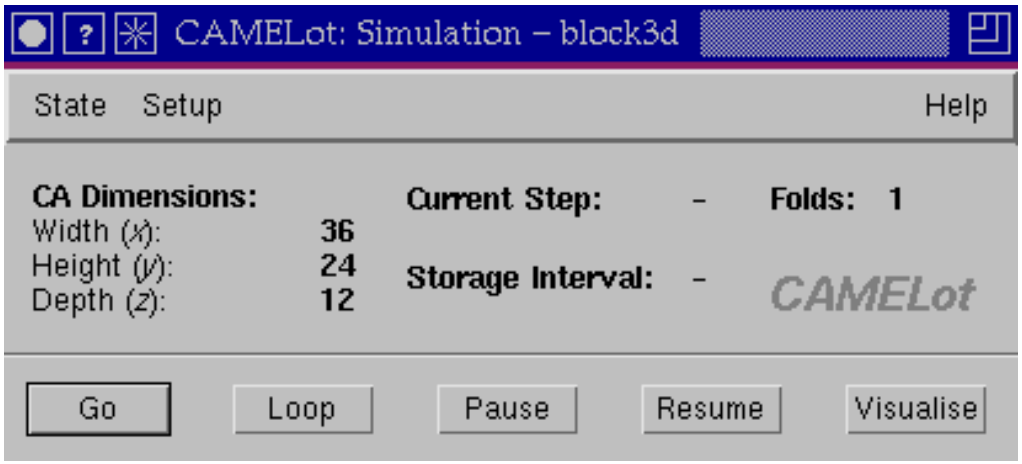
Build

- Per genera l'eseguibile bisogna specificare le dimensioni dell'AC (es. 100x100), i numeri di processori (1!), etc



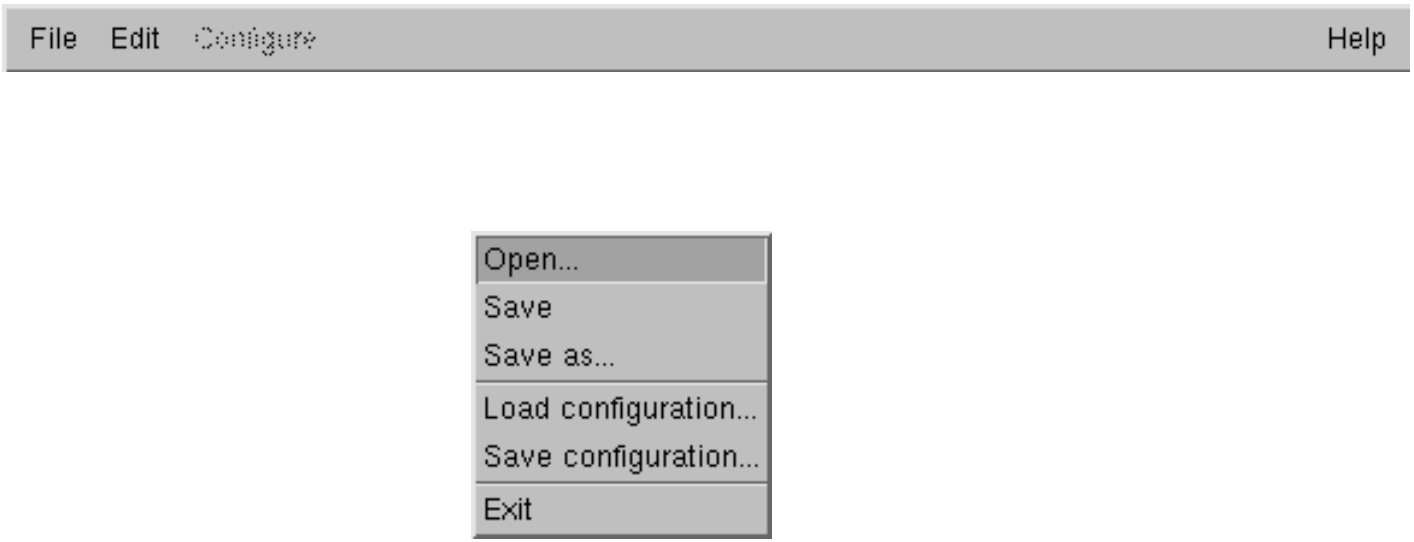
Esecuzione

- Se tutto è andato a buon fine, si fa girare l'eseguibile con RUN
- GO e LOOP eseguono l'AC per un certo numero di passi; PAUSE per pausa; RESUME per riprendere
- VISUALISE visualizza uno stato dell'AC



Menu FILE

Aprire, salvare, salva come, carica configurazione, salva configuarazione, esci

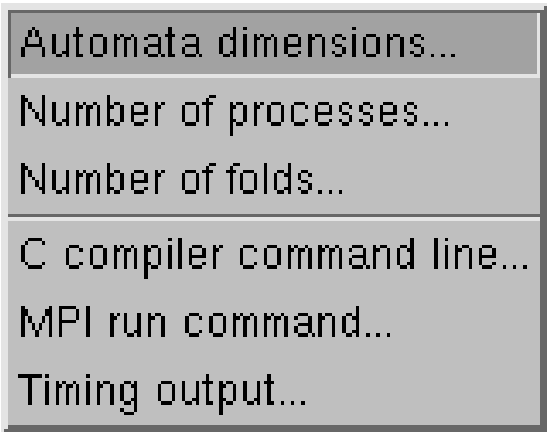


Menu EDIT

Cut, Copy, Paste, Find, Find Next, Replace

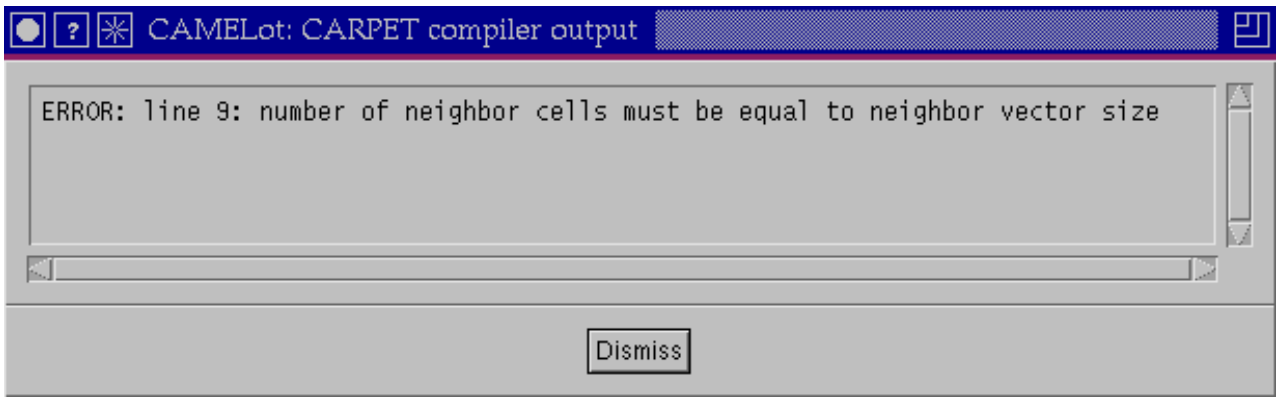
Menu CONFIGURE

Serve per "configurare" l'AC



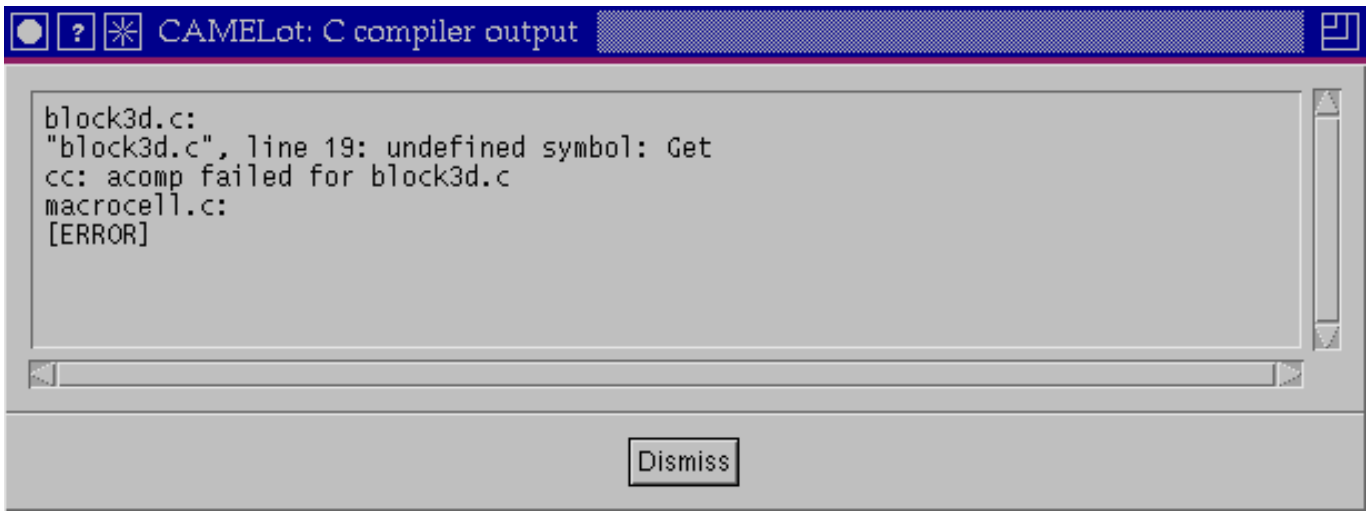
Errori di compilazione

- Vengono individuati se l'utente ha commesso errori di sintassi del programma CARPET



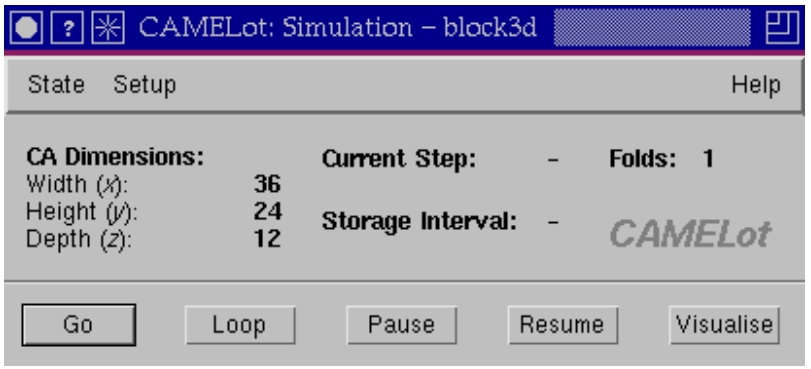
Errori di BUILD

- Il Build compila e fa il link del programma *.c (generato dalla compilazione) con il codice vero e proprio del motore CA



Finestra di Simulazione

- Facendo il Run viene visualizzata la finestra di simulazione



- Esso è composto da un menu, una parte descrittiva dell'AC e da vari bottoni.

Finestra di Simulazione

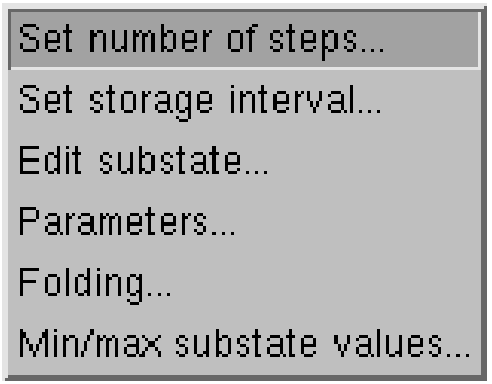
- Menu STATE: permette di caricare o salvare, singoli sottostati o configurazioni dell'AC in esame.
- Gli stati hanno estensione *.cmt
- Le configurazioni hanno estensione *.cpj
- Una tipica configuazione contiene informazioni quali, numero di dimensioni, numero di generazioni, numero di stati, etc

Finestra di Simulazione

Menu SETUP

Tra l'altro, permette di settare:

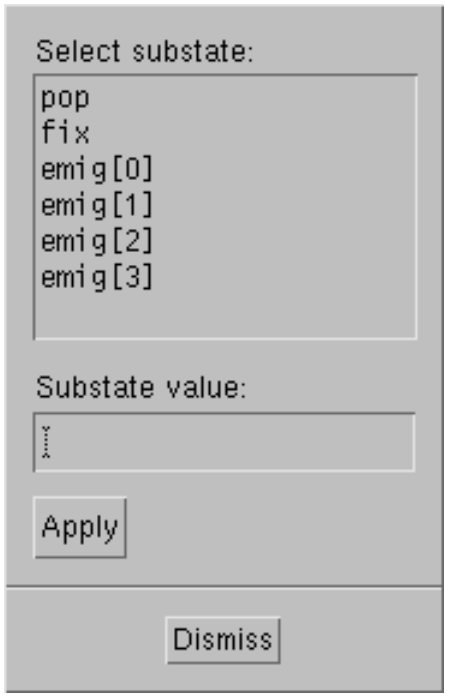
Il numero di passi da eseguire, numero di passi di salvataggio, editare uno stato, etc.



Finestra di Simulazione

- Editare uno stato

E' possibile editare (modificare) il valore dei singoli stati che compongono un'AC

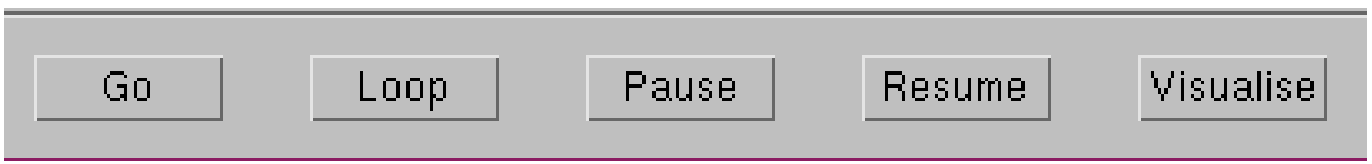


Finestra di Esecuzione

- In questa finestra vengono visualizzati le dimensioni dell'AC, il passo corrente, il passo di salvataggio dati e i folds (bilanciamento del carico)

Finestra di Esecuzione

- GO: VAI! …fino al passo specificato nel menu Setup
- LOOP: VAI! …per sempre (!)
- PAUSE: Pausa!
- RESUME: Ripristino!
- VISUALISE: Visualizza AC (1D, 2D, 3D)



Visualizzazione

- 1d CA: La visualizzazione contiene linee orizzontali che corrispondono all'AC. In orizzontale viene visualizzato il "trascorrere del tempo"
- 2D CA: Visualizzazione ortogonale semplice
- 3D CA: Possibilità di visualizzare gli stati in modo ortografico, oppure isometrico (l'utente fissa il piano x,y o z)

Linguaggio CARPET

- CARPET e' un linguaggio per la definizione di modelli AC e le loro funzioni di transizioni. E' basato su una estensione di ANSI C.
- L'utente deve specificare la funzione di transizione di **una** cella generica
- Possibilità di specificare diversi stati, vicinati, etc
- Le parti che compongono un programma Carpet sono:

1. Sezione dichiarazione AC
2. Funzione di Transizione
3. (opzionale) Funzione di ottimizzazione

Layout generale

cadef
{
declarations
}
[*transition function local variable declarations and subroutine prototypes*]
{
transition function code
}
[*transition function subroutines*]
[
steering
{
steering function code
}
]

La funzione di transizione

- E’ composto da istruzioni C (tranquilli, NON C++ !!!!)
- In aggiunta, le seguenti istruzioni CA:

cell_*substate*
DimX, DimY, DimZ
GetX, GetY, GetZ
NFolds
NProcs
random()
randomise()
srandom()
step
update()
parameter references

Definizione di AC - Esempio

```
cadef
{
     declaration;
     declaration;
      ...
     declaration;
}
```

dove *declaration* può essere:

```
deterministic  //(AC "deterministici")

dimension  // numero di dimensioni dell'AC

neighbour  // vicinato

parameter  // lista parametri globali

radius     // raggio vicinato

region     // serve nell'ottimizzazione

state      // stati (char, int, float, double,
                     (array di…)

threshold  // serve nell'ottimizzazione
```

Definizione di AC - Esempio

```
cadef
{
        dimension 3;
        radius 1;
        region Inside (start+1:end-2, :,:);
        state (float val; int val2);
        neighbour N[6] ([-1,0,0]left,[1,0,0]right,
         [0,-1,0]down,[0,1,0]up,[0,0,-1]in, [0,0,1]out);
        parameter (pi 3.14159);
        deterministic;
        threshold (cell_val == 3);
}
```

Definizione di AC

```
dimension <n>
radius <n>
state <substate_list>
neighbor <neigh_name> <pattern>
parameter <parameter_list>

ES:

state (short temp,quota; float volume);
neighbor Neumann[4]([0,-1]North,[-1,0]West, [0,1]South,
[1,0]East);
```

Funzione di transizione

Si scrive in C, ma per *accedere* ai valori degli stati (sottostati) dell'AC si usa l'struzione `cell_sottostato`

Es:

```
cadef
{
 state (float temp);
}

float val;
val = cell_val+3;
```

Funzione di transizione

- Per *aggiornare* il valore di un sottostato si usa:

```
update(cell_substate, value)
```

Es:

```
val=10;
update(cell_stato, val);
```

Funzioni random

- random(n)

Ritorna un numero casuale tra 0 e n

- randomise()

Genera un nuovo seed

- srandom(seed)

Come sopra, ma l'utente specifica il seed

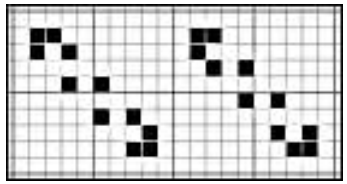
Il Gioco della Vita

- Il gioco della vita di Conway (Matematico inglese), forse il più famoso tra gli "automi cellulari", si è rivelato un vero e proprio universo in miniatura. Nonostante l'estrema semplicità delle leggi che lo regolano, sulla griglia dell'universo di Conway è possibile assistere ad un ribollire di configurazioni caotiche, dalle quali talvolta affiorano forme di vita anche estremamente complesse.

- Una cella accesa rimane accesa se nelle 8 celle adiacenti ci sono 2 o 3 celle accese, si spegne se ce ne sono di più o di meno.
- Una cella spenta si accende se nelle 8 celle adiacenti ci sono 3 celle accese, rimane spenta se ce ne sono di più o di meno.

- S23/B3

- Con grande stupore, configurazioni più o meno casuali di pochi pixel si sono rivelate vere e proprie forme di vita, alcune capaci di muoversi, alcune addirittura di riprodursi (caos).



Rasoio (periodo 2)

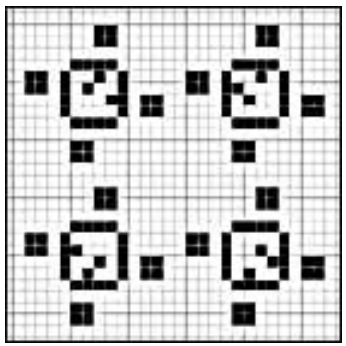
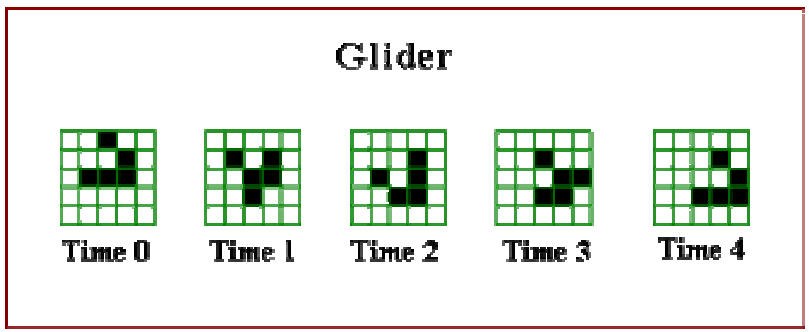


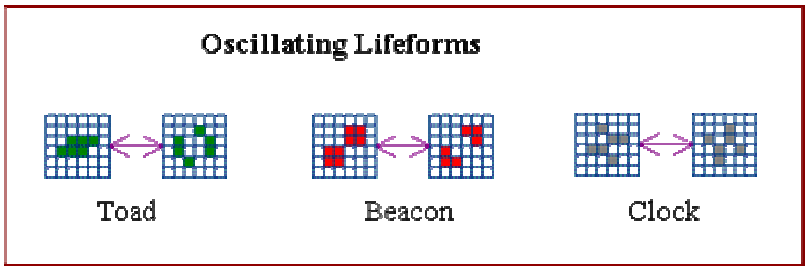
Tavola da biliardo (periodo 2)

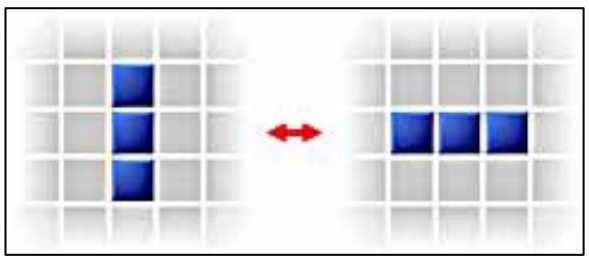
http://psoup.math.wisc.edu/Life32.html

Oscillatori

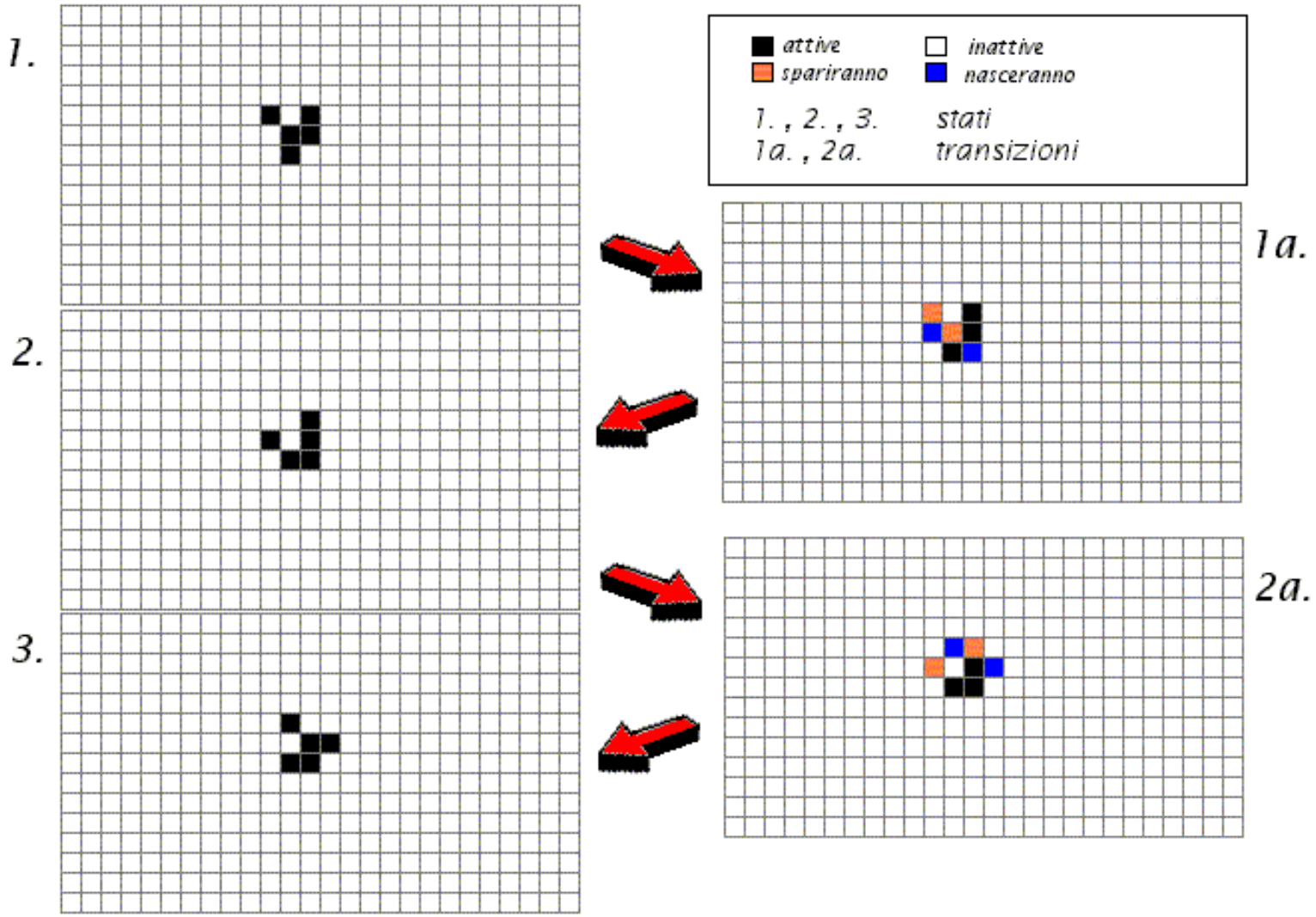
Nel Gioco della Vita compaiono dei modelli che hanno dei comportamenti particolari. Gli oscillatori sono delle figure che assumono negli istanti successivi solo due configurazioni una consecutiva all'altra all'infinito.







Glider



Gioco della Vita – Definizione CA


```
/* Conway's Game of Life implementation
 *
 * caricare life100_100rand!!!!!
 *
 */
#define alive 1
#define dead 0
cadef
  {
    dimension 2;
    radius 1;
    state ( int life );
    neighbor Moore[] ( [ 0,-1], [-1,-1], [-1, 0], [-1,  1],
                       [ 0, 1], [ 1, 1], [ 1, 0], [ 1, -1]  );
}
```

Gioco della Vita – Funzione di Transizione

```
int i;
int sum;
{
  sum = 0;
  for( i=0 ; i<8; i++)
      sum = Moore[i]_life + sum;
  if ( sum == 3  || ( sum == 2 && cell_life == 1) )
      update (cell_life, alive);
  else
      update (cell_life, dead);
}
```

Configurazioni iniziali

- Una volta compilato il programma, si può eseguire una configurazione iniziale dell'AC in due modi:

1. Tramite settaggio esplicito dei parametri e dei sottostati (edit substate)
2. Caricare una configurazione iniziale (valori iniziali di parametri + sottostati), generato tramite un programma C.

Esempio di generatore di configurazione (Life)

- Il programma deve generare due tipi di file, un *.cpj e tanti *.cmt per quanti sono gli stati dell'AC (Es. Life (1 stato!!!): un file `life1.cpj` e un file `life1.cmt`)
- Il file `life1.cpj` deve contenere 8 dati di tipo `int`, e cioè, in ordine:

1. Dimensioni AC
2. Dim x
3. Dim y
4. Dim z
5. Generazione (passo) AC
6. Numeri sottostati
7. Numero folds (1)
8. Numero parametri globali (OBBLIGATORIO!) + lista param

Esempio life1.cpj

```
…
#define dimx 100
#define dimy 100
#define dimz 1
…
FILE *fid;
dimensions=2;
generation=0;
n_sottostati=1;
n_fold=1;
n_parametri=1;
typedef struct {float zero;} Parametri;  //parametro fittizio, sigh!!!
Parametri prm;
…
fid=fopen("life1.cpj","wb");
fwrite(&dimensions,sizeof(int),1,fid);
fwrite(&d_dimx,sizeof(int),1,fid);
fwrite(&d_dimy,sizeof(int),1,fid);
fwrite(&d_dimz,sizeof(int),1,fid);
fwrite(&generation,sizeof(int),1,fid);
fwrite(&n_sottostati,sizeof(int),1,fid);
fwrite(&n_fold,sizeof(int),1,fid);
fwrite(&n_parametri,sizeof(int),1,fid);
fwrite(&prm.zero, sizeof(float),1,fid);

fclose(fid);
```

Esempio life1.cmt

```
…
int life[dimx][dimy];
…
FILE *fid;
…
for(i=0;i<dimx;i++)
   for(j=0;j<dimy;j++)
     life[i][j]=rand()%2  // inizializzazione random
…
fid=fopen("life1.cmt","wb");
for(j=0;j<dimy;j++){
  for(i=0;i<dimx;i++)
     fwrite(&life[i][j],sizeof(int),1,fid);  //stato di tipo int
  }
fclose(fid);
```