UNIVERSITAT POLITÈCNICA
DE
CATALUNYA

FACULTAT D'INFORMÀTICA

Interacció i Disseny d'Interfícies

Elements d'Autòmats Finits

Robert Joan Arinyo

Barcelona, setembre 2015

1 Introducció

Els *autòmats finits*, també coneguts amb el nom de *màquines d'estat finites*, són una eina que permeten definir models matemàtics de comportament de sistemes en diverses àrees del coneixement tals com enginyeria de control, l'enginyeria elèctrica i l'electrònica, ciències de la computació, matemàtiques, biologia, lingüística o filosofia.

Nosaltres utilitzarem els autòmats finits de manera anàloga a l'ús que en fan a l'enginyeria de control per modelar el comportament de **sistemes reactius**, és a dir, aquells sistemes que actuen per reacció davant d'accions tant internes com externes. Oi que els jocs per computador i videojocs responen a aquesta idea bàsica?

Les màquines d'estats finites han estat durnat molts anys un instrument cabdal usat pels programadors de jocs per a computadors per tal de donar la il.lusió de que els programes resultants eren intel.ligents. La majoria dels jocs existents fins avuí dia, han estat desenvolupats sobre màquines d'estats finites i, malgrat la volada que estan agafant altres tècniques, segurament encara seran l'eina més usada durant molts anys. Algunes raons que justifiquen aquesta opinió són:

- Les màquines d'estat finites es codifiquen molt fàcilment en llenguatges de programació. Hi ha diverses maneres de fer-ho i totes elles són raonablement simples d'implementar.
- Posar-les a punt, és a dir, eliminar les errades de programació, resulta senzill. Com que el comportament dels agents es subdivideix en unitats simples, sempre que un agent actui de manera estranya, el programador pot reseguir la seqüència d'events que mena al comportament erroni i per tant pot identificar la part del codi que cal modificar.
- El sobrecost (*overhead*) que implica utilitzar una màquina d'estats finita és petit perquè, en general, no requereixen costosos processos de càlcul ni de decisió. Bàsicament, les màquines d'estat finites es construeixen com un conjunt de regles prefixades del tipus `si-això-aleshores-allò`.
- Són intuïtives. Resulta natural i convenient pels humans pensar que els sistemes estan a cada instant de temps en un estat ben definit, estat que es pot descriure de manera clara i concisa. A més, permeten aplicar de manera directa una tècnica ben coneguda en programació de computadors: l'anàlisi descendet. En efecete, aquesta tècnica permet subdividir un problema (el comportament d'un agent) en un conjunt de subproblemes cadascun dels quals representa un estat determinat de l'agent i, després, definir les regles per resoldre'ls independentment uns dels altres. De fet, no hi ha cap impediment perquè un estat d'una máquina d'estats finita sigui ell mateix una altra màquina d'estats finita. Aquest mateix argument permet els programadors discutir el disseny informàtic amb altres membres del projecte encara que no siguin informàtics.
- Les màquines d'estats finites són flexibles. En efecte, aquestes màquines es poden refinar fins a assolir els comportaments requerits pels dissenyadors. A més, poden

ampliar-se amb noves capacitats funcionals simplement afegint nous estats i noves regles de transició. Així mateix, permeten incorporar de manera simple i natural altres tècniques tals com lògica difusa i xarxes neuronals.

Cas que vulgueu conèixer opinions crítiques negatives sobre les màquines d'estats finites, llegiu l'article de la referència [2].

2 Definició

Un **autòmat finit** o **màquina d'estats finita**, és un model matemàtic d'un sistema que evoluciona en el temps i que resta definit per tres conjunts finits: estats, accions i transicions.

Un estat defineix la situació actual del sistema i, en general, emmagatzema informació del passat. Una acció és una descripció d'una activitat que cal que efectui l'autòmat en un moment donat. Una transició descriu el canvi d'estat de l'autòmat a una detereminada acció. Un canvi d'estat de l'autòmat sempre és el resultat d'una tarnsició però una transició no necessàriament comporta un canvi d'estat.

Les accions es classifiquen segons els següents tipus

- **d'entrada**: és aquella acció que s'executa quan l'autòmat entra en un estat.
- **de sortida**: és aquella acció que s'executa quan l'autòmat abandona un estat.
- **d'input**: és una acció que es desencadena depenent de l'estat actual i les condicions d'entrada de l'autòmat.
- **de transició**: és una acció que s'executa quan l'autòmat efectua una certa transició.

2.1 Definició matemàtica

Matemàticament, un autòmat finit és un sextuple $(\Sigma, \Gamma, S, s_0, \delta, \omega)$ on

1. Σ és un conjunt finit no buit de símbols, anomenat *alfabet d'entrada*.
2. Γ és un conjunt finit no buit de símbols, anomenat *alfabet de sortida*.
3. S és un conjunt finit no buit d'estats.
4. s_0 és un element d'S que defineix l'estat inicial.
5. δ és la funció de transició entre estats, $\delta : S \times \Sigma \to S$.
6. ω és la funció de sortida, $\omega : S \times \Sigma \to \Gamma$.

2.2 Definició intuïtiva

En el nostre context, resulta més útil donar una definició més intuïtiva. Podem dir que una màquina d'estats finita està definida per

1. Un conjunt d'events d'entrada.
2. Un conjunt d'events de sortida.
3. Un conjunt d'estats.
4. Una descripció de l'estat inicial.
5. Una funció que defineix la transició a seguir que depen de l'estat i l'event d'entrada.
6. Una funció que defineix la sortida generada per l'autòmat que depen de l'event d'entrada i de l'estat.

Tots els conjunts són finits i no buits.

És important entendre que un autòmat actua en instants determinats discrets de temps, $t = \{1, 2, \ldots, i, \ldots\}$. Suposem que a l'instant t_i l'autòmat es troba en un estat determinat, diguem-li s_i, està rebent una entrada e_i i generant un sortida o_i . Si ara es produeix un senyal d'entrada e_{i+1}, l'autòmat genera una sortida $o_{i+1} = \omega(s_i, e_{i+1})$ i passa a estar en un nou estat $s_{i+1} = \delta(s_i, e_{i+1})$.

3 Models de representació

Entre les diferents possibilitat existents, les dues tècniques més utilitzades per representar els autòmats finits són els diagrames d'estats i les taules.

Els diagrames d'estats son grafs on els estats es representen mitjançant cercles i les transicions mitjançant arestes orientades que uneixen l'estat de partida i l'estat destinació. Cada cercle s'etiqueta amb un identificador de l'estat que representa. Els estats inicials es representen amb dos cercles concentrics. Cada aresta s'etiqueta amb l'identificador de l'acció o event tal que causa la transició. La representació de la sortida es pot fer de dues maneres. Una representa sobre cada aresta de transició la parella entrada-sortida com *entrada/sortida*. Una altra representa l'identificador de la sortida dins del cercle d'estat, separada de l'estat per una ratlla horitzontal.

Per tal d'il.lustrar la tècnica del diagrama, considerem el procés ben habitual i ben conegut d'inici d'una sessió de treball en un computador (*login*). Veieu la Figura 1. Suposem una funcionalitat simplificada tal que, un cop donada la tensió elèctrica, el sistema operatiu resta en estat d'espera fins que l'usuari entra el nom d'usuari i la contrasenya. Quan s'introdueix un nom d'usuari, el sistema comprova que sigui correcte. Això és un estat. Si el nom d'usuari és correcte, aleshores el sistema demana i espera una contrasenya, això representa un altre estat. Si no és correcte, el sistema resta en l'estat inicial. Quan es dona una contrasenya vàlida, s'inicia la sessió de treball, un altre estat que no detallarem més.

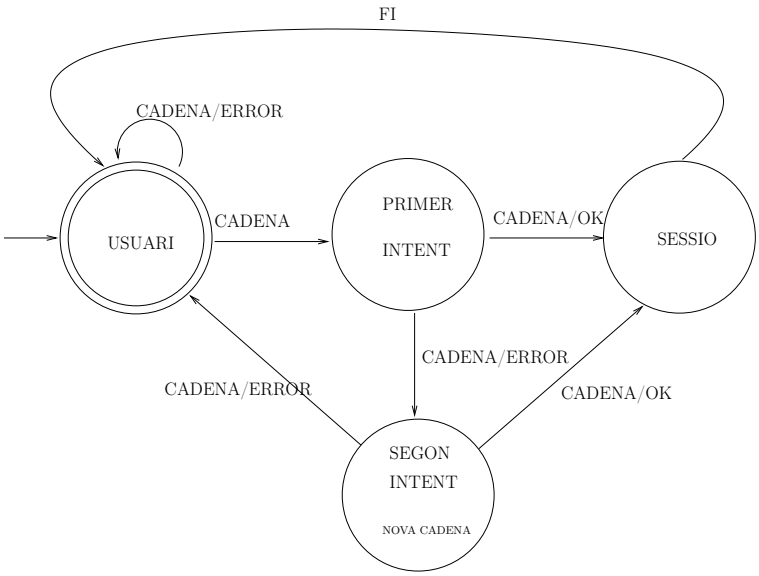

Figura 1: Diagrama d'estats del procès de *login*.

Si la contrasenya no és correcta, s'informa l'usuari i es passa a esperar un segon intent. Aquest serà un quart estat. Si la constrasenya torna a ser errònia, es passa a l'estat inicial on tot torna a començar. Quan la sessió acaba per indicació de l'usuari, el sistema retorna a l'estat inicial.

En la tècnica de taules de transicions d'estats, es representen en una dimensió d'una taula els estats i en l'altra dimensió els events. Cada cel.la definida per la parella (*estat, event*) emmagatzema la parella *acció/estat* que defineixen l'acció que cal desencadenar quan es podueix l'event i l'estat resultant. Les cel.les que no tenen valor assignat indiquen transicions inadmissibles. La Taula 1 mostra la definició corresponent a l'autòmat del diagrama de la Figura 1. Les accions indicades amb la lletra A són accions tals com tancar una finestra, mostrar un missatge d'error, etc, que resulten irrellevants ara. Si bé la representació en forma de diagrama de transicions resulta visualment més entenedora, sempre que el diagrama no sigui gaire gros, la representació en forma de taula resulta més adient a l'hora de programar els autòmats.

Event	NOM OK	NOM ERR	PASS OK	PASS ERR	FI
Estat Inicial (EI)	A/PI	A/EI	–	–	–
Primer Intent (PI)	–	–	A/S	A/SI	–
Segon Intent (SI)	–	–	A/S	A/I	
Sessió (S)	–	–	–	–	Tancar/I

Taula 1: Taula de transicions d'estats del procès de *login*.

4 Programació

Un cop han estat identificats els sis elements que conformen un autòmat, segons el que ha estat dit a la Secció 2.2, resulta senzill dissenyar algorismes que simulen l'evolució de l'autòmat al llarg del temps. Abans però, cal que fem alguns aclariments. Primer, els estats són únics i no poden haver dos estats que generin exactament la mateixa reacció per exactament els mateixos events. Segon, les transicions entre estats només estan permeses quan s'han completat totes les accions corresponents a l'estat actual. Finalment, atès que les màquines d'estats finites són deterministes, les reaccions a un event en un estat són sempre les mateixes. Això fa que, per exemple, després d'una serie d'execucions d'un joc, el jugador pugui deduir quina serà la reacció que obtindrà. Una possible solució és incloure diverses reaccions possibles dins d'un estat de manera que s'en desencadeni una o una altra en funció d'una determinada llei de comportament aleatori.

Descriurem una guia per a programar autòmats com a màquines d'estats finits amb un exemple. Els passos indicats són bàsicament els que cal seguir en general. Naturalment, els detalls de programació de tots els subprogrames auxiliars i la posada a punt de l'autòmat, requereixen la cura i temps propis de la programació de computadors.

5 Exemple

L'objectiu és simular en un computador els moviments d'un robot humanoide comandat mitjançant el teclat.

5.1 Descripció literal de l'autòmat

El robot, d'aparença humanoide podrà estar connectat o desconnectat i, en aquest segon cas, podrà parlar, caminar, còrrer, aturar-se i combinacions d'aquestes accions. Les ordres que podrem donar al robot seran:

- `Inicial`: Situació inicial del robot. Espera ser connectat.
- `Engega`: El robot passarà a la situació de connectat i, a partir d'ara, podrà rebre ordres.
- `Camina`: El robot marxarà caminant.
- `Corre`: El robot marxarà corrent.
- `Parla`: El robot parlarà (es mostrarà un text per pantalla).
- `Calla`: El robot callarà.
- `Aturat`: El robot aturarà la marxa.
- `Apaga`: El robot passarà a la situació inicial de desconnectat. Aquesta ordre tindrà efecte sigui quin sigui l'estat del robot.

- `Acaba`: S'acaba la simulació. Aquesta ordre tindrà efecte sigui quin sigui l'estat del robot.

En aquestes condicions, identifiquem els possibles estats següents: inicial, engegat, caminant, corrent, aturat, engegat-i-parlant, caminant-i-parlant, corrent-i-parlant, aturat-i-parlant, estat final.

5.2 Dibuixa el diagrama de transicions

La Figura 2 representa el diagrama de transicions per l'exemple del robot. Notis la complexitat del dibuix tot i que el sistema no és excessivament complex i que s'han omès la majoria de les arestes corresponents a les transicions `acaba` entre qualsevol estat i l'estat final i a la d'`apaga` entre qualsevol estat i l'estat inicial.

5.3 Defineix la taula de transicions

Per tal de definir la taula de transicions, és a dir, la funció $s_{i+1} = \delta(s_i, e_{i+1})$, analitzarem per a cada parella *(estat, event)*, quin és el nou estat al que cal que passi l'autòmat. La Taula 2 mostra el resultat de la nostra anàlisi del problema del robot. Notis que l'estat final no admet entrades, com cal que sigui, atès que aquest estat significa la fi de l'evolució de l'autòmat i, un cop assolit, s'ha acabat la simulació.

Una qüestió a tenir en compte és que, en general, la taula de transicions d'estats d'un autòmat pot definir una funció parcial, és a dir, que hi hagi parelles *(estat, event)* que no defineixen una sortida. De vegades vol dir que aquesta situació no es pot presentar i d'altres que cal afegir un nou estat d'excepcionalitat o error.

	Apaga	Engega	Camina	Corre	Parla	Calla	Aturat	Acaba
Inicial	In	En	In	In	In	In	In	Fi
Engegat	In	En	Ca	Co	En-i-p	En	At	Fi
Caminant	In	Ca	Ca	Co	Ca-i-p	Ca	At	Fi
Corrent	In	Co	Co	Co	Co-i-p	Co	At	Fi
Aturat	In	At	Ca	Co	AT-i-p	At	At	Fi
Engegat-i-parlant	In	En-i-p	Ca-i-p	Co-i-p	En-i-p	En	A-i-p	Fi
Caminant-i-parlant	In	Ca-i-p	Ca-i-p	Co-i-p	Ca-i-p	Ca	A-i-p	Fi
Corrent-i-parlant	In	Co-i-p	Ca-i-p	Co-i-p	Co-i-p	Co	A-i-p	Fi
Aturat-i-parlant	In	At-i-p	Ca-i-p	Co-i-p	At-i-p	At	A-i-p	Fi
Final	–	–	–	–	–	–	–	–

Taula 2: Taula de transicions d'estats pel robot.

5.4 Defineix la taula de sortides

Caldria ara definir la taula de sortides, és a dir, la funció $o_{i+1} = \omega(s_i, e_{i+1})$. De manera anàloga al cas de les transicions, per a cada parella *(estat, event)*, analitzariem quines són

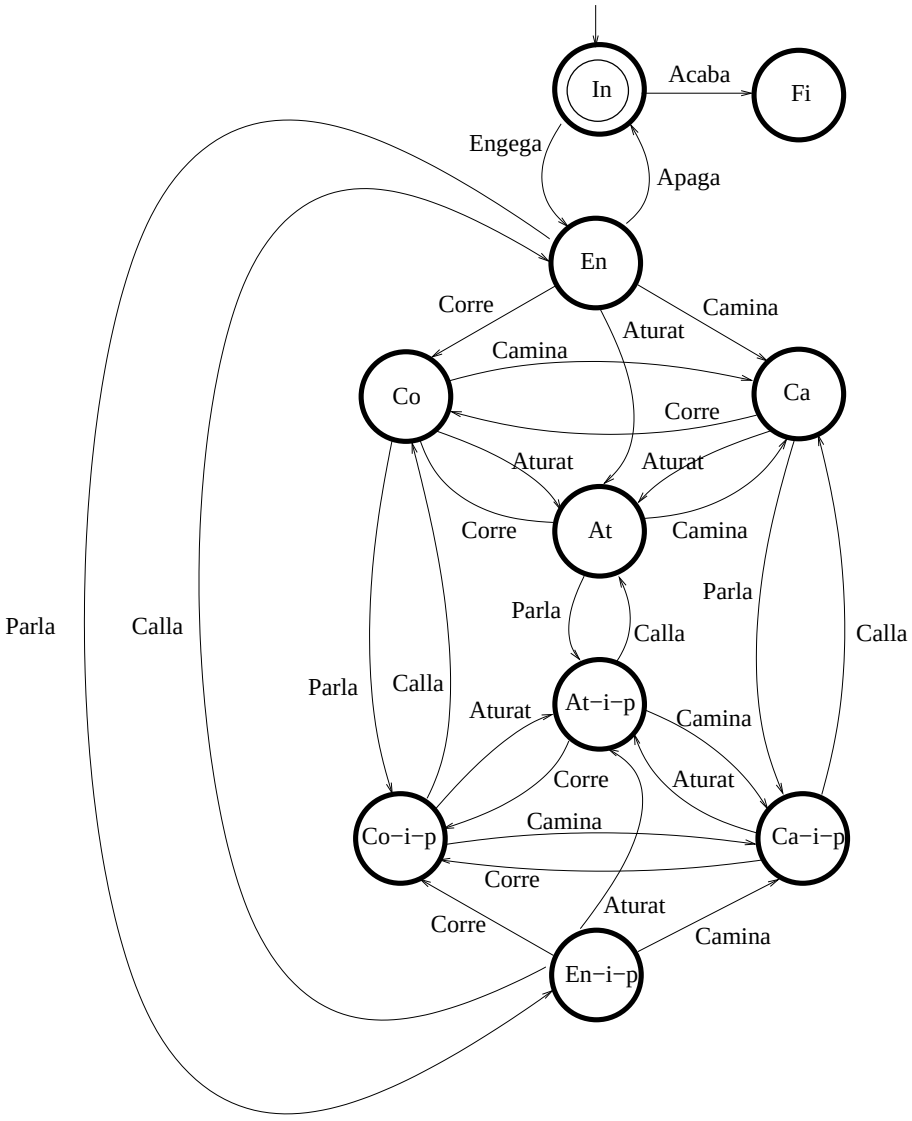

Figura 2: Diagrama (parcial) de transicions d'estats pel robot.

les accions que caldrà que efectui l'autòmat abans de passar a l'estat següent. Atès que les entrades a la taula de sortides són exactament les mateixes que a la taula de transicions, de vegades les sortides s'especifiquen a la mateixa taula, tal com hem fet al cas de la Figura 1. Ara bé, per qüestions de claredat, en general resulta més pràctic construir taules diferents.

Un altre factor a tenir en compte a l'hora de dissenyar la taula de sortides és que la sortida pot ser arbitrariament complexa, per exemple, desencadenar un altre autòmat. Aleshores, des del punt de vista de programació de computadors, el que s'acostuma a fer és considerar que, en general, per cada parella *(estat, event)* hi haurà un únic subprograma que en farà el tractament. Posteriorment, a aquest subprograma se li aplicaran les tècniques usuals de programació de computadors, per exemple anàlisi descendent.

La Taula 3 mostra una part de la taula de sortides per l'exemple del robot. Notis que no dona gaire informació explícita. Aquest fet fa que no sempre es defineixi, sinó que directament es fa l'anàlisi informàtica de cada acció de sortida.

	Apaga	Engega	Camina	Calla
Inicial	Fer res	Engega	Fer res	Fer res
Engegat	Apaga	Fer res	Camina	Fer res
Caminant	Apaga	Camina	Camina	Camina
Corrent	Apaga	Corre	Camina	Corre
Corrent-i-parlant	Apaga	Corre-i-parla	Camina-i-parla	Corre-i-parla
Aturat-i-parlant	Apaga	Aturat-i-parla	Camina-i-parla	Aturat-i-parla
Final	–	–	–	–

Taula 3: Taula parcial de sortides pel robot.

5.5 Codifica l'autòmat

El codi en llenguatge **C** que representa un autòmat té diversos components. Primer cal tenir una unitat de codi que capturi les entrades o events. Suposem que sigui el codi representat per l'acció

```
nouEvent(tEvent *event)
```

Ara cal una unitat de codi que, per a cada estat previst, tracti cadascun dels events possibles. Aquesta unitat s'acostuma a conèixer amb el nom anglès de *dispatcher*, perquè *despatxa* els events de manera anàloga a com el dependent d'una botiga despatxa les comandes dels clients. Com que l'estat inicial de l'autòmat sempre està definit *a priori*, el *dispatcher* té la forma

```
tEstat estat = ESTAT_INICIAL;

int main()
{
  tEvent event;

  while( estat != ESTAT_FINAL )
  {
    switch( estat )
    {
      case ESTAT_INICIAL:
        estatInicial(event);
      break;

      case ESTAT_1:
        estat1(event);
      break;

      case ESTAT_2:
        estat2(event);
      break;
      ...
      case ESTAT_N:
        estatN(event);
      break;
    }
    nouEvent(&event);
  }
  estatFinal();

  return 0;
}
```

Ara, per a cada acció de sortida associada a un estat, s'erfectuarà l'anàlisi corresponent en funció de l'event concret que ha entrat a l'autòmat. Si la taula `nouEstat[estat][event]` emmagatzema la funció de transició d'estats, un patró de programa que despatxa l'event `event` si l'estat és `estatX`, és com segueix

```
void estatX(tEvent event)
{
  switch (event)
  {
    case EVENT_1:
      tractaEvent1();
    break;

    case EVENT_2:
      tractaEvent2();
    break;
    ...
    case EVENT_M:
      tractaEventM();
    break;
  }
  estat = nouEstat[estat][event];
}
```

A partir d'ací, cada acció de tractament d'event s'analitzarà aplicant les tècniques que heu après en les assignatures de programació de computadors.

6 Codi pel robot

Tot seguit teniu un programa complet pel cas de l'autòmat del robot, codificat en llenguatge **C** i preparat per ser compilat i executat en un computador explotat amb el sistema operatiu XP de Microsoft.

En execució, el programa presenta diverses deficiències. Una és que el garfisme resulta poc realista. La deficiència deriva del fet que la llibreria gràfica disponible quan la pantalla està en mode `text`, només ofereix primitives *caràcter*, és a dir, el grafisme més petit que permet definir és una matriu rectangular de pixels, per exemple de 7×5 pixels.

Un altre problema resulta d'haver de barrejar informació gràfica, els dibuixos, i informació textual, les frases que diu el robot. En general, aquesta barreja resulta difícil de governar.

Finalment, simular efectes cinemàtics d'escenes gràfiques amb eines de programació de computadors sense processadors específics, poques vegades permet obtenir una freqüència mínima de refresc de 24 imatges per segon, valor que cal assolir per donar sensació visual de continuïtat.

```
//
// Interaccio i Disseny d'Interficies
// UPC - FIB, R. Joan-Arinyo, Setembre 2015
//
```

```
//              ... Automats ...
//
// Versio amb esborrat explicit de l'escena

// Includes de llibreries

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <Windows.h>



// Control de la pantalla

#define PANTALLA_80x50       C4350
#define FINESTRA_W             80  // Amplada finestra del joc
#define FINESTRA_H             50  // Alsaria finestra del joc
#define LLARGADA               48  // FINESTRA_H - 2
#define ALSADA                 50
#define VELOCITAT_CAMINA        2
#define VELOCITAT_CORRE         8
#define VELOCITAT_FRASE        10
#define DELAY_SINC            400 // Temporitzacio per sincronisme

#define CANVI_H                 1  // Maxima variacio de l'horitzo
#define MINIM_H                20  // Maxima vall de l'horitzo
#define VORERA                  8
#define OFFSET                  4  // Offset a terra pel robot
#define ROBOT_W                 8  // Amplada imatge robot
#define ROBOT_H                19  // Alsada imatge robot
#define NIMATGES                4  // Nombre d'imatges del robot

#define LLEGENDA_X              2
#define LLEGENDA_Y (FINESTRA_H - 2)


// Codis dels grafismes

#define DIAMANT            0x04
#define COR                0x03
#define CARA               0x02
#define QSOLID             0xDB
#define QRATLLAT           0xB1
```

```
// Codis de colors

#define COLOR_HORITZO       BLUE
#define COLOR_VORERA   LIGHTGRAY

#define COLOR_LLEGENDA    YELLOW

#define COLOR_PADDING         -1
#define COLOR_ROBOT            0
#define COLOR_BRAS1            1
#define COLOR_BRAS2            2
#define COLOR_CAMA1            3
#define COLOR_CAMA2            4


// Codis ASCII tecles d'events

#define TECLA_APAGA   65   // A
#define TECLA_ENGEGA  69   // E
#define TECLA_CAMINA  67   // C
#define TECLA_CORRE   75   // K
#define TECLA_PARLA   80   // P
#define TECLA_CALLA   76   // L
#define TECLA_ATURAT  83   // S
#define TECLA_ACABA   70   // F

// Index per les frases del robot

#define FRASE_BLANC  0
#define FRASE_CONNEC 1
#define FRASE_CAMINA 2
#define FRASE_CORRER 3
#define FRASE_ATURAT 4


// Definicio de tipus

typedef enum {
        ESTAT_INICIAL,
        ESTAT_ENGEGAT,
        ESTAT_CAMINANT,
        ESTAT_CORRENT,
        ESTAT_ATURAT,
        ESTAT_ENGEGAT_PARLANT,
        ESTAT_CAMINANT_PARLANT,
```

```
        ESTAT_CORRENT_PARLANT,
        ESTAT_ATURAT_PARLANT,
        ESTAT_FINAL
}tEstat;


typedef enum {
        EVENT_APAGA,
        EVENT_ENGEGA,
        EVENT_CAMINA,
        EVENT_CORRE,
        EVENT_PARLA,
        EVENT_CALLA,
        EVENT_ATURAT,
        EVENT_ACABA,
        EVENT_NUL
}tEvent;


// Variables

/*
    Funcio de transicio
    files    : estats
    columnes : events
    (f, c)   : nou estat

           0 Apaga 1 Engega  2 Camina 3 Corre  4 Parla 5 Calla 6 Aturat 7 Acaba
0 Inicial   In       En       In       In       In      In      In       Fi
1 Engegat   In       En       Ca       Co       En-i-p  En      A        Fi
2 Caminant  In       Ca       Ca       Co       Ca-i-p  Ca      A        Fi
3 Corrent   In       Co       Co       Co       Co-i-p  Co      A        Fi
4 Aturat    In       A        Ca       Co       A-i-p   A       A        Fi
5 En-i-par  In       En-i-p   Ca-i-p   Co-i-p   En-i-p  En      A-i-p    Fi
6 Ca-i-par  In       Ca-i-p   Ca-i-p   Co-i-p   Ca-i-p  Ca      A-i-p    Fi
7 Co-i-par  In       Co-i-p   Ca-i-p   Co-i-p   Co-i-p  Co      A-i-p    Fi
8 At-i-par  In       A-i-p    Ca-i-p   Co-i-p   A-i-p   A       A-i-p    Fi
9 Final     --       --       --       --       --      --      --       --
*/

static tEstat nouEstat[10][8] = {
  { 0,  1,  0,  0,  0,  0,  0,  9},
  { 0,  1,  2,  3,  5,  1,  4,  9},
  { 0,  2,  2,  3,  6,  2,  4,  9},
  { 0,  3,  2,  3,  7,  3,  4,  9},
  { 0,  4,  2,  3,  8,  4,  4,  9},
```

```
  { 0,  5,  6,  7,  5,  1,  8,  9},
  { 0,  6,  6,  7,  6,  2,  8,  9},
  { 0,  7,  6,  7,  7,  3,  8,  9},
  { 0,  8,  6,  7,  8,  4,  8,  9}
};


// Imatges predefinides del robot

static int imatgesRobot[5][19][8] = {
    {
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1, -1, -1, -1, -1, -1},
      {-1, -1, -1,  0,  0, -1, -1, -1},
      {-1, -1, -1,  0,  0,  0, -1, -1},
      {-1, -1, -1,  0,  0, -1, -1, -1},
      {-1, -1,  0,  0,  0,  0, -1, -1},
      {-1, -1,  0,  1,  0,  0, -1, -1},
      {-1, -1,  0,  1,  0,  0, -1, -1},
      {-1, -1,  0,  1,  0,  0, -1, -1},
      {-1, -1,  0,  0,  0,  0, -1, -1},
      {-1, -1,  0,  0,  0,  0, -1,  1},
      {-1, -1,  0,  0,  0,  0,  1,  1},
    },
   {
      {-1, -1, -1,  0,  0, -1, -1, -1},
      {-1, -1, -1,  0,  0,  0, -1, -1},
      {-1, -1, -1,  0,  0, -1, -1, -1},
      {-1, -1, -1,  0, -1, -1, -1, -1},
      {-1, -1, -1,  0, -1, -1, -1, -1},
      {-1, -1, -1,  0, -1, -1, -1, -1},
      {-1, -1,  0,  0,  0,  0, -1, -1},
      {-1, -1,  0,  1,  0,  0, -1, -1},
      {-1, -1,  0,  1,  0,  0, -1, -1},
      {-1, -1,  0,  1,  0,  0, -1, -1},
      {-1, -1,  0,  0,  0,  0, -1, -1},
      {-1, -1,  0,  0,  0,  0, -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, -1},
  {-1, -1, -1,  1, -1, -1, -1, -1},
  {-1, -1, -1,  1, -1, -1, -1, -1},
  {-1, -1, -1,  1, -1, -1, -1, -1},
  {-1, -1, -1,  1,  1, -1, -1, -1}
},
{
  {-1, -1, -1,  0,  0, -1, -1, -1},
  {-1, -1, -1,  0,  0,  0, -1, -1},
  {-1, -1, -1,  0,  0, -1, -1, -1},
  {-1, -1, -1,  0, -1, -1, -1, -1},
  {-1, -1, -1,  0, -1, -1, -1, -1},
  {-1, -1, -1,  0, -1, -1, -1, -1},
  {-1, -1,  0,  0,  0,  0, -1, -1},
  {-1, -1,  0,  1,  0,  0, -1, -1},
  {-1, -1,  1,  0,  0,  0,  2, -1},
  {-1,  1,  0,  0,  0,  0, -1,  2},
  {-1, -1,  0,  0,  0,  0, -1, -1},
  {-1, -1,  0,  0,  0,  0, -1, -1},
  {-1, -1,  0,  0,  0,  0, -1, -1},
  {-1, -1, -1,  3, -1, -1, -1, -1},
  {-1, -1, -1,  3,  4, -1, -1, -1},
  {-1, -1, -1,  3,  4, -1, -1, -1},
  {-1, -1,  3, -1, -1,  4, -1, -1},
  {-1, -1,  3, -1, -1,  4, -1, -1},
  {-1, -1,  3,  3, -1,  4,  4, -1}
},
{
  {-1, -1, -1,  0,  0, -1, -1, -1},
  {-1, -1, -1,  0,  0,  0, -1, -1},
  {-1, -1, -1,  0,  0, -1, -1, -1},
  {-1, -1, -1,  0, -1, -1, -1, -1},
  {-1, -1, -1,  0, -1, -1, -1, -1},
  {-1, -1, -1,  0, -1, -1, -1, -1},
  {-1, -1,  0,  0,  0,  0, -1, -1},
  {-1, -1,  0,  1,  0,  0, -1, -1},
  {-1, -1,  0,  1,  0,  0, -1, -1},
  {-1, -1,  0,  1,  0,  0, -1, -1},
  {-1, -1,  0,  0,  0,  0, -1, -1},
  {-1, -1,  0,  0,  0,  0, -1, -1},
  {-1, -1,  0,  0,  0,  0, -1, -1},
  {-1, -1, -1,  3, -1, -1, -1, -1},
  {-1, -1, -1,  3, -1, -1, -1, -1},
  {-1, -1, -1,  3, -1, -1, -1, -1},
  {-1, -1, -1,  3, -1, -1, -1, -1},
  {-1, -1, -1,  3, -1, -1, -1, -1},
```

```
      {-1, -1, -1,  3,  3, -1, -1, -1}
    },
    {
      {-1, -1, -1,  0,  0, -1, -1, -1},
      {-1, -1, -1,  0,  0,  0, -1, -1},
      {-1, -1, -1,  0,  0, -1, -1, -1},
      {-1, -1, -1,  0, -1, -1, -1, -1},
      {-1, -1, -1,  0, -1, -1, -1, -1},
      {-1, -1, -1,  0, -1, -1, -1, -1},
      {-1, -1,  0,  0,  0,  0, -1, -1},
      {-1, -1,  0,  1,  0,  0, -1, -1},
      {-1, -1,  0,  0,  1,  0, -1, -1},
      {-1,  2,  0,  0,  0,  1,  1, -1},
      {-1, -1,  0,  0,  0,  0, -1, -1},
      {-1, -1,  0,  0,  0,  0, -1, -1},
      {-1, -1,  0,  0,  0,  0, -1, -1},
      {-1, -1, -1,  3, -1, -1, -1, -1},
      {-1, -1, -1,  4,  3, -1, -1, -1},
      {-1, -1, -1,  4,  3, -1, -1, -1},
      {-1, -1,  4, -1, -1,  3, -1, -1},
      {-1, -1,  4, -1, -1,  3, -1, -1},
      {-1, -1,  4,  4, -1,  3,  3, -1}
    }
};


// Colors del robot

int colorsRobot[5]   = {LIGHTGREEN, GREEN, GREEN, GREEN, GREEN};


// Variables d'estat del robot i paistage

int estat;        // estat
int x0, y0;       // referencia pel dibuix del robot
int xf, yf;       // referencia per a la posicio de les frases
int imatge;       // imatge a representar del robot


// Paisatge

int h[FINESTRA_W];  // horitzo. alsaries des de la part superior pantalla
int vorera;         // vorera
```

```
// Taula de frases

char *frases[] =
{
  "                          ",
  "CONNECTAT! ESPERO ORDRES",
  "CAMINAR, QUIN PLAER!    ",
  "CORRER, UF, UF, UF!!!   ",
  "ARA TOCA DESCANSAR      ",
};


// Llegenda

#define NCODIS_LL 8

char *llegenda[] =
{
  "E: ENGEGA ",
  "C: CAMINA ",
  "K: CORRE  ",
  "P: PARLA  ",
  "L: CALLA  ",
  "S: ATURAT ",
  "A: APAGA  ",
  "F: FI     "
};



// accions de l'automat

void estatInicial        (tEvent event);
void estatEngegat        (tEvent event);
void estatCaminant       (tEvent event);
void estatCorrent        (tEvent event);
void estatAturat         (tEvent event);
void estatEngegatParlant (tEvent event);
void estatCaminantParlant(tEvent event);
void estatCorrentParlant (tEvent event);
void estatAturatParlant  (tEvent event);
void estatFinal          ();
void nouEvent            (tEvent *event);

// accions auxiliars
```

```
void engega           ();
void camina           ();
void corre            ();
void parla            (int frase);
void calla            ();
void aturat           ();

void iniGeneral       ();
void escenaInicial    ();
void nouPaisatge      ();
void dibuixaEscena    ();
void dibuixaPaisatge  ();
void dibuixaVorera    ();
void dibuixaHoritzo   ();
void dibuixaRobot     ();
void esborraEscena    ();
void esborraHoritzo   ();
void esborraRobot     ();
void escriuLlegenda   ();
void escriuFrase      (int frase);
void sincronisme      (int velocitat);
int  aleatori         (int max);
int  aleatori101      ();


// Programa principal de l'automat

int main()
{

  iniGeneral();

  tEvent event = EVENT_APAGA;
  estat        = ESTAT_INICIAL;

  textbackground(BLACK);

  while( estat != ESTAT_FINAL )
  {
    switch(estat)
    {
      case ESTAT_INICIAL:
        estatInicial(event);
      break;

      case ESTAT_ENGEGAT:
```

```
        estatEngegat(event);
      break;

      case ESTAT_CAMINANT:
        estatCaminant(event);
      break;

      case ESTAT_CORRENT:
        estatCorrent(event);
      break;

      case ESTAT_ATURAT:
        estatAturat(event);
      break;

      case ESTAT_ENGEGAT_PARLANT:
        estatEngegatParlant(event);
      break;

      case ESTAT_CAMINANT_PARLANT:
        estatCaminantParlant(event);
      break;

      case ESTAT_CORRENT_PARLANT:
        estatCorrentParlant(event);
      break;

      case ESTAT_ATURAT_PARLANT:
        estatAturatParlant(event);
      break;
    }

    nouEvent(&event);
  }
  estatFinal();
  return 0;
}


void iniGeneral()
{

  int i;

  // Inicialitzem el generador de nombres aleatoris
  srand(GetTickCount() % 1000);
```

```
  // Pantalla en mode text 80x50
  textmode(PANTALLA_80x50);

  // Cursor invisible
  _setcursortype(_NOCURSOR);

  // Vorera
  vorera = FINESTRA_H - VORERA;

  // Referencies
  x0 = (FINESTRA_W - ROBOT_W)/2;
  y0 = (FINESTRA_H -(ROBOT_H + VORERA));
  xf = x0 + ROBOT_W;
  yf = y0 + 2;

  // Horitzo inicial
  h[0] = MINIM_H;
  for(i = 1; i <= FINESTRA_W - 1; i ++){
    h[i] = h[i - 1] + aleatori101() * CANVI_H;
    if(h[i] > MINIM_H) h[i] = MINIM_H;
    else if(h[i] < 1) h[i] = 1;
  }

  escenaInicial();
}



/*****************************************************************/
// Estat inicial desconnectat del robot

void estatInicial(tEvent event)
{

  switch (event)
  {

    case EVENT_ENGEGA:
      engega();
    break;
  }
  estat = nouEstat[estat][event];
}
```

```
/*********************************************************************/
// Tracta l'event en l'estat ENGEGAT i determina el nou estat

void estatEngegat(tEvent event)
{

  switch (event)
  {
    case EVENT_APAGA:
      escenaInicial();
    break;

    case EVENT_CAMINA:
      camina();
    break;

    case EVENT_CORRE:
      corre();
    break;

    case EVENT_PARLA:
      parla(FRASE_CONNEC);
    break;
  }
  estat = nouEstat[estat][event];
}



/*********************************************************************/
// Tracta l'event en l'estat CAMINA i determina el nou estat

void estatCaminant(tEvent event)
{

  switch (event)
  {
    case EVENT_APAGA:
      escenaInicial();
    break;

    case EVENT_ENGEGA:
      camina();
    break;
```

```
    case EVENT_CAMINA:
      camina();
    break;

    case EVENT_CORRE:
      corre();
    break;

    case EVENT_PARLA:
      parla(FRASE_CAMINA);
      camina();
    break;

    case EVENT_CALLA:
      camina();
    break;

    case EVENT_ATURAT:
      aturat();
    break;
  }
  estat = nouEstat[estat][event];
}

/*****************************************************************/
// Tracta l'event en l'estat CORRE i determina el nou estat

void estatCorrent(tEvent event)
{

  switch (event)
  {
    case EVENT_APAGA:
      escenaInicial();
    break;

    case EVENT_ENGEGA:
      corre();
    break;

    case EVENT_CAMINA:
      camina();
    break;

    case EVENT_CORRE:
      corre();
```

```
    break;

    case EVENT_PARLA:
      parla(FRASE_CORRER);
      corre();
    break;

    case EVENT_CALLA:
      corre();
    break;

    case EVENT_ATURAT:
      aturat();
    break;
  }
  estat = nouEstat[estat][event];
}

/*****************************************************************/
// Tracta l'event en l'estat ATURAT i determina el nou estat


void estatAturat(tEvent event)
{

  switch (event)
  {
    case EVENT_APAGA:
      escenaInicial();
    break;

    case EVENT_CAMINA:
      camina();
    break;

    case EVENT_CORRE:
      corre();
    break;

    case EVENT_PARLA:
      parla(FRASE_ATURAT);
    break;
  }
  estat = nouEstat[estat][event];
}
```

```
/*********************************************************************/
// Tracta l'event en l'estat ENGEGAT-I-PARLA i determina el nou estat

void estatEngegatParlant(tEvent event)
{

  switch (event)
  {
    case EVENT_APAGA:
      calla();
      escenaInicial();
    break;

    case EVENT_ENGEGA:
      parla(FRASE_CORRER);
      corre();
    break;

    case EVENT_CAMINA:
      parla(FRASE_CAMINA);
      camina();
    break;

    case EVENT_CORRE:
      parla(FRASE_CORRER);
      corre();
    break;

    case EVENT_PARLA:
      parla(FRASE_CONNEC);
    break;

    case EVENT_CALLA:
      calla();
    break;

    case EVENT_ATURAT:
      parla(FRASE_ATURAT);
      aturat();
    break;
  }
  estat = nouEstat[estat][event];
}


/*********************************************************************/
```

```
// Tracta l'event en l'estat CAMINA-I-PARLA i determina el nou estat

void estatCaminantParlant(tEvent event)
{

  switch (event)
  {
    case EVENT_APAGA:
      calla();
      escenaInicial();
    break;

    case EVENT_ENGEGA:
      parla(FRASE_CAMINA);
      camina();
    break;

    case EVENT_CAMINA:
      parla(FRASE_CAMINA);
      camina();
    break;

    case EVENT_CORRE:
      parla(FRASE_CORRER);
      corre();
    break;

    case EVENT_PARLA:
      parla(FRASE_CAMINA);
      camina();
    break;

    case EVENT_CALLA:
      calla();
      camina();
    break;

    case EVENT_ATURAT:
      parla(FRASE_ATURAT);
      aturat();
    break;
  }
  estat = nouEstat[estat][event];
}

/*****************************************************************/
```

```
// Tracta l'event en l'estat CORRE-I-PARLA i determina el nou estat

void estatCorrentParlant(tEvent event)
{

  switch (event)
  {
    case EVENT_APAGA:
      calla();
      escenaInicial();
    break;

    case EVENT_ENGEGA:
      parla(FRASE_CORRER);
      corre();
    break;

    case EVENT_CAMINA:
      parla(FRASE_CAMINA);
      camina();
    break;

    case EVENT_CORRE:
      parla(FRASE_CORRER);
      corre();
    break;

    case EVENT_PARLA:
      parla(FRASE_CORRER);
      corre();
    break;

    case EVENT_CALLA:
      calla();
      corre();
    break;

    case EVENT_ATURAT:
      parla(FRASE_ATURAT);
      aturat();
    break;
  }
  estat = nouEstat[estat][event];
}

/*****************************************************************/
```

```
// Tracta l'event en l'estat ATURAT-I-PARLA i determina el nou estat

void estatAturatParlant(tEvent event)
{

  switch (event)
  {
    case EVENT_APAGA:
      calla();
      escenaInicial();
    break;

    case EVENT_ENGEGA:
      parla(FRASE_ATURAT);
    break;

    case EVENT_CAMINA:
      parla(FRASE_CAMINA);
      camina();
    break;

    case EVENT_CORRE:
      parla(FRASE_CORRER);
      corre();
    break;

    case EVENT_PARLA:
      parla(FRASE_ATURAT);
    break;

    case EVENT_CALLA:
      calla();
    break;

    case EVENT_ATURAT:
      parla(FRASE_ATURAT);
    break;
  }
  estat = nouEstat[estat][event];
}



/*****************************************************************/
void estatFinal()
{
```

```
  // Tractament final


  //Pantalla en mode text inicial
  textmode(LASTMODE);
}


/*****************************************************************/
// Captura de les ordres que l'usuari entra pel teclat

void nouEvent (tEvent *event)
{
  char c;

  if(kbhit())
  {
     c = getch();
     if ('a' <= c && c <= 'z') c = (char) ((int)c - 32);

     switch(c)
     {
        case TECLA_APAGA:
          *event = EVENT_APAGA;
        break;

        case TECLA_ENGEGA:
          *event = EVENT_ENGEGA;
        break;

        case TECLA_CAMINA:
          *event = EVENT_CAMINA;
        break;

        case TECLA_CORRE:
          *event = EVENT_CORRE;
        break;

        case TECLA_PARLA:
          *event = EVENT_PARLA;
        break;

        case TECLA_CALLA:
          *event = EVENT_CALLA;
        break;
```

```
        case TECLA_ATURAT:
          *event = EVENT_ATURAT;
        break;

        case TECLA_ACABA:
          *event = EVENT_ACABA;
        break;
     }
  }
}

void engega(){

  esborraEscena();
  imatge = 1;
  dibuixaEscena();
}


void camina()
{

  esborraEscena();
  nouPaisatge();
  dibuixaEscena();
  sincronisme(VELOCITAT_CAMINA);
  imatge = imatge + 1;
  if(imatge > NIMATGES) imatge = 1;
}


void corre()
{

  esborraEscena();
  nouPaisatge();
  dibuixaEscena();
  sincronisme(VELOCITAT_CORRE);
  imatge = imatge + 1;
  if(imatge > NIMATGES) imatge = 1;
}


void aturat()
{
```

```
  esborraEscena();
  imatge = 1;
  dibuixaEscena();
}


void escenaInicial()
{

  clrscr();
  escriuLlegenda();
  imatge = 0;
  dibuixaEscena();
}


void escriuLlegenda()
{

  int i;

  textcolor(COLOR_LLEGENDA);
  gotoxy(LLEGENDA_X, LLEGENDA_Y);
  for (i = 0; i < NCODIS_LL; i++){
    cprintf("%s", llegenda[i]);
  }
}


void nouPaisatge()
{
  int i;

  // La vorera no canvia

  // nou horitzo - scroll una unitat cap a la'esquerra
  for(i = 1; i <= FINESTRA_W - 1; i++) h[i-1] = h[i];

  h[FINESTRA_W - 1] = h[FINESTRA_W - 1] + aleatori101()*CANVI_H;
  if(h[FINESTRA_W - 1] > MINIM_H) h[FINESTRA_W - 1] = MINIM_H;
  else if(h[FINESTRA_W - 1] <= 1)  h[FINESTRA_W - 1] = 2;
}


void dibuixaEscena()
{
```

```
  dibuixaPaisatge();
  dibuixaRobot();
}


void dibuixaPaisatge()
{

  dibuixaVorera();
  dibuixaHoritzo();
}


void dibuixaVorera()
{

  int i;

  textcolor(COLOR_VORERA);
  gotoxy(1, vorera);
  for(i = 0; i < FINESTRA_W - 1; i ++){
    cprintf("%c", DIAMANT);
  }
}


void dibuixaHoritzo()
{

  int i;

  textcolor(COLOR_HORITZO);
  for(i = 0; i < FINESTRA_W - 1; i ++){
    gotoxy(i, h[i]);
    cprintf("%c", DIAMANT);
  }
}


void dibuixaRobot()
{
  int x, y;
  int color;

  for (y = 0; y < ROBOT_H; y++){
    for (x = 0; x < ROBOT_W; x++){
```

```
      color = imatgesRobot[imatge][y][x];
      if (color != COLOR_PADDING){
         gotoxy(x0+x, y0+y);
         textcolor(colorsRobot[color]);
         cprintf("%c", QSOLID);
      }
    }
  }
}


void esborraEscena()
{

  esborraHoritzo();
  esborraRobot();
}


void esborraHoritzo()
{

  int i;

  textcolor(BLACK);
  for(i = 0; i < FINESTRA_W - 1; i++){
    gotoxy(i, h[i]);
    cprintf("%c", DIAMANT);
  }
}


void esborraRobot()
{
  int x, y;

  textcolor(BLACK);
  for (y = 0; y < ROBOT_H; y++){
    for (x = 0; x < ROBOT_W; x++){
      gotoxy(x0+x, y0+y);
      cprintf("%c", QSOLID);
    }
  }
}
```

```
void parla(int frase)
{

  long t1, t2;

  escriuFrase(frase);

  /*
   t1 = GetTickCount();
  do {
      t2 = GetTickCount();
  } while((t2-t1) < (DELAY_SINC/VELOCITAT_FRASE));
  */
}


void calla()
{

  escriuFrase(FRASE_BLANC);
}


void escriuFrase(int frase)
{

  gotoxy(xf, yf);
  cprintf("%s", frases[frase]);
}


void sincronisme (int velocitat)
{

 long t1, t2;

 t1 = GetTickCount();
 do {
      t2 = GetTickCount();
 } while((t2-t1) < (DELAY_SINC/velocitat));
}


// Retorna un valor aleatori en {-1, 0, 1}

int aleatori101()
```

```
{

  return aleatori(3) - 1;
}

// Retorna un valor aleatori en [0, max-1]

int aleatori(int max)
{

  return (rand() % max);
}
```

7 Ampliació

Has codificat, muntat i executat l'exemple del robot en el teu computador amb èxit? Enhorabona! Ara et proposem uns petits reptes.

1. Si t'agraden les velocitats d'execució que has trobat, anota en un paper els valors de les constants `VELOCITAT_CAMINA`, `VELOCITAT_CORRE`, `VELOCITAT_FRASE` i `RETARD_SINC`. Ara canvia'ls de manera arbitrària i observa com es comporta el programa. Sabries explicar els canvis observats? Si no n'has trobat de millors, restableix els valors inicials.

2. Com que el joc s'executa amb la pantalla en mode `text`, les parts que canvien en el temps, abans d'actualitzar-les, s'esborren selectivament amb el procediment `esborraEscena()`. La rutina de sistema `clrscr()`, que esborra tota la pantalla, és més senzilla i eficient. Susbtitueix cada ocurrència de `esborraEscena()` per `clrscr()` i estudia els efectes sobre l'execució.

3. Com pots imaginar, el món dels autòmats és molt més extens i complex del que en una petita introducció com la que hem fet ací es pot mostrar. Per exemple, un problema important és el de la minimització on l'objectiu és obtenir l'autòmat que representi el comportament desitjat però usant el mínim nombre possible d'estats i accions. Estudia les similituds i diferències entre els etats `Engegat` i `Aturat` i entre `Engegat-i-parlant` i `Aturat-i-parlant`. Es podrien suprimir els estat `Aturat` i `Aturat-i-parlant` tot substituint-los per `Engegat` i `Engegat-i-parlant` respectivament? Prova-ho.

4. Suposa que vols que el robot pugui efectuar salts longitudinals en el sentit de la marxa tot describint una paràbola. Amplia amb aquesta possibilitat l'autòmat donat.

5. Ara suposa que vols afegir una ordre un xic més complicada de resoldre. Vols tenir la possibilitat que el robot es giri i es mogui en sentit oposat al que estigui movent-se en un instant determinat. Amplia amb aquesta possibilitat l'autòmat donat.

6. Afegeix qualsevol altra funcionalitat que voldries que tinguès el robot.

8 Bibliografia

Entre la nombrosa bibliografia publicada que tracta d'autòmats finits, podem destacar la següent.

1. R. Cases, L. Màrquez, *Llenguatges, gramàtiques i autòmats : curs bàsic*, Edicions UPC, 2003.
2. A. J. Champandard, *10 Reasons the Age of Finite State Machines is Over*, URL `http://aigamedev.com/questions/fsm-age-is-over/`, Juliol 2009.
3. T. Faison, *Event-Based Programming: Taking Events to the Limit*, Apress, 2006.
4. J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to automata theory, languages, and computation*, Pearson Education International, 2003.
5. D. Kelley, *Automata and formal languages : an introduction*, Prentice Hall, 1995.
6. H. R. Lewis, C. H. Papadimitriu. *Elements of the Theory of Computation*, Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1981.
7. F. Wagner, R. Schmuki, T. Wagne, P. Wolstenholme, *Modeling Software with Finite State Machines: A Practical Approach*, Auerbach Publications, CRC Press 2006, Boca Raton, Florida.