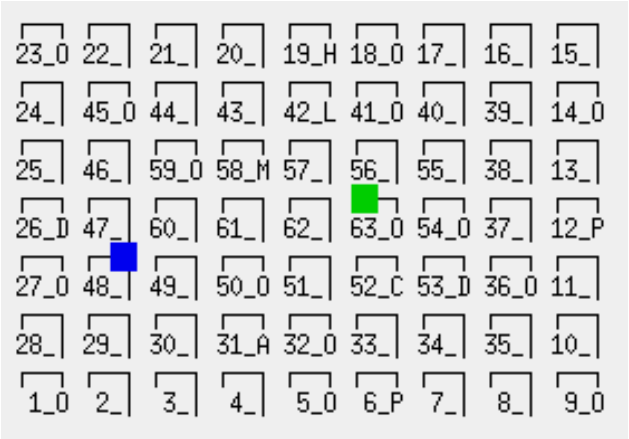
toca.c

como la Oca pero en modo texto



Javier Quintano Merino

ETSI Bilbao

Diciembre 9, 2013

Index

- Intro
  - Motivación
  - Objetivos
- Estructura
  - Modos de ejecución
  - Esqueleto
  - Estructura de datos
  - Funciones
- Extra
  - Desarrollo
  - TO-DO

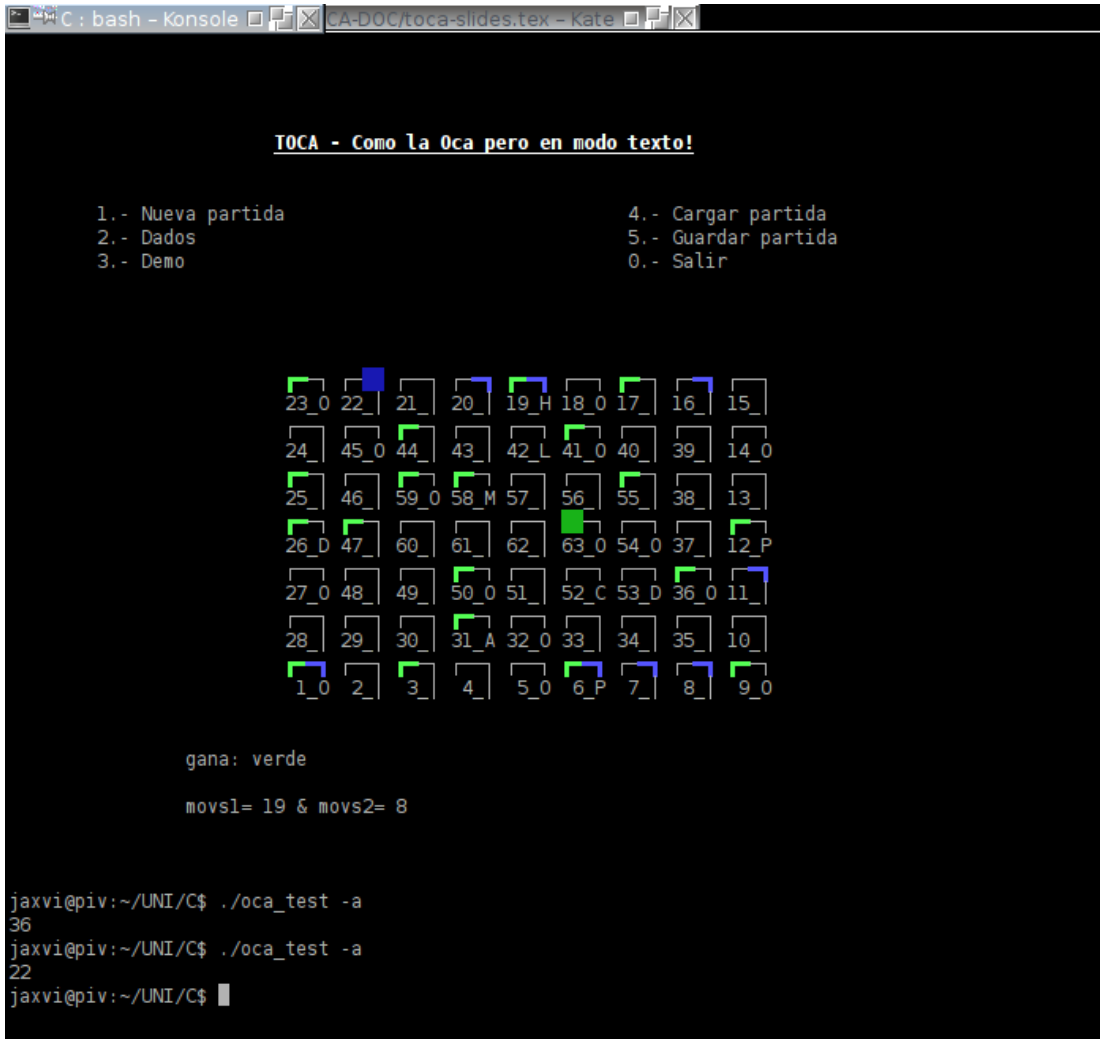
Motivación

- aprobar la asignatura
- aclarar algunas dudas existenciales sobre el juego de la oca
  - ganar/perder a la oca, azar
  - tirada perfecta
  - estadísticas del número de tiradas
  - ...

Objetivos

- programar un juego de la oca en modo texto puro
- hacer uso de los conocimientos adquiridos
  - sintaxis de C
  - estructuras de datos
  - acceso a ficheros
  - ...
- generar datos para estudiar estadísticamente las partidas
- desarrollar un programa sencillo pero completo
- que sea divertido/motivador

Ojetivos


C : bash – Konsole
CA-DOC/toca-slides.tex – Kate
TOCA - Como la Oca pero en modo texto!
1.- Nueva partida
2.- Dados
3.- Demo
4.- Cargar partida
5.- Guardar partida
0.- Salir
23_0 22_ 21_ 20_ 19_H 18_0 17_ 16_ 15_
24_ 45_0 44_ 43_ 42_L 41_0 40_ 39_ 14_0
25_ 46_ 59_0 58_M 57_ 56_ 55_ 38_ 13_
26_D 47_ 60_ 61_ 62_ 63_0 54_0 37_ 12_P
27_0 48_ 49_ 50_0 51_ 52_C 53_D 36_0 11_
28_ 29_ 30_ 31_A 32_0 33_ 34_ 35_ 10_
1_0 2_ 3_ 4_ 5_0 6_P 7_ 8_ 9_0
gana: verde
movs1= 19 & movs2= 8
jaxvi@piv:~/UNI/C$ ./oca_test -a
36
jaxvi@piv:~/UNI/C$ ./oca_test -a
22
jaxvi@piv:~/UNI/C$

Modos de ejecución


```
int main(int argc, const char *argv[]){

    if((argc > 1) && (argv[1][0] == '-') && (argv[1][1] == 'a')){
        runAuto(); //for num in {1..10}; do sleep 1; ./oca_test -a >>/tmp/stats.
        return 0;
    }

    runInteractive();

    printf("\n\n");
    return 0;

}
```

Esqueleto

```
1   /*
2   GPL - jqm - ETSI 2013
3   */
4
5
6   #include <stdio.h>
7   #include <time.h>
8   #include <stdlib.h>
9
10  void showMenu(int intro);
11  void initGame();
12  void printGame();
13  void runAuto();
14  int runInteractive();
15
16
17  struct square{
39
40  struct square game[64];
41
42  int main(int argc, const char *argv[]){
43  ····
44      if((argc > 1) && (argv[1][0] == '-') && (argv[1][1] == 'a')){
45          runAuto(); //for num in {1..10}; do ./toca -a >>/tmp/stats.txt; done
46          return 0;
47      }
48  ····
49      runInteractive();
50  ····
51      printf("\n\n");
52      return 0;
53  ····
54  }
55
56  void initGame(){
90
91  void printGame(){
114
115 void showMenu(int intro){
153
154 void runAuto(){
157
158 int runInteractive(){
```

Estructura de datos

```c
struct square{
    /*casilla base 4x2 chars
    |p1p2
    |36 0
    |casilla Nº 36 tipo Oca
    */
....
    char *player1; //posición coloreada para el jugador 1
    char *player2; //posición coloreada para el jugador 2
    int index;  //numero de casilla
    char *action; /* acción de casilla especial, hardcodeadas según index
                     S = start
                     O = oca {1, 5, 9, 14, 18, 23, 27, 32, 36, 41, 45, 50, 54, 59, 63}
                     P = puente {6, 12}
                     H = hostal/posada {19}
                     D = dados {26, 53}
                     A = agua/pozo {31}
                     L = laberinto {42}
                     C = carcel {52}
                     M = muerte {58}
                     */
};

struct square game[64];
```

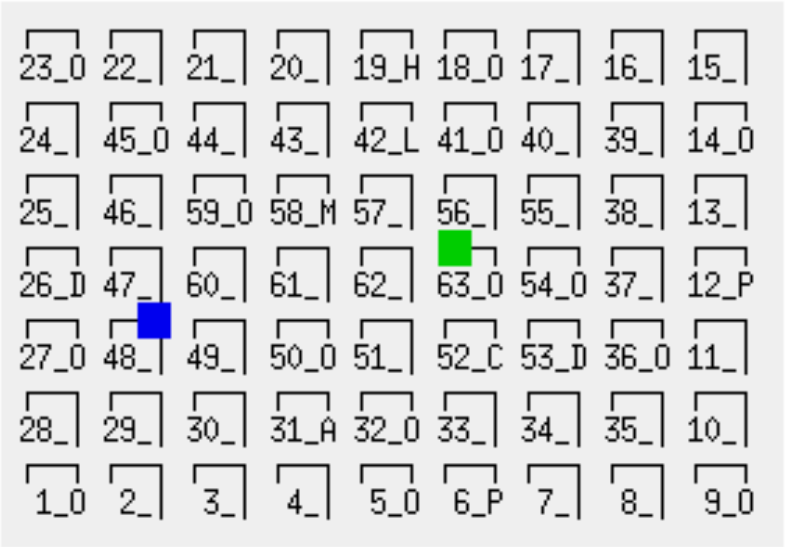
Funciones

```c
#include <stdio.h>
#include <time.h> //para nanosleep / man nanosleep
#include <stdlib.h> //para random / man random

void showMenu(int intro); //muestra el menu de opciones, intro =1 es que escribe
void initGame(); //inicializa el tablero con sus index y actions y decoraciones
void printGame(); //dibuja el tablero
void runAuto(); //juega en modo auto y devuelve numero de movimientos
int runInteractive(); //juega partida interactiva para 2 players

void saveGame(); //guarda partida a fichero en disco
void loadGame(); //carga partida guardada
```

Funciones



P: y el tablero? y los colores? y el refresco? y las animaciones?
R: printf + secuencias de escape + unicode (directamente en el source UTF8...) + random() ...

Funciones

```
void printSquare(int index){
    printf(" %s%s\e[1B\e[4D%2d%s\e[1A",game[index].player1,game[index].player2,game[index].index,game[index].action);
    //1B = 1 linea abajo, 4D = 4 backward, 1A = 1 arriba
}
void typewrite(char *string){
    int i=0;
    srandom (time (0)); //inicializar la seed para random() con el time actual
    struct timespec tim; //tim es una struct del tipo timespec (en time.h) que nanosleep espera como parámetros
    tim.tv_sec = 0;  //0 segundos, se usará tim.tv_nsec + una parte de random

    while(string[i]){

        tim.tv_nsec = 25000000 + (random()% 100000000); //1/4 segundo serían 250 ms 250000 us 250000000 ns

        putchar(string[i]);
        fflush(stdout);
        nanosleep(&tim, NULL);
        i++;
    }
}
```

```
for(i=1;i<64;i++){
    game[i].player1= "┌─";
    game[i].player2= "─┐";
    game[i].index= i;

    if(i==1 || i==5 || i==9 || i==14 || i==18 || i==23 || i==27 || i ==32 || i ==36 || i ==41 || i ==45 || i ==50 || i ==54 || i ==59 || i ==63)
        game[i].action = "_O";
    else if (i==6 || i==12)
        game[i].action = "_P";
    else if (i==19)
        game[i].action = "_H";
```

Desarrollo

- sólo librería standard stdio.h, time.h, stdlib.h
- puro texto (UNICODE) sin librerías gráficas (ncurses(linux) o conio.h(win))
- funciones dentro de POSIX (https://en.wikipedia.org/wiki/POSIX) e.g. nanosleep frente a usleep
- uso de GIT (https://en.wikipedia.org/wiki/Git_(software)) para control de código
- básicamente lo estudiado en clase + alguna treta :P

TO-DO

- real time scores
- $>$ de 2 players
- menu settings (colores, penalizaciones, casillas especiales...)
- correcciones, sugerencias, bugs...

TO-DO

...