Programación de Sistemas
Para
El Control de Procesos

PROGRAMACION EN C DEL JUEGO

SNAKEBOOM

Francisco Javier Jorge Trujillo
Electrónica
74918076-S
Tlfno. 626158102

Finalidad del trabajo

Se pretende crear un programa en lenguaje C que simule al famoso juego de la serpiente, este programa incluye una serie de variaciones respecto al juego original:

- Se incluye un modo de 2 jugadores simultáneos, es decir, cada jugador maneja una serpiente dentro del tablero, el juego consiste en intentar no chocar con la otra serpiente, con uno mismo o con los límites del tablero, de este modo se finaliza el juego.

  También va apareciendo de forma aleatoria un ITEM que permite hacer crecer la serpiente. La velocidad del juego aumentará a medida que pase el tiempo

- En el juego de un jugador, irán apareciendo bombas por el tablero, de forma que si la serpiente pasa por encima de alguna de ellas, la bomba se activa.

  Una vez activada la bomba, comenzará la cuenta atrás para su explosión, una vez que explote, se crean 8 trozos que se desplazarán por el tablero.

  Si la serpiente choca con alguno de estos trozos, con los límites del tablero o consigo mismo el juego termina.

- Existe tres niveles de juego cada uno con una determinada velocidad de la serpiente y con un numero de bombas.

- El programa permite también la posibilidad de guardar y visualizar las máximas puntuaciones obtenidas en cada nivel.

- El programa incluye una ayuda indicando los controles de la serpiente y las características del juego (pulsando F1).

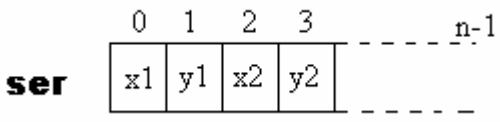
Resolución de problemas

A continuación se muestran las distintas soluciones escogidas para programar las distintas características de la serpiente (manejo, movimiento, choque,…..), posteriormente se muestra el método para generar y guardar el estado de las bombas, así como de los pedazos producidos por ellas.

Serpiente

Declaración puntos de la serpiente

Para guardar todas las posiciones (coordenadas x e y) de la serpiente se ha utilizado un array de enteros que contendrá todos los puntos de la serpiente:



(x1,y1) = posición del punto 1
(x2,y2) = posición del punto 2

De este modo si queremos un serpiente con un tamaño de 4 puntos, se crea un array de 8 elementos (2 posiciones por punto).

La variable **N** indica el tamaño de la serpiente, para declarar el tamaño del array se utiliza la función malloc.

```
ser =(int *)malloc(N*sizeof(int));
```

Si se quiere acceder a las coordenadas **x** e **y** de cada punto se utiliza el siguiente bucle que ira recorriendo el array de 2 en 2:

```
for(i=0;i<N;i=i+2)
  {
  ser[i]        /*Coordenada x*/
  ser[i+1]      /*Coordenada y*/
  }
```

Mediante la función **iniciar_arrays( )** se inicializa los puntos iniciales de la serpiente

```
i=0;
do
  {
  ser[i]=200-(10*(i/2));
  ser[i+1]=150;
   i=i+2;
  }while (i != N);
.....................................
```

ser (por ejemplo para N=8):

200	150	190	150	180	150	170	150

Movimiento de la serpiente

Para mover la serpiente se observa el algoritmo de movimiento:

Supongamos que tenemos la siguiente serpiente:

ser

200	150	190	150	180	150	170	150

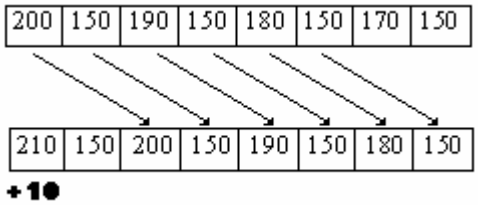
* * * *

ser (tras moverse a la derecha)

210	150	200	150	190	150	180	150

* * * *

Se puede observar que el primer punto es el que sufre una modificación (según la dirección) y el resto de puntos sufren un desplazamiento a la izquierda.



Por tanto se crea un array de enteros que contendrá la posición futura de todos los puntos de la serpiente (**ser2**), tendrá el mismo tamaño que el array **ser**.

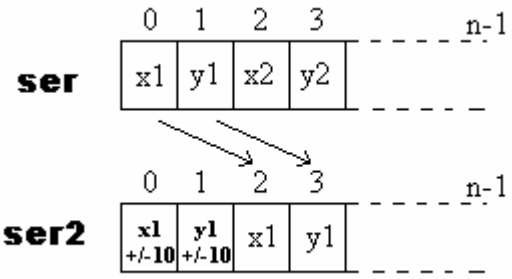
```
ser2=(int *)malloc(N*sizeof(int));
```

El primer punto del array **ser2** será el primer punto del array **ser** modificado según la dirección de la serpiente.

```
ser2[0] = ser[0] (+10 derecha -10 izquierda);
ser2[1] = ser[1] (+10 abajo     -10 arriba);
```

El segundo punto de **ser2** será el primero de **ser**, el tercer punto de **ser2** será el segundo de **ser**,…….así hasta acabar con todos los puntos.

```
ser2[2] = ser [0]
ser2[3] = ser [1]
ser2[4] = ser [2]
ser2[5] = ser [3]…….
```



La siguiente función se calcula el estado próximo de los puntos y los guarda en el array **ser2**:

```
int mueve_serpiente(int *ser,int *ser2,int N,int dir,int i)
 {
  for (i=2;i<N;i=i+2)       /*El n_punto del array ser2 sera (n-1)_punto del
                              array ser*/
   {
   ser2[i]=ser[i-2];
   ser2[i+1]=ser[i-1];
   }

  switch(dir)               /*Se cambiara el primer punto (cabeza) segun
                              la direccion*/
    {
    case 0:ser2[0]=ser[0]+10;ser2[1]=ser[1];i=0;break;
    case 1:ser2[0]=ser[0]-10;ser2[1]=ser[1];i=1;break;
    case 2:ser2[0]=ser[0];ser2[1]=ser[1]+10;i=2;break;
    case 3:ser2[0]=ser[0];ser2[1]=ser[1]-10;i=3;break;
    }
  return(i);                /*Se devuelve el valor de la direccion actual*/
 }
```

Tras calcular el estado próximo éste se copia al estado actual (**ser2 → ser**)

```
void actualizo_ser(int *ser,int *ser2, int i, int N)
  {
  for (i=0;i<N;i++)
    {
    ser[i]=ser2[i];
    }
  }
```

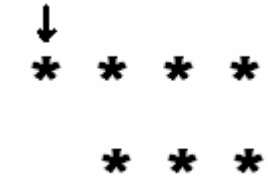
Pintar serpiente

Para evitar tener que borrar la serpiente entera y volver a dibujar todos sus puntos de nuevo, se realiza lo siguiente:

- Antes de calcular el estado próximo, se borra el último punto del estado actual.

ser

200	150	190	150	180	150	***170***	***150***

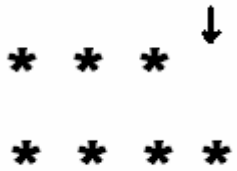


- Se calcula el estado próximo (**ser2**) y se actualiza **ser (**ser = ser2**)**.

- Se pinta el primer punto.

ser

210	***150***	200	150	190	150	180	150



Este es el orden en que se utilizarían las funciones:

```
pinta_ser(ser,N,0);          /*Se borra el ultimo punto de la serpiente*/

actualizo_ser(ser,ser2,i,N); /*Se actualiza el estado proximo (ser2)
                               al actual (ser)*/
pinta_ser(ser,N,1)           /*Pinta el primer punto de la serpiente*/
```

Mover serpiente

La dirección actual de la serpiente se guarda en la variable entera **dir**, según su valor se tiene la dirección : 0 derecha,1 izquierda, 2 abajo y 3 arriba.

Al pulsar una tecla, se comprueba si esa tecla pulsada corresponde con los controles de la serpiente, si esto ocurre y la dirección anterior (guardada en la variable **dir2**) no es contraria a la dirección que se quiere ir (es decir, si se quiere ir a la derecha la serpiente no puede estar moviéndose a la izquierda, si se quiere ir hacia arriba, la serpiente no puede estar moviéndose hacia abajo,….., de este modo se evita que la serpiente cambie de sentido), posteriormente se actualiza el valor de **dir** a la dirección deseada.

```
if (c=='\x48'&dir2!=2)        /*Se pulsa tecla cursor arriba*/
          dir=3;               /*Direccion serpiente1 = arriba*/

 if (c=='\x50'&dir2!=3)       /*Se pulsa tecla cursor abajo*/
          dir=2;               /*Direccion serpiente1 = abajo*/
……
……
```

Después se mueve la serpiente según la variable **dir** mediante la función **mueve_serpiente()**, y el valor que devuelve (que es la dirección actual) se guarda en dir2 **(dir2=mueve_serpiente())**.

Choque serpiente

Para comprobar si la serpiente ha chocado consigo mismo o con el tablero, se comprueba si el primer punto (**ser[0], ser[1]**) coincide con la posición de algún punto del resto del cuerpo (mediante un barrido con un bucle for de todos los puntos siguientes) o con el límite del tablero.

Mediante esta función se comprueba si existe choque:

```
int choque_ser(int i,int N,int *ser,int k)
 {
 k=0;
 for (i=2;i<N;i=i+2)
  {
  if((ser[0] = = ser[i]) & (ser[1] = =ser[i+1]))  /*Choca con su cuerpo*/
    k=1;
  }

  if (ser[0] <= 105 || ser[0]>520 || ser[1]<90 || ser[1] >390)
    k=1;                                          /*Se sale del tablero*/

 return(k);
```

Para comprobar si la serpiente ha chocado con algunos de los trozos cuando una bomba explota, se comprueba si algún punto de la serpiente coincide con la posición de algún trozo(se realiza un barrido mediante un bucle for de todos los puntos del array ser), los trozos se van generando a partir del array pedazos (que guarda la posición inicial y el incremento).

Función que comprueba si ha chocado con algún trozo

```
int choque_trozo(int *pedazo,int *ser,int N,int choquer,int i,int k)
{
 choquer=0;

 for(i=0;i<40;i=i+4)
  {
  for (k=0;k<N;k=k+2)
   {
   if(pedazo[i]==ser[k] & pedazo[i+1]+pedazo[i+2]= =ser[k+1])
   {
   choquer=1;
   }
   .....Se  comprueba los 6 trozos
```

Mediante la variable **choque** se determina si ha chocado la serpiente (1 hay choque/0 no hay choque), el valor que devuelven las funciones anteriores se guarda en esta variable.

Crecimiento de la serpiente

Cuando la variable **cuenta** se hace cero, la serpiente aumenta su tamaño en un punto (se le añade 2 elementos más al array con la función **realloc**).

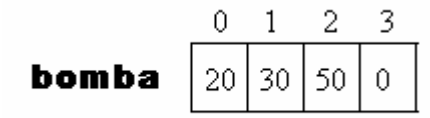
```
if(cuenta==0)                /*La serpiente 1 crece*/
        {
        N=N+2;
        ser = (int *) realloc ((int *)ser,N*sizeof(int));
        ser2 = (int *) realloc ((int *)ser2,N*sizeof(int));
        agrandar_serpiente(ser,N,dir);
        cuenta=75;
        }
```

La **funcion agrandar_serpiente ()**, coloca un punto al final de la serpiente dependiendo de la dirección en que se este moviendo.

Bombas

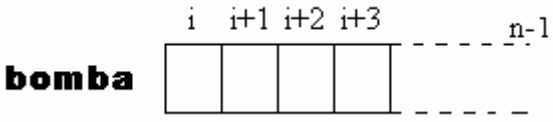
Declaración

Cada bomba cuenta con cuatro elementos: coordenada x, coordenada y, cuenta explosión, y estado (activada/desactivada).



bomba[0] = coordenada x
bomba[1] = coordenada y
bomba[2] = cuenta para explosión
bomba[3] = estado bomba (1 activada/ 0 desactivada)

Todos estos elementos se almacenan en un array de enteros denominado **bomba[ ],** cuyo tamaño viene determinado por **nb** (número de bombas).



Si se necesitan **n** bombas y cada bomba ocupa 4 posiciones, entonces se creará un array bombas[ ] con un tamaño de **4*nb**.

```
bomba=(int *)malloc((4*nb)*sizeof(int));
```

Para recorrer las distintas características de cada bomba se utiliza el siguiente bucle:

```
for(i=0;i<(4*nb);i=i+4)
        {
          bomba[i] = coordenada x
          bomba[i+1] = coordenada y
          bomba[i+2] = cuenta para explosión
          bomba[i+3] = estado bomba (1 activada/ 0 desactivada)
          }
```

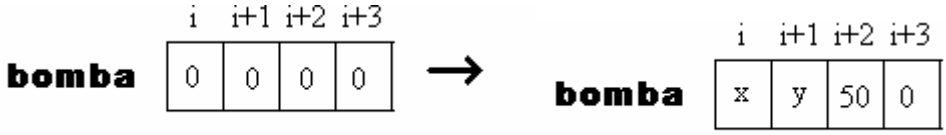
Comportamiento de las bombas

Activacion: Si en algún momento la cabeza de la serpiente coincide con la posición de alguna bomba, se activa (**bomba[i+3]=1**) y la puntuación se incrementa:

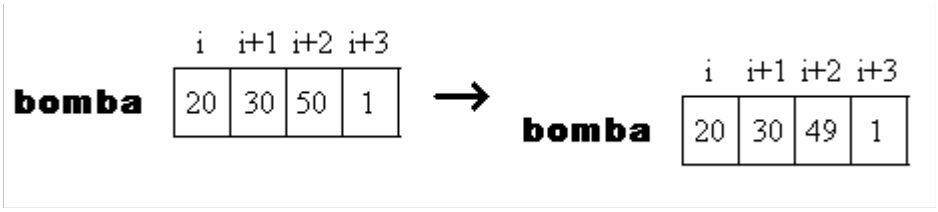
```
for(i=0;i<(4*nb);i=i+4)
        {
        if(bomba[i]==ser[0] & bomba[i+1]==ser[1])  /*Se activa la bomba*/
          {                                         /*si la serpiente pasa*/
          bomba[i+3]=1;                            /*por encima de ella*/
          puntos=puntos+50;                        /*puntos+50*/
          }
        }
```

La función **refresca_cuenta_bom( )**, realiza las siguientes tareas:

- Si la **posición** de alguna bomba es **(0,0)**, es decir no existe o ya ha acabado de explotar, se crea una nueva bomba (se le asigna un punto válido, se carga la cuenta y el estado se establece como desactivado).



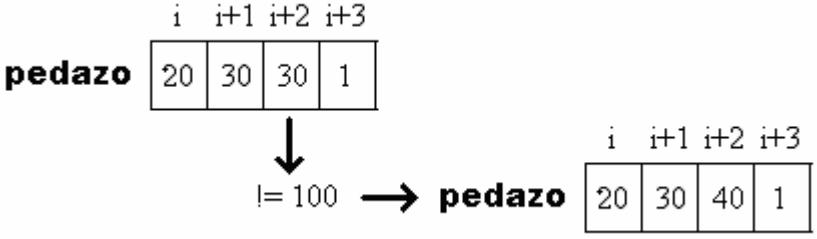
- Si alguna bomba está **activada** y la cuenta no es cero, se decrementa la cuenta -1



- Si **la cuenta** de alguna bomba **ha terminado (bomba[i+2] = 0)** se guarda la posición de la bomba en pedazo[i] y en pedazo [i+1], que será el punto de inicio de la explosión.

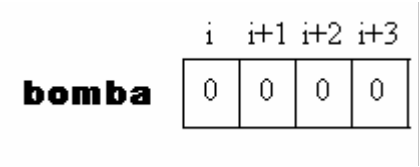
```
bomba[i]=pedazo[i];
bomba[i+1]=pedazo[i+1]
```

- Hasta que el **incremento de pedazo** (**pedazo[i+2]**)no alcance un valor de 100 se incrementa esa variable.



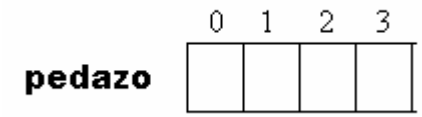
- Posteriormente se pinta los pedazos llamando a la **función pinta_pedazos** (se explicará su funcionamiento más adelante).

- Una vez que algun incremento de pedazo hayan alcanzado cierta distancia (**pedazo[i+2] = 100**) esa bomba ha acabado de explotar, este hecho se indica poniendo todo el array a 0.



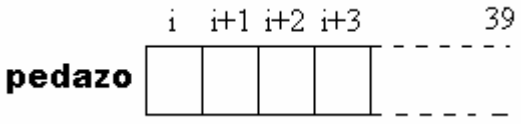
Comportamiento de los pedazos

La idea es generar 8 pedazos a partir del punto donde explotó la bomba, para ello se utiliza un array de enteros donde cada pedazo tiene cuatro elementos:

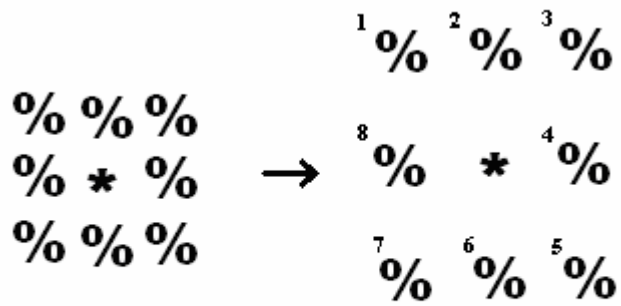


pedazo[0] = coordenada x de inicio de explosión
pedazo[1] = coordenada y de inicio de explosión
pedazo[2] = distancia al centro de explosión
pedazo[3] = 1

Todos estos elementos se almacenan en un array de enteros denominado **pedazos[ ],** cuyo tamaño será fijo (pedazos[40]) ya que como máximo el programa utilizará 10 bombas, por tanto solo necesita como máximo 10 explosiones.



Para pintar los 6 trozos, se toma como referencia el punto de inicio de explosión, luego se le suma o se le resta la distancia que contiene el elemento **pedazo[i+3]** a cada uno de los 6 trozos (generados mediante un patrón). La **funcion pinta_pedazos ( )** realiza dicha acción.



* Punto de inicio explosión

Patrón de generación de trozos

Trozo 1	x-incr, y-incr.	peazo[i]-peazo[i+2], peazo[i+1]-peazo[i+2].
Trozo 2	x, y-incr.	peazo[i], peazo[i+1]-peazo[i+2].
Trozo 3	x+incr, y-incr.	peazo[i]+peazo[i+2], peazo[i+1]-peazo[i+2].
Trozo 4	x+incr, y.	peazo[i]+peazo[i+2], peazo[i+1].
Trozo 5:	x+incr, y+incr.	peazo[i]+peazo[i+2], peazo[i+1]+peazo[i+2].
Trozo 6	x, y+incr.	peazo[i], peazo[i+1]+peazo[i+2].
Trozo 7	x-incr, y+incr.	peazo[i]-peazo[i+2], peazo[i+1]+peazo[i+2].
Trozo 8	x-incr, y.	peazo[i]-peazo[i+2], peazo[i+1].

Archivo de máxima puntuación

Las máximas puntuaciones estarán recogidas en el archivo de texto "max_score.txt".

Mediante tres lecturas consecutivas se obtendrá la puntuación y el nombre de cada nivel

(primera lectura→nivel fácil, segunda lectura→nivel medio, tercera lectura (nivel difícil))

Para evitar modificaciones en el fichero de máxima puntuación por parte del usuario, se crea un numero secreto fruto de la combinación de todas las puntuaciones y de operaciones matemáticas, dicho número será incluido al final del fichero.

Antes de cargar el menú, el programa mediante la función **comprobar_fich_max( ),** verifica que el archivo "max_score.txt" no ha sido modificado, es decir, comprueba si el numero que hay al final del fichero coincide con el numero secreto generado, si este no coincide o el fichero no existe, crea un nuevo fichero inicial (puntuaciones = 0 nombre = jugador).

Modo 2 jugadores

Se crea una nueva serpiente, cuyo comportamiento es similar al de la serpiente anteriormente explicada, por tanto empleará las mismas funciones para moverse, comprobar si existe choque, pintar en pantalla,…..

Se crean nuevos arrays y variables para la serpiente 2, de forma análoga se tiene:

-2 arrays de enteros: **sera** (guarda el estado actual), **sera2** (guarda el estado futuro)

-Variables: **N2** (tamaño de la serpiente2), **choque2** (determina si ha chocado la serpiente2), **cuenta2** (cuenta para hacer crecer la serpiente).

Para comprobar si una serpiente ha chocado con la otra se comprueba si la cabeza de una de las serpientes (primer punto) coincide con alguna posición del cuerpo de la otra serpiente.

```
/*Se comprueba si la serpiente 2 choca con el cuerpo de
   la serpiente 1*/

        for(i=0;i<N;i=i+2)
         {
         if(sera[0]==ser[i] && sera[1]==ser[i+1])
           choque2=1;
         }

 /*Se comprueba si la serpiente 1 choca con el cuerpo de
   la serpiente 2*/

        for(i=0;i<N2;i=i+2)
         {
         if(ser[0]==sera[i] && ser[1]==sera[i+1])
           choque=1;
         }
```

Ítems

Durante el juego para 2 jugadores aparecerá un ítem, si alguna de las serpientes pasa por encima del ítem crecerá su tamaño un punto.

La posición del ítem se guarda en un array de enteros de 2 elementos que contendrán la posición x e y generada de forma aleatoria (dicho punto no puede coincidir con la posición de ninguna de las serpientes y estar dentro de los límites del tablero)

item [2]

Con la función **crea_item ()** se comprueba si el array contiene solo ceros, en ese caso se rellena el array con punto válido.

Posteriormente mediante con la función **comprueba_item()**, se comprueba si alguna serpiente coincide con el ítem, si es así la función devuelve un uno en caso contrario un cero.

Cuando la función comprueba_item devuelve un 1 para una serpiente,pone su cuenta (cuenta para serpiente1, cuenta 2 para serpiente2) a 0, de este modo crecerá la serpiente.

```
if(comprueba_item(item,ser,N,sigue,k)==1) /*Si ser1 coje el item*/
        {
        cuenta=0;                   /*Cuenta para crecer*/
        setcolor(0);
        outtextxy(item[0],item[1],"o");     /*Borra el item*/
        setcolor(1);
        outtextxy(item[0],item[1],"*");
        item[0]=0;                  /*Inicializa la posicion*/
        item[1]=0;                  /*del item*/
        }

      if(comprueba_item(item,sera,N2,sigue,k)==1) /*Si ser2 coje el item*/
        {
        cuenta2=0;                   /*Cuenta para crecer*/
        setcolor(0);
        outtextxy(item[0],item[1],"o");      /*Borra el item*/
        setcolor(4);
        outtextxy(item[0],item[1],"*");
        item[0]=0;                   /*Inicializa la posicion*/
        item[1]=0;                   /*del item*/
        }
```