Inteligencia Artificial

Néstor Nápoles López

Diciembre de 2012

Documentación de proyecto final

En la modalidad de software, el proyecto consta de un programa gráfico que presenta a un personaje hipotético que busca su casa (Kibus). El proyecto es desarrollado a lo largo de cuatro etapas, donde se utilizan mecanismos cada vez más complejos en implementación para lograr tal fin.

Índice

1. Etapa primera del proyecto

1.1. Resumen de la etapa

Durante esta etapa, básicamente se construye la interfaz gráfica que se empleará a lo largo del proyecto. Algunos de los lineamientos que se siguen para construir dicha interfaz son los siguientes:

- Tener una rejilla de tamaño mínimo 15 filas por 15 columnas.
- Establecer la posibilidad de colocar obstáculos en el mapa.
- Establecer la posibilidad de colocar un punto de partida(casa).
- Manipular la posición del personaje en el mapa.
- Implementar un mecanismo para que el personaje regrese al punto de partida.

Al concluir esta etapa, se tienen los elementos mínimos para desarrollar las etapas más complicadas del proyecto. Un escenario, un personaje, obstáculos, y un punto de partida (posteriormente meta).

1.2. Propuesta de solución

Se propone resolver los problemas de esta etapa de la siguiente manera:

- Utilizar un lenguaje de programación de propósito general, donde sean implementables las soluciones más complicadas de etapas posteriores. Ésto en conjunto con una biblioteca gráfica que permita construir todos los elementos visibles por el usuario de manera accesible.
- Una vez resuelto ésto, para el problema del escenario, construir una matriz con un número de filas y columnas variable, modificable por el usuario. Preferentemente una estructura de datos que almacene datos importantes sobre cada celda de la rejilla.
- El personaje como una estructura de datos independiente al escenario, que pueda modificar su estado e interacción con el escenario a lo largo del programa.
- Los obstáculos y el punto de partida serán alojados como atributos de la estructura de datos que alberga al escenario.

1.3. Implementación

El lenguaje escogido para implementar el proyecto(hasta su etapa final), fue el **Lenguaje C Estandar**. En conjunto con la biblioteca **Allegro5.0**.

La matriz del escenario fue construida como un array bidimensional de estructuras (**struct**) que almacena coordenadas para cada celda del escenario, así como un tipo de celda.

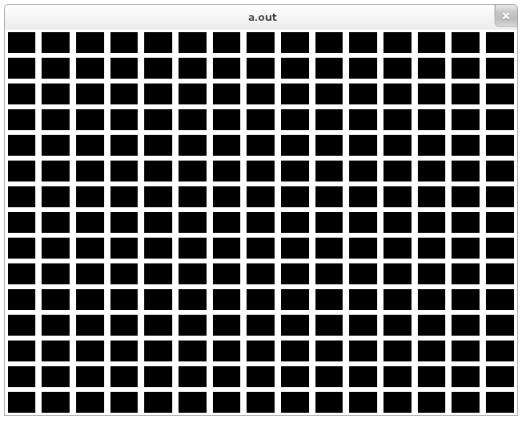

Figura 1: División de la pantalla en celdas

Posteriormente, los obstáculos y el punto de partida/meta son añadidos. El personaje es inicializado a la posición del punto de partida.

Figura 2: Vista del escenario con obstáculos, punto de partida y personaje.

1.4. Código

```
//Section 1
struct square{
  float x;
  float y;
  int type;
};
```

```
7
8  //Section 2
9  if(redraw&&al_is_event_queue_empty(event_queue)){
10   int i,j;
11   redraw=false;
12   al_clear_to_color(al_map_rgb(0,0,0));
13   if(set_random_map){
14     set_random_map=false;
15     randomize_obstacles(g_rows,g_cols,res_x,res_y,grid,obstacle_percent)
           ;
16   if(house_i!=-1&&house_j!=-1){
17     grid[house_i][house_j].type=HOUSE;
18     stack_delete(&stack);
19     kibus_i=house_i;
20     kibus_j=house_j;
21     stack_push(&stack,kibus_i,kibus_j);
22   }
23 }
24
25 //Section 3
26 if(kibus_return){
27   if(!stack_is_empty(stack)){
28     if(kibus_return_delay>9){
29       kibus_i=stack->x;
30       kibus_j=stack->y;
31       stack_pop(&stack);
32       kibus_return_delay=0;
33     }
34     kibus_return_delay++;
35   }else{
36   stack_push(&stack,kibus_i,kibus_j);
37   kibus_return=false;
38 }
39
40 //Section 4
41 case ALLEGRO_KEY_W:
42   if(set_kibus&&!kibus_return){
43     if(kibus_i>=1){
44       if(grid[kibus_i-1][kibus_j].type!=OBSTACLE){
45         kibus_i--;
46       }
47     }
48   }
49   break;
50
51 case ALLEGRO_KEY_E:
52   if(set_kibus&&!kibus_return){
53     if(kibus_i>=1&&kibus_j<=(g_cols-2)){
54       if(grid[kibus_i-1][kibus_j+1].type!=OBSTACLE){
55         kibus_i--;
56         kibus_j++;
57       }
58     }
59   }
60   break;
```

primeraetapa.c

En la figura anterior se observan cuatro secciones importantes de código:

- En la primera sección se define la estructura empleada para cada celda del escenario
- En la segunda sección se muestra la porción de código donde se modifica el tipo de celda donde se alojó el punto de partida. Además se inicializa la posición del personaje en el escenario.

- En la tercera sección se muestra el proceso de liberar la pila de posiciones recorridas por el personaje a partir de la posición de partida.
- En la cuarta sección se muestra la lectura de caracteres del teclado y su efecto en la posición del personaje.

1.5. Conclusiones

Al finalizar esta etapa, se tiene un módulo medular del proyecto, la implementación gráfica, así como las estructuras de datos principales, mismas que serviránpara soportar los algoritmos más complejos de etapas posteriores.

2. Etapa segunda del proyecto

2.1. Resumen de la etapa

En esta etapa, el algoritmo para la búsqueda de la casa es una línea de Bresenham, que se traza de la posícion actual del personaje (lejos de la casa) hacia la casa.

Al utilizar una línea de Bresenham en este contexto, surgen algunos problemas:

- La trayectoria construida por el algoritmo de la línea de Bresenham no siempre es recorrible, por la presencia de obstáculos.
- Esta presencia de obstáculos altera las vías que convergen a la casa, de manera que es necesario implantar una conducta emergente en el personaje si se pretende llegar a la casa en todos los casos (o al menos la mayoría).

2.2. Propuesta de solución

Conforme a la presencia de obstáculos, éstos están determinados por el tipo de celda. Cuando el personaje pretende recorrer la línea de Bresenham trazada hacia la casa, primero verifica que la siguiente celda que pretende visitar **no** sea un obstáculo.

En el caso del comportamiento emergente, se propone implementar un algoritmo que se active para todos los casos donde la siguiente celda a recorrer no sea viable. Este algoritmo buscará entre las 7 celdas restantes, una que sea recorrible por el personaje y tomará ese camino. Siempre tomará la decisión en sentido horario o anti-horario de manera persistente. Para los casos en que se exceda el número máximo de pasos (conforme al tamaño de la rejilla), se cambiará el sentido de la toma de decisiones.

2.3. Implementación

A lo largo de las ejecuciones, el personaje se verá en contextos como el siguiente, donde su camino ideal está bloqueado por un obstáculo y tendrá que recurrir a las conductas emergentes.

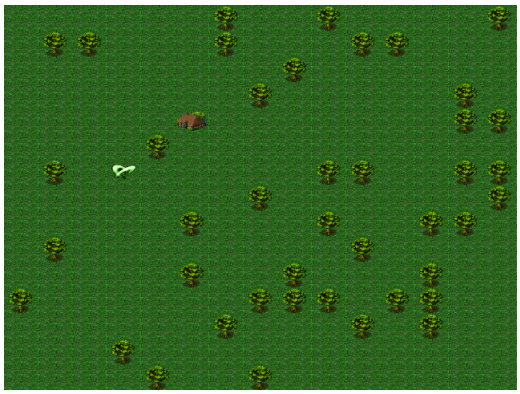

Figura 3: Contexto de bloqueo de la ruta propuesta por la línea de Bresenham.

En la siguiente figura se muestra una elección en sentido horario.

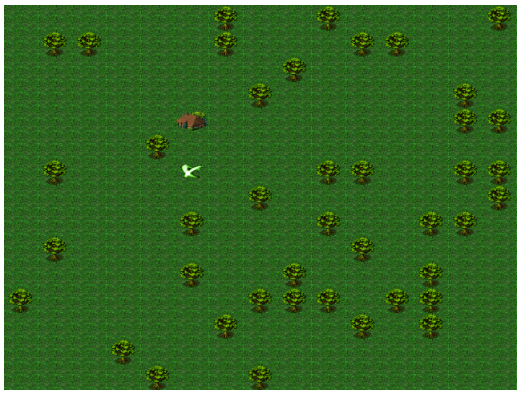

Figura 4: Toma de decisión en sentido **horario** ante un bloqueo.

Posteriormente, el mismo contexto en sentido anti-horario.

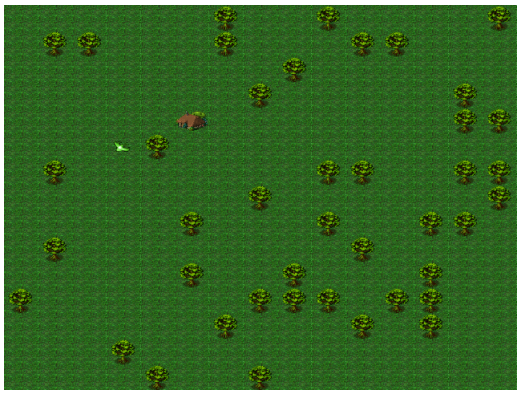

Figura 5: Toma de decisión en sentido **anti-horario** ante un bloqueo.

2.4. Código

```
//Section 1
void line(int x0,int y0,int x1,int y1,int *route_line_i,int *
    route_line_j){
  int dx=abs(x1-x0),sx=x0<x1?1:-1;
  int dy=abs(y1-y0),sy=y0<y1?1:-1;
  int err=(dx>dy?dx:-dy)/2,e2;

  if(x0==x1&&y0==y1){
    *route_line_i=-1;
    *route_line_j=-1;
    return;
  }
  e2=err;
  if(e2>-dx){
    err-=dy;
    x0+=sx;
  }
  if(e2<dy){
    err+=dx;y0+=sy;
  }
  *route_line_i=x0;
  *route_line_j=y0;
}

//Section 2
void recalculate_next_step(int kibus_i,int kibus_j,int *kibus_next_i,int
     *kibus_next_j,int emergent_reaction){
  int i=(*kibus_next_i)-kibus_i;
  int j=(*kibus_next_j)-kibus_j;
  if(emergent_reaction==LEFT){
    if(i==UP&&j==STAY){
      *kibus_next_i=kibus_i+UP;
      *kibus_next_j=kibus_j+LEFT;
    }else if(i==UP&&j==LEFT){
      *kibus_next_i=kibus_i+STAY;
      *kibus_next_j=kibus_j+LEFT;
    }else if(i==STAY&&j==LEFT){
      *kibus_next_i=kibus_i+DOWN;
      *kibus_next_j=kibus_j+LEFT;
    }else if(i==DOWN&&j==LEFT){
      *kibus_next_i=kibus_i+DOWN;
      *kibus_next_j=kibus_j+STAY;
```

```
41      }else if(i==DOWN&&j==STAY){
42        *kibus_next_i=kibus_i+DOWN;
43        *kibus_next_j=kibus_j+RIGHT;
44      }else if(i==DOWN&&j==RIGHT){
45        *kibus_next_i=kibus_i+STAY;
46        *kibus_next_j=kibus_j+RIGHT;
47      }else if(i==STAY&&j==RIGHT){
48        *kibus_next_i=kibus_i+UP;
49        *kibus_next_j=kibus_j+RIGHT;
50      }else if(i==UP&&j==RIGHT){
51        *kibus_next_i=kibus_i+UP;
52        *kibus_next_j=kibus_j+STAY;
53      }
54    }
55    //...
56  }
57
58  //Section 3
59  void kibus_next_step(int g_rows,int g_cols,int *kibus_i,int *kibus_j,
        struct square grid[][g_cols],int *last_step_i,int *last_step_j,int
        route_line_i,int route_line_j,int emergent_reaction){
60
61    int starting_direction_i=route_line_i-(*kibus_i);
62    int starting_direction_j=route_line_j-(*kibus_j);
63    int kibus_next_i=(*kibus_i)+starting_direction_i;
64    int kibus_next_j=(*kibus_j)+starting_direction_j;
65    int recalculations=0;
66
67    while(recalculations<8){
68      if(grid[kibus_next_i][kibus_next_j].type==OBSTACLE||
69         ((*last_step_i)==kibus_next_i&&(*last_step_j)==kibus_next_j)||
70         kibus_next_i>g_rows-1||kibus_next_i<0||kibus_next_j>g_cols-1||
              kibus_next_j<0){
71        recalculate_next_step(*kibus_i,*kibus_j,&kibus_next_i,&
              kibus_next_j,emergent_reaction);
72        recalculations++;
73      }else{
74        *last_step_i=(*kibus_i);
75        *last_step_j=(*kibus_j);
76        *kibus_i=kibus_next_i;
77        *kibus_j=kibus_next_j;
78        break;
79      }
80    }
81    if(recalculations==8){
82      *last_step_i=(*kibus_i);
83      *last_step_j=(*kibus_j);
84      *kibus_i=(*last_step_i);
85      *kibus_j=(*last_step_j);
86    }
87  }
```

segundaetapa.c

En este código se muestran tres secciones críticas de la etapa. Respectivamente:

- Se muestra un algoritmo optimizado de líneas de Bresenham, que será utilizado para trazar trayectorias ideales.
- Se muestra la búsqueda de la siguiente posición disponible en base al algoritmo emergente.
- Se muestra la función donde el personaje evalúa la viabilidad de la siguiente posición que sugiere la línea de Bresenham, y su llamado a la función de

búsqueda de posiciones disponibles cuando no es posible tomar la decisión ideal.

2.5. Conclusiones

Al finalizar esta etapa, se tiene un algoritmo que por primera vez, busca y encuentra el punto meta (en la mayoría de los casos. No obstante, este resultado se logra **conociendo de antemano la posición del punto meta**. No por el personaje, pero sí por el algoritmo de trazo de líneas.

3. Etapa tercera del proyecto

3.1. Resumen de la etapa

En esta etapa, el personaje es auxiliado por cinco agentes que sensan el calor en un escenario térmico. La posición donde se encuentra el punto meta, es también la posición más caliente del escenario. Los cinco agentes rodean al personaje y a través de sus resultados, guían al personaje por el camino hacia el calor(la meta).

3.2. Propuesta de solución

Para construir a los cinco agentes. Se podría emplear, por una parte; una estructura, similar a la de las celdas del escenario, y por otra parte; una interacción independiente de éstas estructuras hacia el escenario, similar al comportamiento del personaje.

Los agentes además, comparten sus hallazgos en una tabla, ahí alojan el calor registrado en las celdas que visitaron y se selecciona el camino más caliente para continuar la búsqueda. Es posible implementar esa tabla de resultados como una función, donde se toman los valores de búsqueda de los agentes y se lleva a cabo el proceso de selección. Una vez llevado a cabo este proceso, el personaje caminará en la dirección elegida.

En adición, debido a los bloqueos provocados por los obstáculos, es necesario que los agentes sean capaces de modificar su entorno, enfriando celdas que no parezcan conducir a la solución.

3.3. Implementación

Durante la implementación, se modificó el color de las celdas conforme al calor. Un color más obscuro representa una celda más fría, ésto para evidenciar el proceso de enfriamiento de los agentes. Un buffer almacenará cinco pasos electos por los agentes, y una vez almacenados los cinco, el personaje los recorrerá.

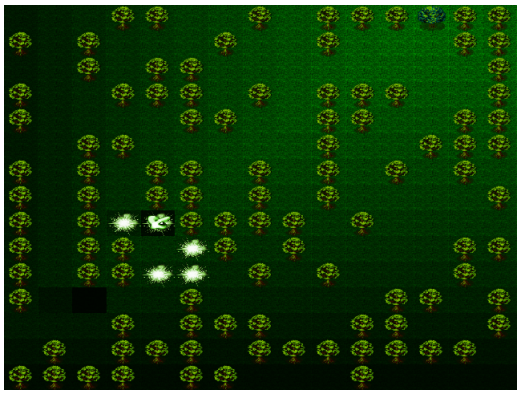

Figura 6: Agentes sensando el entorno y enfriando.

3.4. Código

```
//Section 1
void cooling_cell(int g_rows,int g_cols,struct square grid[][g_cols],int
     board_buffer[BEES][3]){
  int i,j,b[5]={0,0,0,0,0},delta=MAXWEIGHT/max(g_cols,g_rows);
  for(i=0;i<5;i++){
    for(j=0;j<5;j++){
      if(board_buffer[i][0]==board_buffer[j][0]&&
   board_buffer[i][1]==board_buffer[j][1]){
  b[i]++;
      }
    }
  }
  for(i=0;i<5;i++){
    printf("%d %d\n",i,b[i]);
    if(b[i]>1){
      if(grid[board_buffer[i][0]][board_buffer[i][1]].weight>delta){
  grid[board_buffer[i][0]][board_buffer[i][1]].weight-=delta;
      }
    }
    if(b[i]>2){
      if(grid[board_buffer[i][0]][board_buffer[i][1]].weight>delta){
  grid[board_buffer[i][0]][board_buffer[i][1]].weight-=delta;
      }
    }
  }
}

//Section2
int bee_board(int root_i,int root_j,struct bee bees[BEES]){
  int winner=0,i;

  for(i=1;i<5;i++){
    if(bees[winner].weight>bees[i].weight){
      winner=winner;
    }else if(bees[winner].weight<bees[i].weight){
      winner=i;
    }else{
      winner=rand()%2==0?winner:i;
    }
  }
  return winner;
}
```

```
42
43 //Section 3
44 void bee_propagation(int g_rows,int g_cols,int environment,int root_i,
       int root_j,struct square grid[][g_cols],struct bee bees[BEES]){
45   int next_bee;
46   int all_bees_ready=0;
47   int r=rand()%4;
48   int tries=0;
49   int mask[8]={1,2,4,8,16,32,64,128};
50   for(next_bee=0;next_bee<5;next_bee++){
51     bees[next_bee].i=root_i;
52     bees[next_bee].j=root_j;
53     bees[next_bee].weight=0;
54   }
55   next_bee=0;
56   while(tries<8&&!all_bees_ready){
57     if(environment&mask[tries]){
58       switch(mask[tries]){
59       case 128:
60   bees[next_bee].i=root_i;
61   bees[next_bee].j=root_j+LEFT;
62   break;
63       case 64:
64   bees[next_bee].i=root_i+DOWN;
65   bees[next_bee].j=root_j+LEFT;
66   break;
67       case 32:
68   bees[next_bee].i=root_i+DOWN;
69   bees[next_bee].j=root_j;
70   break;
71       case 16:
72   bees[next_bee].i=root_i+DOWN;
73   bees[next_bee].j=root_j+RIGHT;
74   break;
75       case 8:
76   bees[next_bee].i=root_i;
77   bees[next_bee].j=root_j+RIGHT;
78   break;
79       case 4:
80   bees[next_bee].i=root_i+UP;
81   bees[next_bee].j=root_j+RIGHT;
82   break;
83       case 2:
84   bees[next_bee].i=root_i+UP;
85   bees[next_bee].j=root_j;
86   break;
87       case 1:
88   bees[next_bee].i=root_i+UP;
89   bees[next_bee].j=root_j+LEFT;
90   break;
91       }
92       bees[next_bee].weight=grid[bees[next_bee].i][bees[next_bee].j].
             weight;
93       next_bee++;
94     }
95     tries++;
96     if(next_bee==5){
97       all_bees_ready=1;
98     }
99   }
100 }
```

terceretapa.c

Se muestran tres secciones de código críticas para el funcionamiento de esta etapa. Respectivamente:

- Se muestra la función de enfriamiento de celdas.
- Se muestra la función "tablero" que medirá y seleccionará los resultados de los agentes.
- Se muestra el algoritmo de propagación de los agentes sobre el escenario, siempre intentado que **todos** los agentes sean asignados a una posición del escenario(no siempre es posible), con el fin de sensar más celdas, en diferentes direcciones.

3.5. Conclusiones

Al concluir esta etapa del proyecto, se tiene un algoritmo que encuentra el punto meta, **sin conocer su posición exacta de ninguna manera**. Sin embargo, se tienen nociones del camino que podría llevar al punto meta, gracias al escenario térmico.

4. Etapa cuarta del proyecto

4.1. Resumen de la etapa

Esta etapa, es la etapa final del proyecto. En esta etapa, se parte de un desconocimiento absoluto de la ubicación de la casa, y a través de varias iteraciones de entrenamiento, se construye una representación del mapa como grafo ponderado. Esta representación almacenará la información que le permitirá al personaje llegar a la meta de manera eficaz y certera.

4.2. Propuesta de solución

Para la construcción de la representación ponderada del mapa, se pretende utilizar una estructura de datos similar a la que almacena el escenario, esta será construida durante cada iteración de entrenamiento, sin información alguna sobre previas iteraciones. Esta construcción servirá a la vez para establecer los pesos definitivos que tendrá cada celda del escenario en la estructura principal. Los mecanismos para lograr esto son el **máximo histórico**, **mínimo histórico** y la **media histórica** que se registran en cada iteración. Durante las iteraciones se realizan movimientos aleatorios para llegar al punto meta, el número total de pasos empleados es registrado y utilizado para construir el máximo, mínimo y media histórica.

Concluída la etapa de entrenamiento, se utiliza un algoritmo de selección entre los pesos del escenario para llegar a la casa.

4.3. Implementación

Proceso de entrenamiento del personaje, donde se construyen los pesos definitivos de cada celda del escenario. Cada iteración restante implica encontrar el punto meta a través de pasos aleatorios, y posteriormente el uso de los datos recabados en la construcción de pesos.

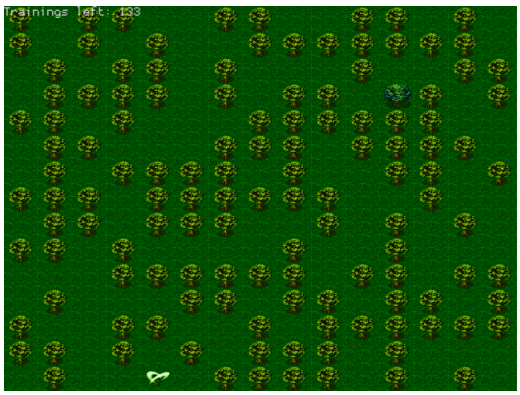

Figura 7: Transcurso de entrenamiento del personaje.

4.4. Código

```
//Section 1
  if(training){
  if(!graphic_training){
    while(kibus_i!=house_i||kibus_j!=house_j){
      environment_calculate(g_rows,g_cols,&kibus_i,&kibus_j,grid,
          balloons);
      steps++;
    }
  }else{
    if(kibus_i!=house_i||kibus_j!=house_j){
      environment_calculate(g_rows,g_cols,&kibus_i,&kibus_j,grid,
          balloons);
      steps++;
    }
  }
  if(kibus_i==house_i&&kibus_j==house_j){
    if(trainings==0){
      hist_max=steps;
      hist_min=steps;
      hist_avg=steps;
      kibus_i=tmp_kibus_i;
      kibus_j=tmp_kibus_j;
      trainings++;
      printf("steps=%d\nhist_max:%d\nhist_min:%d\nhist_avg:%d\n\n",steps
          ,hist_max,hist_min,hist_avg);
    }else{
      trainings++;
      punishment(g_rows,g_cols,grid,balloons,steps-hist_avg);
      printf("steps=%d\nhist_max:%d\nhist_min:%d\nhist_avg:%d\n\n",steps
          ,hist_max,hist_min,hist_avg);
      hist_max=steps>hist_max?steps:hist_max;
      hist_min=steps<hist_min?steps:hist_min;
      hist_avg=(hist_max+hist_min)/2;
      if(trainings==max_trainings){
        trained=true;
        training=false;
        trainings=0;
      }
      kibus_i=tmp_kibus_i;
      kibus_j=tmp_kibus_j;
    }
    steps=0;
  }
  set_text=true;
  trainings_remaining_text=true;
      }
   }
}

//Section 2
void punishment(int g_rows,int g_cols,struct square grid[][g_cols],int
    balloons[][g_cols],int delta){
  int i,j;
  printf("delta:%d\n",delta);
  for(i=0;i<g_rows;i++){
    for(j=0;j<g_cols;j++){
      if(balloons[i][j]!=0){
  grid[i][j].weight+=delta*balloons[i][j];
  balloons[i][j]=0;
      }
    }
  }
}

//Section 3
```

```c
int best_first(int g_rows,int g_cols,int *root_i,int *root_j,int delta,
    struct square grid[g_rows][g_cols],int balloons[g_rows][g_cols]){
  int environment[10]={INT_MAX,INT_MAX,INT_MAX,INT_MAX,INT_MAX,INT_MAX,
      INT_MAX,INT_MAX};
  int tmp_i=(*root_i)+UP;
  int tmp_j=(*root_j)+LEFT;
  int min=0;
  int i=0;
  if(tmp_i>=0&&tmp_i<=g_rows-1&&
     tmp_j>=0&&tmp_j<=g_cols-1&&
     grid[tmp_i][tmp_j].type!=OBSTACLE){
    environment[0]=grid[tmp_i][tmp_j].weight;
    if(balloons[tmp_i][tmp_j]>=1){
      environment[0]=INT_MAX-100+balloons[tmp_i][tmp_j];
    }
  }
  //...
}
```

cuartaetapa.c

Se muestran tres secciones de código críticas en la funcionalidad de esta etapa. Respectivamente:

- El proceso de entrenamiento que se lleva a cabo, de manera gráfica y oculta, que concluye cuando se ha cumplido con el número de iteraciones elegidas por el usuario.
- La rutina que aplica los cambios en el peso de las celdas del escenario.
- El método de búsqueda que emplea el personaje para seleccionar la dirección de sus pasos.

4.5. Conclusiones

Al concluir esta etapa, se tiene un personaje que en un estado inicial, tiene ninguna noción sobre la ubicación del punto meta. En un segundo estado, construye una representación de su entorno en base a recorridos aleatorios. Y finalmente, en un tercer estado busca el punto meta a través de la información construida en tiempo de ejecución.