8.7 Auto-Fred

8.7.1 Tres métodos para el control automático

Este apartado presenta tres métodos para controlar automáticamente a Fred, sustituyendo la intervención del jugador por un sistema que genera las pulsaciones de teclado necesarias para guiarlo hasta la salida del laberinto.

Llamamos a esta variante *Auto-Fred*. La idea es simple: basta con sustituir el estado 2 –Check Controls– por un algoritmo que determine, en cada paso, el valor de los controles sInput_x y sInput_y. Además, como Fred no evita a los enemigos, irá perdiendo vida cada vez que se encuentre con uno. Por ello, es necesario activar el modo de vida infinita, asignando sInfinitePower=true.

Estos son los tres métodos:

– Regla de la mano derecha.
– Exploración en profundidad.
– Algoritmo de Dijkstra.

8.7.2 Método 1: regla de la mano derecha

Este método tiene en cuenta que el laberinto es *simplemente conexo*. Es decir, sus paredes están conectadas y no quedan partes aisladas en el espacio transitable, de modo que es posible llegar desde un punto a cualquier otro. Por ello, se puede aplicar la *regla de la mano derecha* –o izquierda– para recorrerlo.

Imaginamos el laberinto sin cuerdas, como una proyección en planta. Partiendo de un punto cualquiera, elegimos una pared –por ejemplo, la situada a la derecha– y recorremos el laberinto siguiéndola hasta llegar a la salida.

La ventaja de este algoritmo es que no requiere guardar información sobre el camino recorrido: basta con mantener la dirección de la pared en todo momento.

Se fundamenta en mantener en cada paso la dirección de movimiento anterior, salvo cuando Fred está en una intersección. Entonces, toma la siguiente dirección en sentido horario o antihorario –según la mano elegida– y comprueba si la celda correspondiente permite avanzar. Si no hay una piedra, esta pasa a ser la nueva dirección de movimiento.

En caso contrario, recorre las direcciones en el sentido opuesto al elegido, hasta encontrar una que permita avanzar, que será la que adopte.

Como veremos, los algoritmos de movimiento del vampiro y del esqueleto se basan en esta idea.

La Figura 8.66 muestra el diagrama de flujo, que asume que la máquina de estados ya está inicializada, con Fred orientado hacia la izquierda y toma la mano derecha como criterio –sentido horario–. Primero, comprueba si Fred debe mantener la dirección; en ese caso, termina sin modificar los controles.

Observe que deberá mantener la dirección hasta que alcance una posición que permita el cambio. Es decir, si está caminando, debe cumplirse sFred_cx=0; si está en la cuerda, sFred_cy=0. También mantiene la dirección durante los saltos laterales o verticales y al cambiar de lado en la cuerda, hasta que estos movimientos se completen. Para indicarlo utilizamos la variable gAFLockDirection, como veremos a continuación.

Esto resulta necesario en el caso del salto vertical, ya que, una vez completado, el siguiente movimiento debe ser hacia arriba. Es decir, debe mantener la dirección un instante más antes de evaluar un posible cambio.

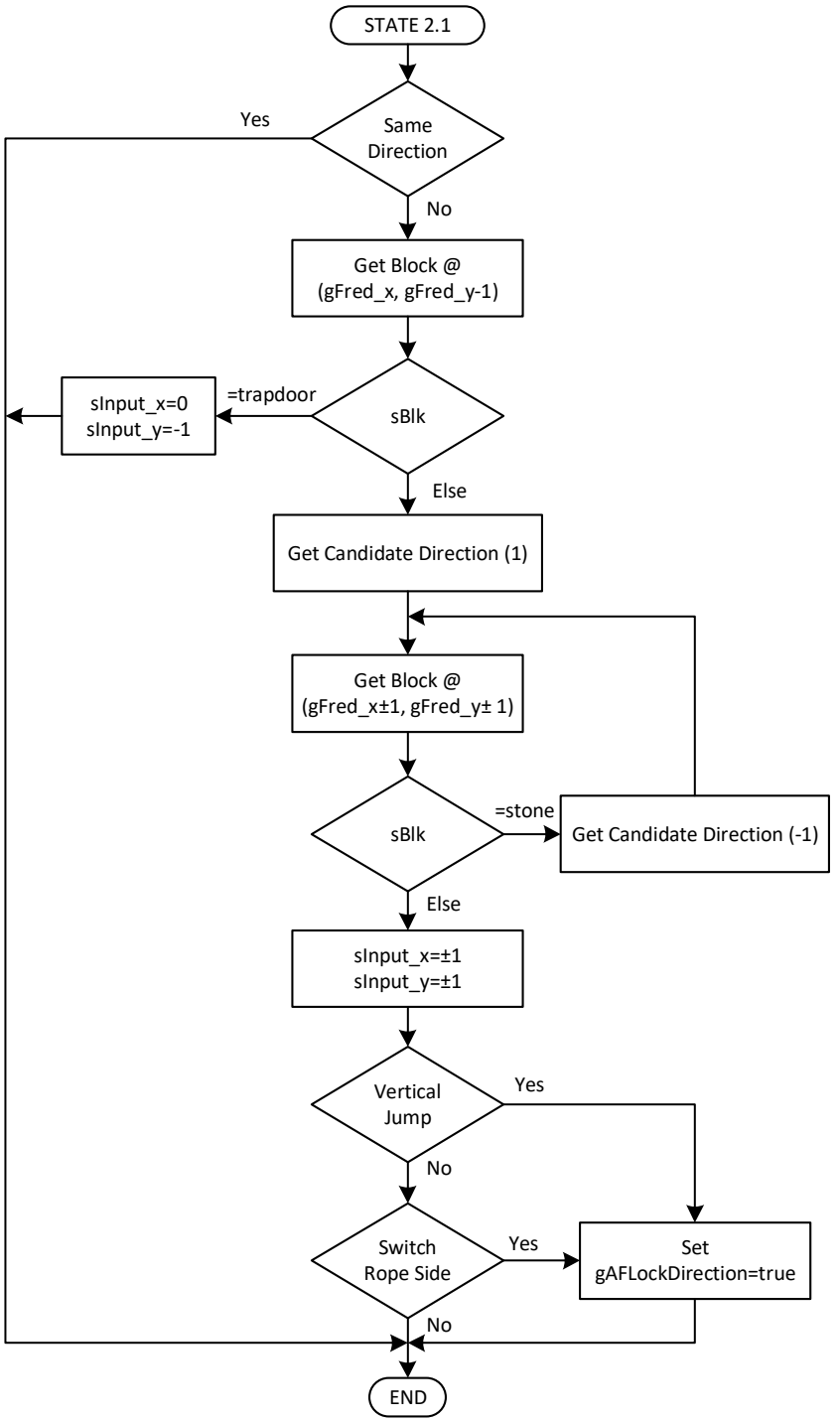

Figura 8.66. Diagrama de flujo del algoritmo de la mano derecha.

La razón es que el salto vertical no se traduce en un avance de celda, sino únicamente en subir a la cuerda, permaneciendo en la misma celda. Lo mismo sucede con el cambio de lado en la cuerda: una vez realizado el cambio, debe mantener la dirección de movimiento un instante más.

En caso contrario, comprueba si en la celda superior está la trampilla de salida. Si es así, toma dirección arriba, lo que le llevará a salir del laberinto. Si no es así, toma la primera dirección en sentido horario (1). Si encuentra una piedra, toma la siguiente en sentido antihorario (-1), y continúa del mismo modo –en sentido antihorario–hasta obtener una dirección válida.

Finalmente, asigna los valores de sInput_x y sInput_y –el diagrama denota la dirección correspondiente con ± 1– e inicializa la variable gAFLockDirection a true cuando Fred debe mantener la dirección en los casos de salto vertical y cambio de lado de la cuerda.

La Figura 8.67 muestra un ejemplo. Fred está en la celda (28, 1) siguiendo la pared inferior. La línea roja representa el recorrido que realizará con este algoritmo siguiendo la pared situada a su derecha –imaginando una vista en planta–.

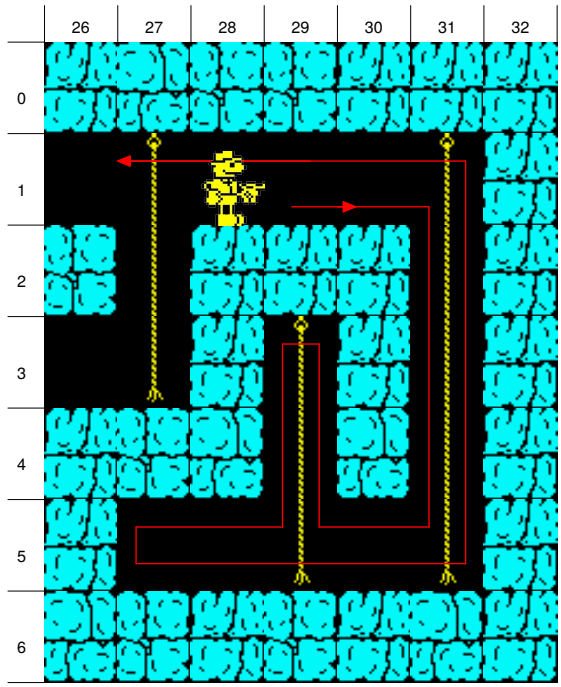

Figura 8.67. Trayectoria seguida por el algoritmo de la mano derecha.

8.7.3 Método 2: exploración en profundidad

Este método recorre el mapa de un modo similar al algoritmo que crea el laberinto. Como en el caso anterior, se basa en mantener la dirección de movimiento hasta llegar a una intersección. Si hay más de un camino posible, elige uno al azar –o simplemente el primero– y anota los demás en una lista de nodos. Cada nodo contiene la posición de la intersección, la dirección de llegada y la lista de direcciones pendientes desde ese punto.

Al llegar a un punto donde no es posible avanzar más, toma la dirección opuesta –izquierda-derecha o arriba-abajo– para deshacer el camino recorrido. En la siguiente intersección, consulta el nodo correspondiente en la lista y elige una de las direcciones pendientes, que elimina del nodo. Si, tras eliminarla, no quedan direcciones pendientes, toma la dirección de llegada y elimina el nodo de la lista.

De este modo se alcanza la salida del laberinto. A diferencia del método de la mano, esta aproximación requiere mantener en memoria la lista de nodos.

La Figura 8.68 muestra el diagrama de flujo. Como en el caso anterior, primero comprueba si debe mantener la dirección. Después, evalúa las direcciones posibles en la posición actual –dada por (gFred_x, gFred_y)–. Si encuentra la trampilla de salida, selecciona la dirección arriba y termina, para salir del laberinto. En caso contrario, actúa según el número de direcciones disponibles.

Para ello, coloca las direcciones en una lista, comenzando por la dirección contraria a la actual y añadiendo progresivamente el resto. Una vez completada, si contiene una o dos direcciones, toma la última. Esto garantiza que, o bien continúa avanzando por el pasillo o bien da la vuelta. En este último caso, toma la única dirección restante de la lista: la insertada en primer lugar, que corresponde a la dirección contraria a la de llegada.

Si en la lista hay más de dos direcciones, comprueba si este nodo –esta celda– ya está en la tabla. Si es así, extrae la última dirección y continúa. Si al hacerlo no quedan más direcciones, elimina el nodo.

Si el nodo no está en la tabla, toma la última dirección y anota el nodo con la celda y sus direcciones.

Finalmente, como en el caso anterior, asigna los valores de sInput_x y sInput_y e inicializa la variable gAFLockDirection a true cuando Fred debe mantener la dirección en los casos de salto vertical y cambio de lado de la cuerda.

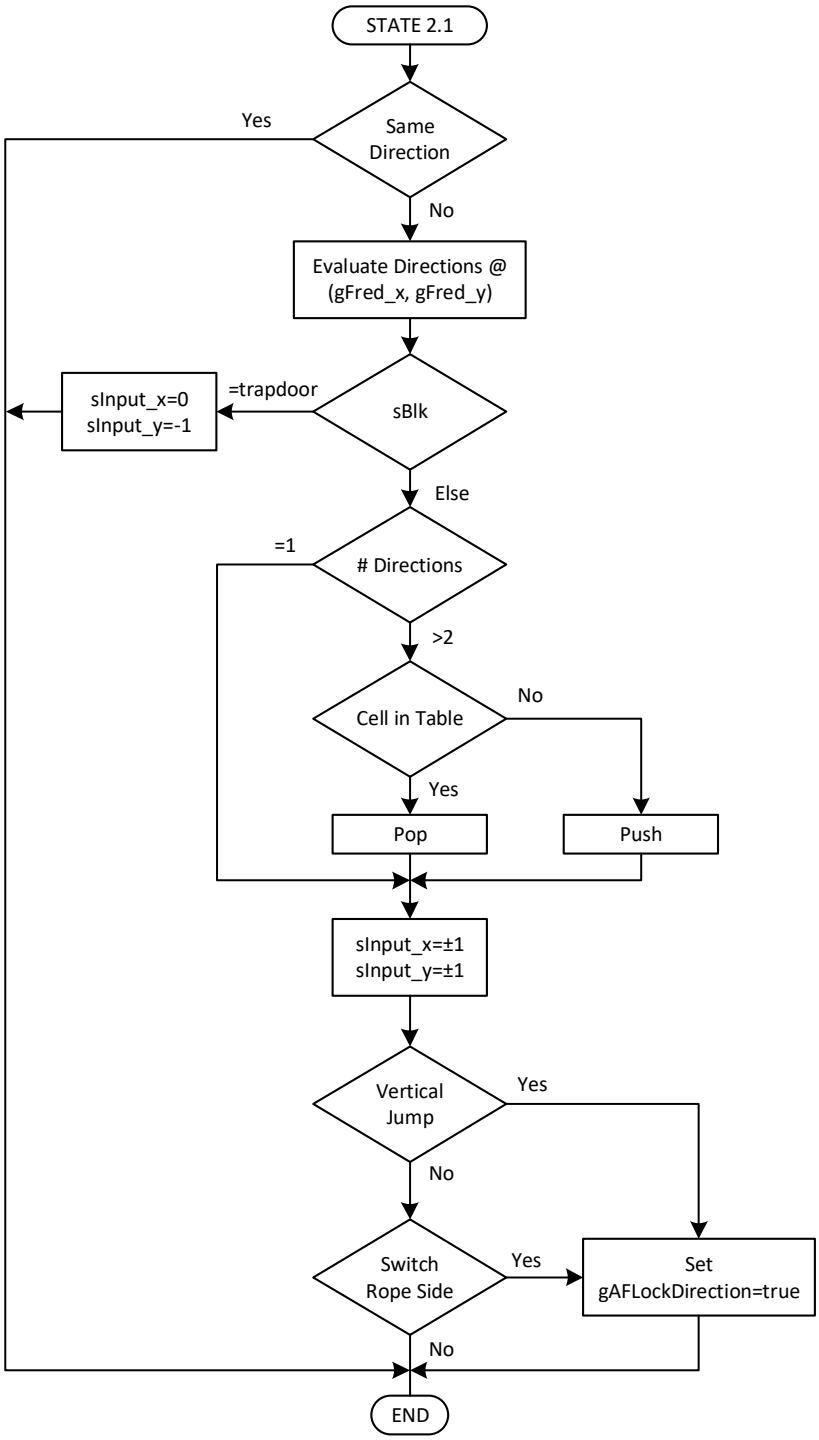

Figura 8.68. Diagrama de flujo del algoritmo de exploración en profundidad.

8.7.4 Método 3: algoritmo de Dijkstra[18]

En lugar de explorar el laberinto, se precalcula un mapa auxiliar de distancias para llegar desde cada celda hasta la salida.

El cálculo se realiza una vez generado el laberinto, antes de iniciar el movimiento, y se guarda en una estructura paralela a la del mapa. Para generarlo, basta con recorrer el laberinto comenzando desde la salida, de forma análoga a la empleada durante su construcción. A medida que el recorrido avanza, se asigna a cada celda su distancia, en número de celdas, hasta la salida.

Durante el avance se mantiene la dirección de movimiento hasta llegar a una intersección. Entonces la intersección se anota en una tabla, quedando pendiente de explorar. El proceso continúa por uno de los caminos disponibles hasta que no es posible avanzar más; en ese momento, se toma una intersección de la tabla y se reanuda el recorrido desde ella.

Una vez calculado el mapa de distancias, el movimiento de Fred resulta muy sencillo: basta con evaluar las celdas vecinas y avanzar hacia aquella que reduce la distancia a la salida.

En la práctica, esto constituye un *algoritmo de descenso del gradiente*, definido de forma discreta sobre la cuadrícula del laberinto[19].

Las Figuras 8.69 y 8.70 representan, respectivamente, el algoritmo de construcción del mapa auxiliar y las distancias calculadas sobre el mapa de depuración. Las celdas en blanco contienen ceros, que se omiten por claridad.

El proceso comienza asignando las distancias -1 y 1 a las celdas donde se encuentran la trampilla y su cuerda, respectivamente, y 0 al resto. Después, explora el laberinto incrementando la distancia a medida que avanza. Al encontrar una intersección, guarda las coordenadas de la celda en una tabla para volver a ella cuando se agota el camino. Cuando la tabla está vacía, el proceso termina.

---

[18] Este algoritmo fue ideado por el matemático y físico Edsger W. Dijkstra en 1956. Permite encontrar el camino más corto entre nodos de un grafo en el que se conocen las distancias entre nodos adyacentes –grafo ponderado–.

[19] Un algoritmo de descenso del gradiente consiste en avanzar, en cada paso, hacia la posición vecina con menor valor. En este caso, Fred se desplaza siempre hacia la celda cuya distancia a la salida es menor.

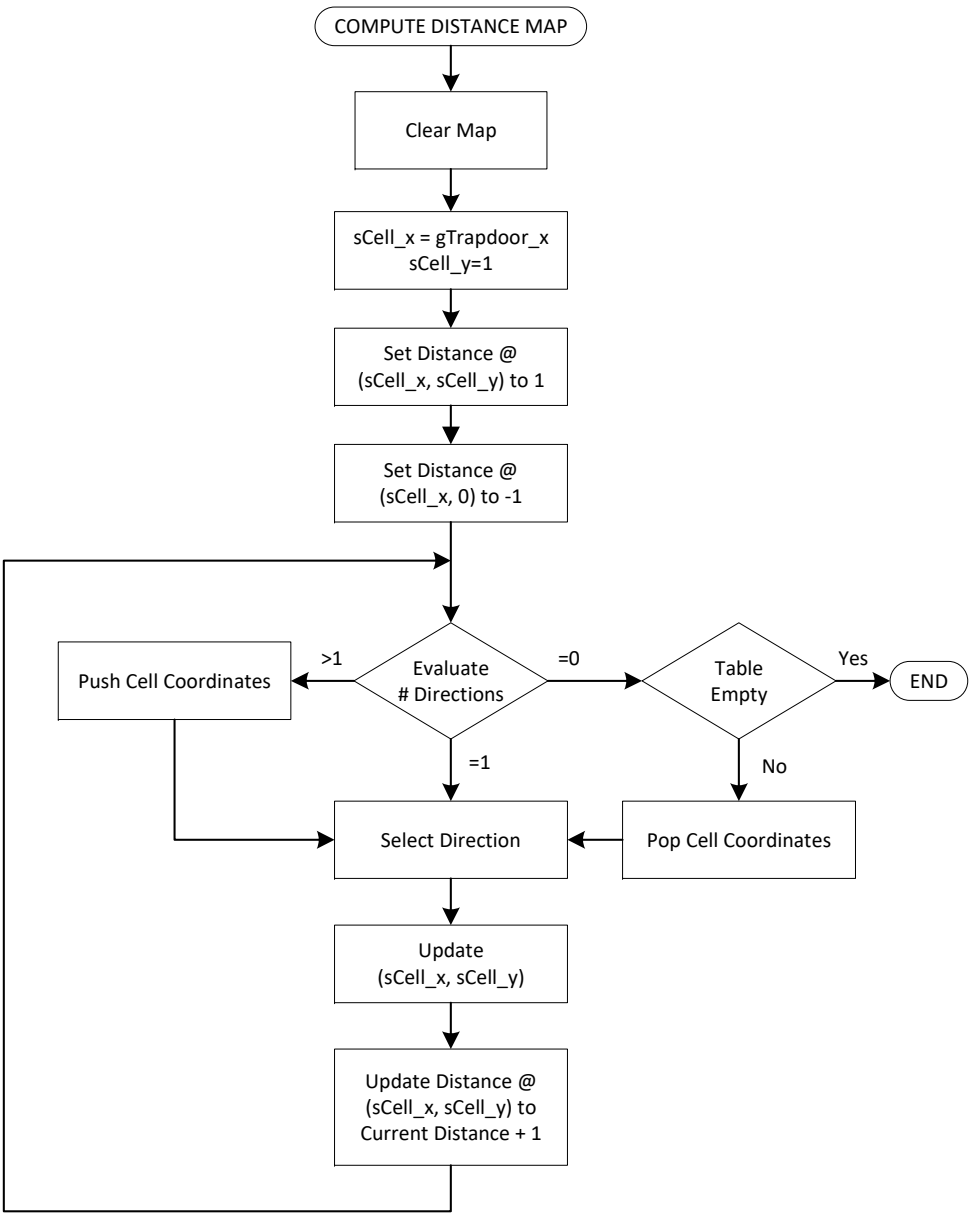

Figura 8.69. Diagrama de flujo para crear el mapa de distancias.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0																										-1							
1		37	38	39	40	41	42	43		17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	2	3	4	5	6	7	
2		36						44		18																		4				8	
3		35	34	33	32	31		45		19	20	21		43	42	41	42	43	44	45		15	14	13		7	6	5		15		9	
4		36				30						22				40						16		12		8				14		10	
5		37	38	39		29	28	27	26	25	24	23	24	25		39	38	37	36	35		17		11	10	9		15	14	13	12	11	
6														26						34		18											
7		211	212	213	214	215		221	222	223	224	225		27	28	29	30	31	32	33		19	20	21	22	23	24	25	26	27	28	29	
8		210				216		220				226												22								30	
9		209	208	207		217	218	219		181		227	228	229	230	231	232	233		239	240	241		23	24	25	26	27	28	29		31	
10				206						180								234		238		242										32	
11		203	204	205		183	182	181	180	179	178	177	176	175	174	173		235	236	237		243	244	245	246	247	248	249		35	34	33	
12		202														172												250		36		34	
13		201	200	199	198	197		175	174	173	172	171	170	169	170	171	172	173	174	175		181		255	254	253	252	251		37		35	
14						196		176						168						176		180		256						38		36	
15		163		193	194	195		177	178	179		165	166	167		269	268	267		177	178	179		257		43	42	41	40	39		37	
16		162		192				178		180		164						266						258				42					
17		161		191		181	180	179		181		163		113	112	111		265	264	263	262	261	260	259		45	44	43		57	58	59	
18		160		190						182		162				110										46				56		60	
19		159		189	188	187	186	185	184	183		161	160	159		109	108	107	106	105	104	103	102	101		47	48	49		55		61	
20		158												158								104		100				50		54		62	
21		157		151	150	149	150	151		161	162	163		157	156	155	154	153		107	106	105		99	98	97		51	52	53		63	
22		156		152		148		152		160								152		108						96						64	
23		155	154	153		147		153		159		145	146	147	148	149	150	151		109		91		93	94	95		73	72	71		65	
24		156				146		154		158		144								110		90		92				74		70		66	
25		157	158	159		145		155	156	157		143		121	120	119		113	112	111		89		91		77	76	75		69	68	67	
26						144				158		142				118		114		112		88		90				76				68	
27		139	140	141	142	143		161	160	159		141	140	139		117	116	115		113		87		89	88	87		77	78	79		69	
28		138				144								138						114		86				86				80		70	
29		137		147	146	145		131	132	133	134	135	136	137		119	118	117	116	115		85	86	87		85	84	83	82	81		71	
30		136						130								120						84										72	
31		135	134	133	132	131	130	129	128	127	126	125	124	123	122	121		87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	
32																																	

Figura 8.70. Mapa de distancias hasta la celda de salida para el mapa de depuración.