Documentación Práctica 1

1. Memoria del agente: variables de estado

```
44  private:
45      const int min_bateria_para_recargar = 3000;
46
47      // Tamaño del mapa
48      unsigned int map_size;
49
50      // Variables de estado (memoria del agente)
51      int fil, col, brujula;
52      Action ultimaAccion;
53      bool bien_posicionado;
54      bool girar_dcha;
55      bool inicio_juego;
56
57      // Salva informacion de estado
58      bool tiene_bikini, tiene_zapatillas;
59
60      int contador_forward;
61      int max_forward;
62
63      bool salida_obstaculo_izq;
64      bool salida_obstaculo_dcha;
65
66      // Memoria del agente (cuando aún no ha encontrado G)
67      vector<vector<unsigned char>> mapaProvisional;
68
68      pair<int,int> min_coordenadas;  // fila, col
69      pair<int,int> max_coordenadas;
70
71      queue<Action> accionCompuesta;
72      int objetivo; // Casilla objetivo
73
74      bool recargar_bateria;
```

Las variables de estado son la memoria del agente, es por esto por lo que son de vital importancia en el caso de tener que recorrer el mundo (pese a que su comportamiento sea puramente reactivo).

En primer lugar, la constante “min_bateria_para_recagar” establece un umbral a partir del cual es necesario cargar la batería (si es posible). Si su acción actual es la de recargar la batería, la variable “recargar_batería” se activa. La variable “inicio_juego” sólo es utilizado al comienzo de la partida.

Las variables “fil”, “col”, “brujula” almacenan la posición del agente y la van actualizando acordemente (son sacadas del tutorial). Más específicamente, almacenan inicialmente (si no es el nivel 0) una posición y orientación relativa en un mapa de memoria del agente el cual es provisional. En cuanto se encuentra una casilla de posicionamiento se actualizan en los valores reales y se actualiza la variable “bien_posicionado”.

Además, el agente también necesita saber si tiene los recursos de bikini y batería; dicha información la almacenan las variables booleanas “tiene_bikini” y “tiene_zapatillas”.

En cuanto a las acciones, “ultimaAccion” almacena la última acción realizada. La variable booleana “girar_dcha”, así como las variables “contador_forward” y “max_forward” proporcionan al agente un comportamiento aleatorio en caso de que no reconozca la situación con la que se encuentra. Evitan también que quede atrapado en un bucle. Así mismo, las variables “salida_obstaculo_izq” y “salida_obstaculo_dcha” rigen un comportamiento que permite al agente esquivar los obstáculos que se encuentra.

Siguiendo con el comportamiento, son también muy importantes las variables “en_accion_compuesta” y la cola de acciones. Sirven para el caso de que el agente perciba un objetivo (como puede ser una casilla de batería o localización) y necesite realizar una secuencia de acciones para llegar hacia él.

Por último, como la práctica se basa en descubrir el mundo, las parejas de valores “min_coordenadas”, “max_coordenadas”, “map_size”, y “mapaProvisional” ayudan al agente a almacenar todo lo que ve (historial) cuando está desorientado y/o desubicado.

2. Métodos auxiliares

El comportamiento de los métodos viene explicado en los comentarios. Cabe

```
76  // Métodos adicionales
77      /**
78       * @brief Restablece las variables de estado a su valor por defecto
79       */
80      void init();
81
82      /**
83       * @brief Reestablece los valores por defecto tras morir
84       */
85      void reestablecer();
86
87      /**
88       * @brief Devuelve true si la casilla objetivo se percibe por los sensores
89       * @param obj Caracter tipo de la casilla
90       * @param sensores
91       * @return true si se encuentra en la vision
92       */
93       int en_vision(unsigned char obj, Sensores sensores);
94
95       /**
96        * @brief Genera una secuencia de acciones para llegar a un objetivo
97        * @param obj índice de la casilla de visión a la que se desea llegar
98        * @post Actualiza la cola accionCompuesta
99        */
100      void perseguir(int obj);
101
102 // Métodos para percepción y posicionamiento
103     /**
104      * @brief Calcula la profundidad en función a la representación del sensor de
    terreno o de superficie
105      * @param i índice dentro del vector
106      * @return valor positivo con la profundidad
107      */
108     int profundidad(int i);

110     /**
111      * @brief Calcula la lateralidad dentro del vector del sensor de terreno/-
    superficie
112      * @param i indice dentro del vector de posicion
113      * @return valor entero con la lateralidad (negativo si izq, positivo si dcha)
    en funcion del centro 0
114      */
115     int lateralidad(int i);
116
117     /**
118      * @brief actualiza la variable mapaResultado en función de su visión (sensor
119      * @pre el jugador debe estar bien posicionado antes de invocar al método
120      */
121     void actualizaMapaResultado(Sensores sensores);
122
123     /**
124      * @brief Actualiza el mapa provisional con información actualizada de sensores
125      * @param sensores
126      */
127     void actualizaMapaProvisional(Sensores sensores);
```

```
129  // Métodos auxiliares para actualizar el mapa de memoria auxiliar y
130  // volcarlo en la memoria real
131      /**
132       * @brief Calcula el ángulo de giro correspondiente (orientacion_real -
     brujula_relativa)
133       * @param orientacion_real
134       * @return Grados de giro en sentido horario
135       */
136      int rotacion (int orientacion_real);
137
138      /**
139       * @brief Genera una submatriz de dimensiones reducidas, ajustando
     mapaProvisional
140       * en función del área recorrida (min_coordenadas y max_coordenadas)
141       * @post Modifica mapaProvisional
142       * @post Modifica las variables fil, col
143       */
144      void submatriz();
145
146      /**
147       * @brief Convierte mapaProvisional en su matriz traspuesta
148       * @post Modifica mapaProvisional
149       * @post Modifica las variables fil, col
150       */
151      void traspuesta();
152
153      /**
154       * @brief Rota 90 grados en sentido horario la matriz mapaProvisional
155       * @post Modifica mapaProvisional
156       * @post Modifica las variables fil, col
157       */
158      void rotar_90_grados();
159
160      /**
161       * @brief Rota 180 grados en sentido horario la matriz mapaProvisional.
162       * @post Modifica mapaProvisional
163       * @post Modifica las variables fil, col
164       */
165      void rotar_180_grados();
166
167      /**
168       * @brief Rota 270 grados en sentido horario la matriz mapaProvisional
169       * @post Modifica mapaProvisional
170       * @post Modifica las variables fil, col
171       */
172      void rotar_270_grados();
173
174      /**
175       * @brief Vuelca la información recopilada en mapaProvisional en mapaResultado
176       * realizando las transformaciones necesarias para la vericidad de esta.
177       * Posteriormente limpia mapaProvisional (ya no es necesaria).
178       * @param sensores
179       * @pre Se debe de tener información correcta en los sensores, es decir, se debe
180       * de estar ubicado en la casilla de posicionamiento G
181       */
182      void transforma_matriz(Sensores sensores);
```

3. Descripción del comportamiento del agente:

El comportamiento del agente queda definido por la función “think”. Esta consta de dos partes, tratadas en el siguiente orden:

(a) Actualización de las variables del agente en función de la información sensorial percibida.

1. Si la última acción fue morir, restablece las variables de memoria.

2. Actualiza las variables de posición en función de la última acción (en caso de que el nivel sea 0, obtiene dicha información directamente desde los sensores).
3. Actualiza las variables relacionadas con el mapa:
    a. Si está bien posicionado, actualiza mapa resultado con su percepción sensorial.
    b. Si no está bien posicionado:
        i. Si se encuentra en la casilla de posicionamiento, vuelca toda su memoria provisional en el mapa resultado y activa la variable “bien_posicionado”, así como las variables de posicionamiento “fil”, “col” y “brujula” con sus valores reales.
        ii. Si está en cualquier otra casilla, actualiza el mapa de memoria provisional con la información sensorial obtenida.
4. Actualiza el resto de información de variables de entorno:
    a. Si se encuentra en la casilla de Bikini ‘K’ y la variable “tiene_bikini” está desactivada, la activa (coge el bikini).
    b. Si se encuentra en la casilla de Zapatillas ‘D’ y la variable “tiene_zapatillas” está desactivada, la activa (coge las zapatillas).
    c. Si se encuentra en la casilla de Recarga ‘X’ y la batería actual es menor al mínimo para recargar (“min_bateria_para_recargar”), activa la variable “recargar_bateria”.
5. Por último, comprueba si se encuentra en mitad de una acción compuesta. Se pueden dar los siguientes casos:
    a. Se encuentra en mitad de una acción compuesta pero ha colisionado, entonces vacía la cola y elimina el objetivo.
    b. Si no hay ningún objetivo (es negativo), intenta encontrar un objetivo en su visión y perseguirlo, con el siguiente orden de prioridades:
        i. Prioridad 0: Recargar batería (si la batería es menor al mínimo).
        ii. Prioridad 1: Localizarse (si no está bien posicionado)
        iii. Prioridad 2: Obtener bikini (si no lo tiene)
        iv. Prioridad 3: Obtener zapatillas (si no lo tiene)

    * El método “en_vision” actualiza el objetivo si lo ve y “perseguir” genera una secuencia de acciones para llegar al objetivo (actualiza la cola “accionCompuesta”). En caso afirmativo, se actualiza la variable “en_accion_compuesta”.

(b) Determinación de la acción a realizar en función a la percepción y las variables.

Las acciones se comprueban en el orden establecido, en cuanto una de ellas se cumpla, el resto se descartan.

1. Si se encuentra recargando batería, no se mueve (“actIDLE”) hasta que se recargue al máximo la batería.
2. Si se encuentra a un lobo o un aldeano en la superficie, tampoco se mueve y espera a que se vayan.
3. Si está en mitad de una acción compuesta, la continúa ejecutando.

4. Si no tiene bikini ni zapatillas, tiene menos de 3500 de batería y se encuentra con bosque o agua en frente (sin estar dentro de ella), decide girar para ahorrar batería.
5. Si está esquivando un obstáculo (tanto a derecha como a izquierda, el orden en este caso es irrelevante), lo termina de esquivar (es una acción de dos pasos).
6. Si reconoce un obstáculo (a su izquierda o derecha está el obstáculo y una casilla delante del mismo está la salida), comienza a esquivarlo.
7. Si no puede avanzar (se da el caso de que se encuentra justo delante de él, un muro, un precipicio, o agota el máximo de avances hacia delante consecutivos), determina un giro aleatoriamente a derecha o izquierda.
8. En caso de que no se de ninguna situación de las anteriores, avanza hacia delante.

Una vez determinada la acción a realizar, actualiza la variable contadora de avances (“contador_forward” y “max_forward”) y actualiza la variable “ultimaAccion” con la nueva acción determinada. Finalmente la devuelve para que el agente la ejecute.