Práctica 1: Agente Reactivo

Mundos de Belkan

Pablo Olivares Martínez

1. Funcionamiento del agente

La práctica nos pedía la realización de un agente reactivo. Para ello, he declarado variables de estado y métodos para conseguir dicho objetivo. La idea principal sobre el funcionamiento de mi agente consistía en dividir las acciones en etapas e ir escogiendo la más útil en cada momento para realizar una acción. Al comenzar, `etapa = inicio`, realizo una espiral para encontrar algún muro. Los cálculos de dicha espiral se realizaban basándome en la *sucesión de Fibonacci*. Cuando éste encuentra un muro o *frontera* (casilla que no puede atravesar, ya sea por agentes externos, por ser muros, precipicios o casillas con coste adicional sin objeto), `etapa = muro`, el agente comenzará a seguir dicha frontera durante al menos 25 pasos y hasta que el generador de números aleatorios distribuidos uniformemente en [0,1] genere un número con una probabilidad inferior a `LEAVE_WALL_PROB`. En ese caso, el agente dejará la frontera, `etapa = dejarMuro`, girando a la izquierda y comenzando a comportarse de manera simple, es decir, `etapa = simple`. El comportamiento simple no es más que el realizado durante el tutorial de la práctica. En caso de ver un objeto o una casilla especial que necesite en su campo de visión, `etapa = especial` , éste se dirigirá a él si nada se interpone en su camino. Si dicha casilla es de recarga, en caso de cumplir el criterio de que su cociente entre bateria y vida es menor que `CHARGE_PROP`, comenzará a recargar su batería en la `etapa = bateria`. Cuando esté cargada con el porcentaje exigido, se irá. Si un aldeano se interpone en su camino, el agente lo esquivará en `etapa = esquivar`. Finalmente, si el agente no descubre más del 2% del mapa respecto a la anterior comprobación (en mi caso 750 pasos, para evitar así que se quede en zonas sin salidas), `etapa = morir`, este se suicidará cuando encuentre un precipicio para reaparecer en otro punto volviendo a empezar con `etapa = inicio`.

Además, el agente también considera aspectos como aparecer en zonas desfavorables. Al finalizar la ejecución, el agente inferirá sobre las casillas no descubiertas a partir de las conocidas, cual es la más probable y así pintar en `mapaResultado`.

2. Variables de estado

Primero he declarado dos enumerados para facilitar la comprensión del programa:

```
/// @brief Enumerado con las distintas direcciones del entorno del agente
enum Entorno { frenteIzq, frente, frenteDer, derecha, atrasDer, atras, atrasIzq,
izquierda};
/// @brief Enumerado con las diferentes etapas del agente
enum Etapa {simple, inicio, muro, dejarMuro, esquivar, especial, morir,
recargar};
```

A continuación, haré una breve descripción de la utilidad de cada variable de estado declarada en mi programa:

```
// Enteros constantes del programa
static const int VISION_DEPTH = 3, UNKNOWN = -1, MAX = 100, MIN_STEPS_LEAVE_WALL
= 25;
// Constates con la probabilidad de dejarMuro, la proporción de carga deseada y
el porcentaje descubierto
const float LEAVE_WALL_PROB = 0.05, CHARGE_PROP = 5.0 / 3.0, DISCOVER_PERC =
0.03;
// Cuenta los pasos relativos para calcular el porcentaje de mapa, mientras que
pasosFrontera y controladorPuerta
int pasos, pasosFrontera, controladorPuerta;
// Sensores del mapa y del mapa auxiliar
int fil, col, auxFil, auxCol, brujula;
// Variables usadas para realizar la espiral de la etapa inicial
int fib_n0, fib_n1, contIni;
// Porcentajes de mapa actual y anterior
float perDescPrev, perDesc;
// caminoASeguir indica que se ha establecido un camino a la casilla especial y
que haga lo que ruta
bool caminoASeguir;
// terrenoIdeal indica si mi personaje está sobre una casilla sin zapatillas o
bikini respectivamente
bool terrenoIdeal;
// Si es verdadera, no realiza una etapa
bool accionDecidida;
// Indican si tenemos dichos objetos
bool zapatillas, bikini;
// Nos dicen si la accion simple gira a la derecha o izquierda y bien_situado si
conocemos todos los datos del sensor relativo a la posición
bool girar_derecha, bien_situado;
// Booleanos usados mientras seguimos el muro, tanto para detectarlo y encontrar
huecos para pasar por ellos
bool fronteraEncontrada, posiblePuerta, puerta;
// Indica la etapa
Etapa etapa;
// Indica la anterior acción
Action ultimaAccion;
// Almacena los pasos a seguir hasta la casilla especial
list<unsigned char> ruta;
// Alamcena el entorno del agente
vector<unsigned char> entorno;
// Mapa que graba la visión mientras el agente no esté bien situado
vector<vector<unsigned char>> mapaAux;
```

3. Métodos privados

A pesar de tenerlos documentados en *ComportamientoJugador.hpp*, pongo aquí una copia:

```
    /**
     * @brief mensaje
     * Función que muestra el mensaje de información
     *  del comportamiento.
```

```
     */
    void mensaje(Sensores sensores);
    /**
     * @brief mapearVision (Sensores sensores)
     * Función que mapea la vision del agente.
     * @param mapa Mapa que queremos pintar (auxiliar o resultado).
     * @param vision Sensores que usaremos para registrar el campo de vision.
     * @param orientacion Orientación del agente para pintar el mapa.
     * @param f Fila en la que se encuentra el agente.
     * @param c Columna en la que se encuentra el agente.
     */
    void mapearVision(vector<vector<unsigned char>>& mapa,
                      vector<unsigned char> vision, int orientacion, int f, int
c);
    /**
     * @brief incluirMapa
     * Función que incluye el mapa auxiliar en el mapa principal.
     */
    void incluirMapa();

    /**
     * @brief encontrarEspecial
     * Función que encuentra el punto de especialidad, ya sea un bikini,
     * zapatillas, recarga o posicionamiento.
     * @param sensores Sensores que usaremos para explorar el campo de vision.
     * @return int Posición en el campo de visión del elemento espacial.
     */
    int encontrarEspecial(Sensores sensores);

    /**
     * @brief evaluarTerreno
     * Función que evalúa el terreno en el que se encuentra el agente.
     * @param casilla Casilla que se quiere evaluar. Dice si es un terreno
     * idoneo para caminar o no y lo graba en el booleano correspondiente.
     */
    void evaluarTerreno(char casilla);

    /**
     * @brief evitarAldeano
     * Función que indica si hay un aldeano en la casilla de superficie.
     * @param superficie Posición donde se desea evalua si hay un aldeano.
     * @return true Hay un aldeano.
     * @return false No hay un aldeano.
     */
    bool evitarAldeano(char superficie);

    /**
     * @brief setEntorno
     * Establece el entorno del agente, es decir, las casillas que le rodean.
     */
    void setEntorno();

    /**
     * @brief esFrontera
     * Indica si la casilla es frontera, es decir, si podemos pasar por ella
     * o no. Se cinsidera frontera los puntos del mapa que sean muros y
     * precipicios (los he denomiado hardFrontier) y aquellos que sean más
     * costosos de pasar por no disponer de los objetos necesarios
```

```
     * (softFrontier).
     * @param casilla Casilla que se quiere evaluar.
     * @return true Es una frontera.
     * @return false No es una frontera.
     */
    bool esFrontera(char casilla);

    /**
     * @brief determinaCamino
     * Función que establece el camino a una casilla especial. Éste se almacena
     * en la variable ruta.
     * @param dest Casilla destino desde el campo de visión de los sensores.
     * @return true Ha podido establecer el camino.
     * @return false El camino que ha intentado establecer no existe o no se
     * cumplen las condiciones para establecerlo.
     */
    bool determinaCamino(int dest);

    /**
     * @brief sigueCamino
     * En caso de que haya una ruta disponible, seguirá el camino que ha
     * establecido en determinaCamino().
     * @return Action Siguiente acción a realizar.
     */
    Action sigueCamino();

    /**
     * @brief accionSimple
     * Función que determina un comportamiento sencillo del agente. Se basa en
     * el comportamiento realizado en el tutorial de la práctica.
     * @param sensores Sensores que usará el agente para decidir su acción.
     * @return Action Siguiente acción a realizar.
     */
    Action accionSimple(Sensores sensores);

    /**
     * @brief inicioAgente
     * Inicio de la etapa de exploración. El agente realizará una espiral
     * mientras las condiciones sean favorables. Está basado en la sucesión de
     * Fibonacci.
     * @param sensores Sensores que usará el agente para decidir su acción.
     * @return Action Siguiente acción a realizar.
     */
    Action inicioAgente(Sensores sensores);

    /**
     * @brief seguirFrontera
     * Función que se encarga de seguir la frontera que ha encontrado.
     * @param sensores Sensores que usaremos para detectar la frontera y tomar
     * decisiones respecto al estado actual de esta.
     * @return Action Siguiente acción a realizar.
     */
    Action seguirFrontera(Sensores sensores);

    /**
     * @brief randomGenerator
     * Función que genera un número aleatorio distribuido uniformemente entre 0
     * y 1.
```

```
     * @return float Número aleatorio entre 0 y 1.
     */
    float randomGenerator();

    /**
     * @brief calcularPerDesc
     * Función que calcula el porcentaje de mapa descubierto.
     * @return float Porcentaje de mapa descubierto.
     */
    float calcularPerDesc();

    /**
     * @brief reset
     * Función que resetea el mapa auxiliar y las variables de estado tras
     * morir.
     */
    void reset();

    /**
     * @brief rellenar
     * Función que rellena el mapa principal infiriendo el valor de las casillas
     * desconocidas a partir de las conocidas. Hay dos versiones, en el programa
solo
     * utilizo la primera.
     */
    void rellenar();
    void rellenarV2();

    /**
     * @brief voyAMorir
     * Función que indica si el agente morirá en la próxima acción. Esta función
es
     * la que está implementada ya en la práctica y adaptada a mi problema.
     * @param accion Acción decidida.
     * @param sensores Sensores que usará el agente para decidir su acción.
     * @return true El agente muere en la siguiente acción.
     * @return false El agente no muere en la siguiente acción.
     */
    bool voyAMorir(Action accion, Sensores sensores);
```