Memoria Practica 1 IA:

David Martínez Díaz GII-ADE

En primer lugar, realizamos en clase el tutorial básico, para ir quedándonos con los conceptos básicos y las funcionalidades más fáciles del programa, como movimientos de andar hacia adelante y girar.

En la primera toma de contacto que tuve, descubrí como utilizar la casilla especial G e ir explorando el mapa casilla a casilla poco a poco, consiguiendo unos porcentajes muy bajos en los primeros mapas como del 12%:

```
if(sensores.terreno[0] == 'G' and !bien_situado){
        fil = sensores.posF;
        col = sensores.posC;
        bien_situado = true;
}
if(bien_situado){
        mapaResultado[fil][col] = sensores.terreno[0];
}
```

Mas adelante, probe a crearme una matriz auxiliar donde podía ir añadiendo el camino que había recorrido hasta el momento, con la finalidad de no tener que pasar mas veces por ese sitio, sin embargo, al darme muchos errores me di por vencido con esa idea:

```
matrizRecorrida[fil][col] = true;

next_fil = fil;
next_col = col;

switch (brujula){
        case 0: next_fil--; break;
        case 1: next_col++; break;
        case 2: next_fil++; break;
        case 3: next_col--; break;
}

if(matrizRecorrida[next_fil][next_col]){
        cout << "La casilla de delante ha sido descubierta" << endl;
}
else{
        cout << "La casilla de delante no ha sido descubierta" << endl;
}
```

Una semana después, decidí mejorar el campo de visión y empecé a diseñar un algoritmo que me cogiese la matriz triangular que se formaba gracias al sensor de visión, se me quedo un código bastante grande pero necesario para conseguir mayores porcentajes en los mapas:

```
    for(int j=0; j<tam_vision+1; j++){

            if(mapaResultado[fil+sensor_fila][col+sensor_columna] == '?')
                    mapaResultado[fil+sensor_fila][col+sensor_columna] = sensores.terreno[i];

            mapaResultado[fil+sensor_fila][col+sensor_columna] = sensores.terreno[i];
            sensor_columna++;
            i++;
    }

    tam_vision += 2;
    sensor_fila ++;
    sensor_fila --;
    saltos--;
```

Ese mismo día, se me ocurrió la idea de querer implementar un algoritmo basado en costes, donde decidiese ir a donde menos batería le fuese a consumir, dándole las primeras ideas lógicas al programa, consiguiendo grandes avances aunque con muchos fallos:

```
if(cont_coste > 5){

        int coste_zona_a = 0,
                coste_zona_b = 0;

        int coste_terreno = 0;

        for(int j=0; j<16; j++){

                switch(sensores.terreno[j]){

                        case 'T':
                                coste_terreno = 2;
                        break;

                        case 'A':

                                if(bikini){
                                        coste_terreno = 10;
                                }
                                else{
                                        coste_terreno = 200;
                                }

                        break;

                        case 'B':

                                if(zapatillas){
                                        coste_terreno = 15;
                                }
```

Una semana después, retomando el trabajo, decidí pulir algunos detalles de estos códigos, por ejemplo, dándoles un grado de arriesgo al algoritmo base de exploración para que a veces se atreviese a ir de frente, aunque hubiese casillas de mucho coste, entre otras cosas:

```
grado_arriesgo = 1 + rand() % (4 + 1 - 1) ;

switch(grado_arriesgo){

        case 1:
                grado_arriesgo = 10000;
        break;

        case 2:
                grado_arriesgo = 500;
        break;

        case 3:
                grado_arriesgo = 100;
        break;

        case 4:
                grado_arriesgo = 1;
        break;
}

cout << "Diferencia de coste: " << diferencia_coste << endl;

if(diferencia_coste < grado_arriesgo and sensores.superficie[2] == '_'  and sensores.terreno[2] !=
```

Una de las decisiones mas claves a la hora de pulir el código, fue la implementación de un algoritmo de búsqueda de la casilla de posicionamiento ‘G’, la cual me permitía empezar a pintar el mapa de manera muy rápida:

```
portamientoJugador::AlgoritmoPosicionamiento(Sensores sensores){

if(!ir_hacia_zonaA and !ir_hacia_zonaB and !ir_recto){

        for(int j=0; j<16; j++){

                if(sensores.terreno[j] == 'G'){

                        casilla_encontrada = true;
                }

                if ((j == 1 or j == 4 or j == 5 or j == 9 or j == 10 or j == 11) and casilla_encontrada){
                        ir_hacia_zonaA = true;
                        casilla_encontrada = false;
                }
                else if ((j == 3 or j == 7 or j == 8 or j == 13 or j == 14 or j == 15) and casilla_encontr
                        ir_hacia_zonaB = true;
                        casilla_encontrada = false;
                }
                else if ((j == 2 or j == 6 or j == 12) and casilla_encontrada){
                        ir_recto = true;
                        casilla_encontrada = false;
                }
        }
}
```

Mas tarde, mejore el código haciendo que buscase también las casillas de bikini y de las deportivas para que consumiese mucha menos batería:

```
toJugador::AlgoritmoBikiniDeportivas(Sensores sensores){

cia_zonaA and !ir_hacia_zonaB and !ir_recto){

or(int j=0; j<16 and (casilla_encontrada == false); j++){

        if((bikini == false and sensores.terreno[j] == 'K') or (zapatillas == false and sensores.terreno[j] == 'D')){

                casilla_encontrada = true;
        }
```

Luego también me di cuenta de la existencia de la casilla especial ‘X’ de recargar batería, la cual la implemente a mi algoritmo de búsqueda de casillas especiales y le di la interpretación de que se quedase a descansar unos cuantos ciclos para llenar la batería a la vez que lo unifique todo en una sola función:

```
if((bien_situado == false and sensores.terreno[j] == 'G') or (bikini == false and sensores.terreno[j] == 'K') or (zapatillas == false and sensores.terreno[j] == 'D')){
if((bien_situado == false and sensores.terreno[j] == 'G') or (bikini == false and sensores.terreno[j] == 'K') or (zapatillas == false and sensores.terreno[j] == 'D') or (fin_recarga
```

Una de mis ultimas implementación fue crear un algoritmo que detectase si había puertas por las que poder pasar y no quedarse encerrados en una sola parte del mapa, lo cual ayudo a adquirir un porcentaje mayor a nivel general:

```
256 + void ComportamientoJugador::AlgoritmoPuerta(Sensores sensores){
257
    -         if(cont_coste < 50){
258 +         if(!ir_hacia_zonaA and !ir_hacia_zonaB and !ir_recto){
259 +
260 +                 for(int i=9; i<16; i++){
261
262 +                         if(sensores.terreno[i] != 'M' and sensores.terreno[i] != 'P'){
263 +                                 casillas_libres++;
264 +                         }
265 +                 }
266
    -                 if(sensores.superficie[2] == '_'  and sensores.terreno[2] != 'M' and sensores.terreno[2] != 'P'){
267 +                 if(casillas_libres > 0 and casillas_libres  <= 2){
268 +
269 +                         casillas_libres = 0;
270 +
271 +                         for(int j=9; j<16 and (casilla_encontrada == false); j++){
272 +
273 +                                 if((sensores.terreno[j] != 'M' and sensores.terreno[j] != 'P')){
274 +
275 +                                         casilla_encontrada = true;
276 +                                 }
277 +
278 +                                 if ((j == 9 or j == 10 or j == 11) and casilla_encontrada){
279 +                                         ir_hacia_zonaA = true;
280 +                                         casilla_encontrada = false;
281 +                                 }
282 +                                 else if ((j == 13 or j == 14 or j == 15) and casilla_encontrada){
283 +                                         ir_hacia_zonaB = true;
284 +                                         casilla_encontrada = false;
285 +                                 }
286 +                                 else if ((j == 12) and casilla_encontrada){
287 +                                         ir_recto = true;
288 +                                         casilla_encontrada = false;
289 +                                 }
290 +                         }
291 +                 }
292 +
293 +         }
294 +                 if(!ir_hacia_zonaA and !ir_hacia_zonaB and !ir_recto){
295 +                         accion = AlgoritmoBusquedaCoste(sensores);
296 +                 }
```

Ya por último también le hice unas mejoras para cuando se quedase encerrado dentro de algunas casas y pudiera salir, cree un algoritmo que detectase puertas a nivel lateral y no solo de frente, para así no quedarse haciendo bucles:

```
void ComportamientoJugador::AlgoritmoMuros(Sensores sensores){

	if(!ir_muro_derecha and !ir_muro_izquierda){

		if(((sensores.terreno[1] == 'M' or sensores.terreno[1] == 'P') and  (sensores.terreno[5] == 'M' or sensores
			ir_muro_izquierda = true;
			cout << "Muro izquierda encontrado" << endl;
		}
		else if (((sensores.terreno[3] == 'M' or sensores.terreno[3] == 'P') and  (sensores.terreno[7] == 'M' or se
			ir_muro_derecha = true;
			cout << "Muro derecha encontrado" << endl;
		}
	}

		if(!ir_muro_derecha and !ir_muro_izquierda){

			cout << "Aplicamos busqueda de coste: " << endl;

			AlgoritmoBusquedaCoste(sensores);
		}

		if(ir_muro_derecha){

			switch(ultimaAccion){

				case actFORWARD:

					if(cont_adelante == 3){
						accion = actTURN_R;
						cont_adelante = 0;
					}
					else {

						if(sensores.superficie[2] == '_'  and sensores.terreno[2] != 'M' and sensor

							accion = actFORWARD;
							cout << "Aplico default zona a, Avanzo" << endl;
							cont_adelante++;
						}
```

Por último, añadí una función que me permitía deducir cual son las casillas restantes según las casillas descubierta alrededor de su entorno:

```
void ComportamientoJugador::PintarFinalMapa(Sensores sensores){

    for(int i=0; i<mapaResultado.size(); i++){

        for(int j=0; j<mapaResultado.size(); j++){

            if(mapaResultado[i][j] == '?'){

                if((i < 3) or (j < 3) or ((mapaResultado.size()-i) < 3) o
                    mapaResultado[i][j] = 'P';
                }

                else {

                    switch(mapaResultado[i-1][j]){

                        case 'B':
                            cont_bosque++;
                        break;

                        case 'A':
                            cont_agua++;
                        break;

                        case 'S':
                            cont_piedra++;
                        break;

                        case 'T':
                            cont_arena++;
                        break;
```