INTELIGENCIA ARTIFICIAL

LOS EXTRAÑOS MUNDOS DE BELKAN

29 de abril de 2018

Laura Gómez Garrido
Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

1. PLANTEAMIENTO DEL PROBLEMA

Nos situamos en un mapa, cuya representación en una matriz tendrá de dimensión máxima 100x100 casillas. Nada más comenzar, apareceremos en una casilla aleatoria y nos darán una posición a la cual debemos dirigirnos, nuestro objetivo principal es llegar hasta allí.

Este sencillo planteamiento, tiene algunos inconvenientes para nosotros como que no podemos pasar por todas las casillas pues algunas nos bloquean el camino y otras pueden llevarnos a la muerte, deberemos de evitar tanto el colisionar con un obstáculo como morir. Además, a partir del nivel 2 dejaremos de estar solos en el mundo y aparecerán unos aldeanos que estorbarán a la hora de cumplir nuestra meta.

Además, a partir del nivel 3 no sólo dejaremos de conocer nuestra posición de origen sino que tampoco tendremos la información completa de cómo es el mundo en el que nos situamos. Deberemos valernos de nuestros ojos, que en este caso son un sensor de terreno y un sensor de superficie, para ir descubriendo cómo es nuestro mapa y cómo reaccionar hasta que lleguemos a una casilla PK que nos indicará, por fin, cuál es nuestra situación concreta. Pero no todo está perdido, ¡siempre sabemos que aparecemos mirando al norte!

En resumen, deberemos de implementar un comportamiento deliberativo y otro reactivo para llegar hasta nuestro destino. A continuación se asumirá un buen entendimiento del problema planteado y se procederá a explicar cuál era la idea al programar dichos comportamientos y cómo funcionan a grandes rasgos.

Hacemos aquí una mención especial al método *think* que recibirá unos sensores y dependiendo de la situación en la que nos encontremos decidirá actuar forma deliberativa o de forma reactiva. Se encargará principalmente de actualizar nuestra brújula y nuestra posición en el mapa, además de llamar a *actualizarMapa*, reaccionar a los aldeanos(¿Tengo un aldeano delante? ¡Pues me espero!) y de asegurarse que las decisiones tomadas por *reactivo* y *pathFinding* no nos llevarán a una muerte segura.

2. COMPORTAMIENTO DELIBERATIVO

En esta parte, era obligatorio la implementación de unos de los algoritmos vistos en clase. En esta solución del problema, se ha optado por utilizar A*. Procederemos a enunciar las clases y métodos utilizadas para implementación.

- Clase *celda*: Si bien podríamos haber utlizado simplemente el struct *estado* que era proporcionado en el código, se ha decidido implementar esta clase un poquito más potente por simple gusto y comodidad de la creadora del código pues, bajo su punto de vista, facilita la visualización del código y esquematización a la hora de pensarlo.

  Sus datos miembro son: celda * *padre*, list <celda>*adyacentes*, int *distanciaOrigen*, estado *pos*, int *dest_fil*a e int *dest_columna*. De la misma forma, además de los constructores, posee a *getAdyacentes, getPadre, getOrientacion, setOrientacion, menorOrigen, valorCelda, accesible* y definiciones de los operadores < y == como métodos públicos; como métodos privados tiene a *CalcAdy, CalcDistOrig* y a *CalcDistHeur*.

  Como se entiende que los nombres empleados son lo suficientemente explicativos por sí sólos, se indicaran exclusivamente los que se cree que puedan a llegar a ser un poco más confusos.

  - *menorOrigen* requiere como parámetro otra celda y comparará cuál de las dos tiene una menor distancia a nuestra posición inicial. Para ello, se ayudará de *CalcDistOrig* que calculará dichas distancias y las insertará en *distanciaOrigen* en función del valor de la misma variable en la celda padre que referenciamos con la variable del mismo nombre. Puede surgir la siguiente pregunta, ¿por qué se calcula en cada evaluación? La respuesta es sencilla, porque nuestro algoritmo puede hacer sustituciones de celdas y nos interesa tener siempre los valores lo más actualizados posible.
  - < este operador nos compara dos celdas y nos indica cuál de las dos tiene menor *valorCelda*. Es especialmente útil para poder hacer, en *pathFinding*, *abiertos.sort()* y valernos, así, de la stl cómodamente para ordenar nuestra lista de abiertos.
  - == nos compara dos celdas y nos indican si son iguales o no. ¿Cuándo son dos celdas iguales para nosotros? La respuesta es clara, cuando están situadas en la misma fila y columna, todos las demas variables son totalmente prescindibles para esta comparación.
  - *CalcAdy*, como su propio nombre indica, nos calcula las celdas adyacentes a la actual y se las asigna a nuestra variable *adyacentes*.

- Método *pathFinding*: Implementa el algoritmo A* para realizar la búsqueda del camino más adecuado. Para ello se valdrá de la clase *celda* y de los métodos de jugador *celdaCerrada* y *calcularPlan*. El primero mencionado nos devolverá un booleano indicando si la celda pertenece a la lista de cerrados, la cual le pasaremos por parámetro, y el segundo nos calculrá el plan una vez hemos llegado a la celda destino con nuestro algoritmo A*.

3. COMPORTAMIENTO REACTIVO

Primero de nada, aclarar que no es un comportamiento perfecto y aún posee muchas situaciones por depurar y arreglar. Después de todo, aquí intentamos implementar una búsqueda de casillas PK teniendo un conocimiento casi nulo del lugar en el que nos encontramos. Una respuesta pueder ser implementar en función de un factor aleatorio, en cuyo caso habrá situaciones en la que en un mismo mapa podamos encontrar el punto PK y otras en las que, bajo exactamente las mismas circunstancias, no.

Otra respuesta, la optada en este caso y ni mucho menos la respuesta ideal o perfecta, es implementar un comportamiento de acuerdo a un conjunto de situaciones predefinidas, de esta forma nos aseguramos que bajo las mismas condiciones siempre obtendremos el mismo resultado. Aquí surgen varios problemas, el código se vuelve engorroso, requiere un gran tiempo el simplificarlo, y nos podemos encontrar con infinitas situaciones cada cual más distinta de la anterior. De esta forma, dependiendo del mapa y la situación original, podemos encontrarnos con que en un mapa de tamaño 100 logramos alrededor de 20 objetivos y en otro del mismo tamaño ninguno a no ser que nos los encontremos por casualidad por el simple hecho de no haber sido capaces de encontrar la casilla PK.

Una vez aclarado esto, hacemos mención de los métodos implementados para desarrollar nuestro comportamiento reactivo.

- *girar*: La principal idea de nuestro programa se trata de realizar una búsqueda por columnas, aprovechando lo máximo posible nuestro sensor de terreno. Este método simplemente se encarga de devolver un plan con dichos giros y será llamado cuando corresponda o, al menos, sea posible.

- *miniBusqueda*: Este es uno de los métodos que más necesitan ser depurados. Básicamente, es llamado cuando nuestro sensor terreno encuentra una casilla PK y se encarga de, si es posible, organizar una ruta para llegar hasta allí. Hay algunos casos particulares analizados aquí, otros no son analizados porque pueden llegar a serlo al avanzar hacia adelante y, finalmente, hay otros que no son analizados y no son alcanzables directamente. Especialmente por estos últimos es que se ha mencionado que este método necesita ser más depurado. También requiere aumentar la legibilidad para facilitar su compresión.

- *reactivo*: El encargado principal de analizar nuestras próximas acciones para la búsqueda de nuestra casilla PK. Básicamente, comprueba si nuestros sensores han encontrado dicha casilla y también comprueba si podemos avanzar o tenemos una serie de movimientos bloqueados. Dependiendo de estas respuestas, programará un giro con *girar*, llamará a *miniBusqueda* o implementará directamente otra serie de movimientos. Es el método más importante y también requiere de una serie de depuraciones para analizar más casos aún y tomar mejores decisiones, esto puede con llevar en la aparición de otra serie de métodos que faciliten el trabajo y la legibilidiad.

Como la extensión máxima de esta memoria son 5 páginas, no mencionaremos los casos concretos se contemplan tanto en reactivo como en miniBusqueda. No son sencillos de explicar estos y la motivación tras cada uno de ellos y como, en caso de haber futuras versiones, muy probablmente estos sufran de modificaciones no se cree necesaria una explicación sobre cada uno de ellos.