Inteligencia Artificial

Curso 2018-19

Práctica 2. Agentes Reactivos/Deliberativos: los extraños mundos de BelKan

Autor[1]: Víctor García Carrera, victorgarcia@correo.ugr.es

Síntesis.

En esta memoria se recoge un breve resumen del trabajo realizado y el proceso seguido para implementar los niveles 1 y 2 de la práctica 2 de Inteligencia Artificial de agentes reactivos/deliberativos. Esto comprende la implementación en C++ de diversos algoritmos de búsqueda (sin información) como las búsquedas en profundidad, anchura y coste uniforme (nivel 1), junto con una implementación propia de un agente reactivo/deliberativo para el nivel 2 capaz de reajustar de forma efectiva su plan ante cambios imprevistos, evitando obstaculos.

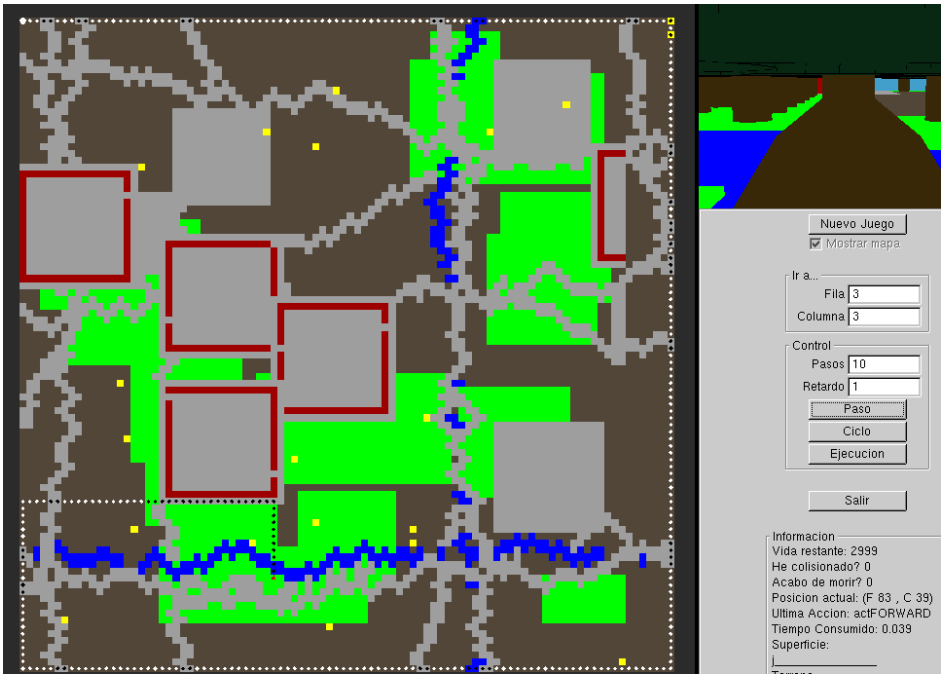
Nivel 1. Busqueda en anchura

Disponemos de una implementación de la búsqueda en profundidad al inicio de la práctica. Con la información vista en clase y junto con nueva documentación obtenida tras una búsqueda en internet del algoritmo de búsqueda en anchura, disponemos de los recursos necesarios para implementar esta búsqueda. En primer lugar, este método de búsqueda sin información se asemeja tremendamente a la búsqueda en profundidad, pues ambos emplean el mismo algoritmo con la diferencia en la estructura de datos que emplean para almacenar los nodos(estados) a explorar. La búsqueda en profundidad emplea una Pila LIFO, mientras que la búsqueda en anchura emplea una Cola FIFO. Esto ocasiona que la búsqueda en profundidad, siempre que encuentre un nuevo nodo a explorar, pase a explorarlo, profundizando en el arbol de búsqueda siempre que pueda. La búsqueda en anchura, por el contrario, analiza todos los nodos de un nivel antes de pasar a analizar los de un nivel de profundidad más. No queda atrapada en callejones sin salida, a diferencia de la búsqueda en profundidad, y siempre encuentra el camino (si existe) más corto. Implementamos el algoritmo basándonos en la descripción de las diapositivas, y los resultados son notables. A la hora de poner a prueba esta búsqueda, comprobamos que efectivamente encuentra el camino, y además el más corto, a diferencia de la búsqueda en profundidad, aunque esta última resulta ser más rápida (notablemente) en algunos casos, en aquellos donde el estado destino se encuentra muy profundo en el árbol de búsqueda y al explorar profundiza muy rápido en una rama que lleva directamente a éste, mientras que la búsqueda en anchura, al explorar todos los niveles progresivamente, tarda mucho hasta llegar a ese nivel. Un ejemplo de este caso es el del MAPA 100. Se puede apreciar que la búsqueda en profundidad apenas tarda 1 segundo, mientras que la de en anchura tarda 126 segundos (el objetivo está en una esquina del mapa, lo más distante y por ende profundo en el árbol de búsqueda).

Ejemplo PROFUNDIDAD gana ANCHURA (en tiempo de cálculo del plan)

---

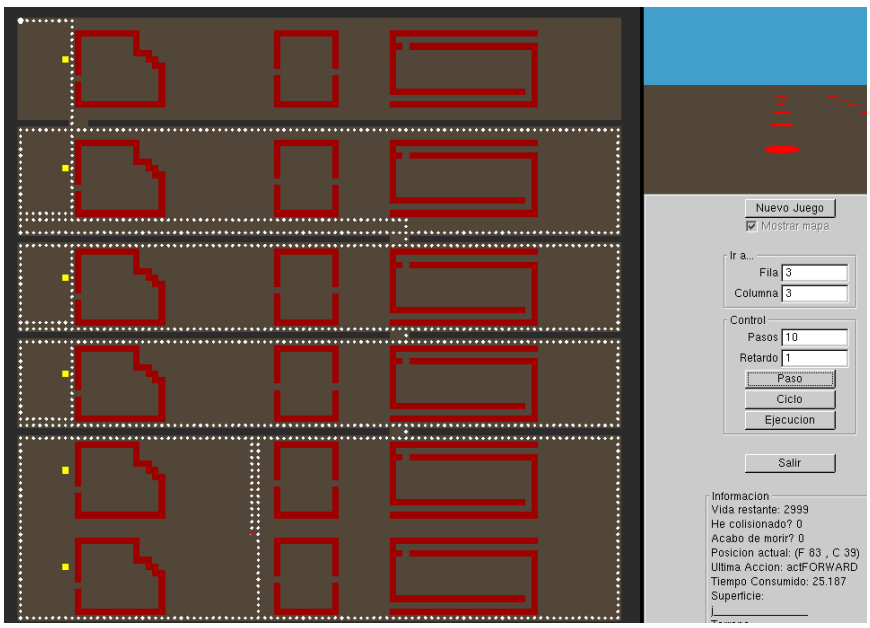
1 Como autor declaro que los contenidos del presente documento son originales y elaborados por mi. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la "Normativa de evaluación y de calificaciones de los estudiantes de la Universidad de Granada" esto "conllevará la calificación numérica de cero … independientemente del resto de calificaciones que el estudiante hubiera obtenido ..."



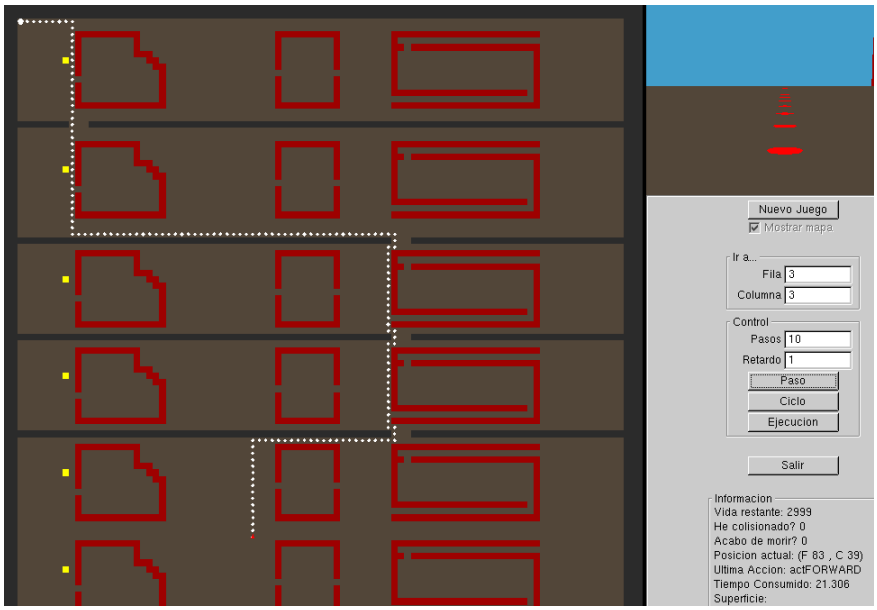
Profundidad

Anchura

Ejemplo "estandar" de PROFUNDIDAD vs ANCHURA



Profundidad



Anchura (mejor tiempo)

Resulta destacable la visualización de los planes que obtienen los 2 algoritmos pues muestran las semejanzas y diferencias entre ambos. La búsqueda en profundidad tiende a analizar caminos por el borde del mapa, pues este algoritmo, mientras encuentre un nuevo nodo a explorar, pasa a explorarlo, coge un camino (una acción modelo) y profundiza todo lo que puede con esa acción, hasta que se encuentra un obstáculo o no genera nuevos nodos. Es un algoritmo rígido, que siempre que puede avanza hacia delante. Esto es debido a que se implementa con una pila LIFO y siempre pasa a analizar si puede la última acción añadida, que es avanzar hacia delante (si no hay un obstáculo). El problema del algoritmo es que, sea cual sea la acción que pongamos al final para que valore a analizar (podría haber sido girar a la derecha), siempre tenderá a realizarla, recorre el mapa de fuera hacia dentro, todo recto hasta chocarse con algún obstáculo, girar a uno de los lados (primero intenta izquierda) y volver a avanzar recto. Esto ocasiona que acabe chocando con un borde del mapa y continue bordeandolo. *Útil para objetivos en los extremos del mapa, Inútil para el resto de objetivos.*

La búsqueda en anchura obtiene siempre que existe el camino más "recto", con el menor número de acciones hasta el objetivo. No tiene en cuenta el coste de avanzar en el mapa. Para lograr este camino con menor número de acciones, analiza todos los nodos(estados) de un nivel antes de pasar al siguente. Esto se traduce en que analiza todas las casillas a una acción, luego a 2, a 3... hasta llegar al destino tras *n* acciones. Eso implica que el camino que obtiene al destino es el menos profundo, el de menor número de nodos (acciones) hasta él. Sin embargo, el tiempo para obtener el camino aumenta exponencialmente con cada nivel de profundidad, recorre el mapa de dentro a fuera, con cuadrados centrados en el origen de lados 2, 3... hasta el cuadrado donde esté el objetivo. *Muy Útil para objetivos cercanos al jugador (origen), Útil para distancias medias y pierde rendimiento y eficacia para distancias muy lejanas.*

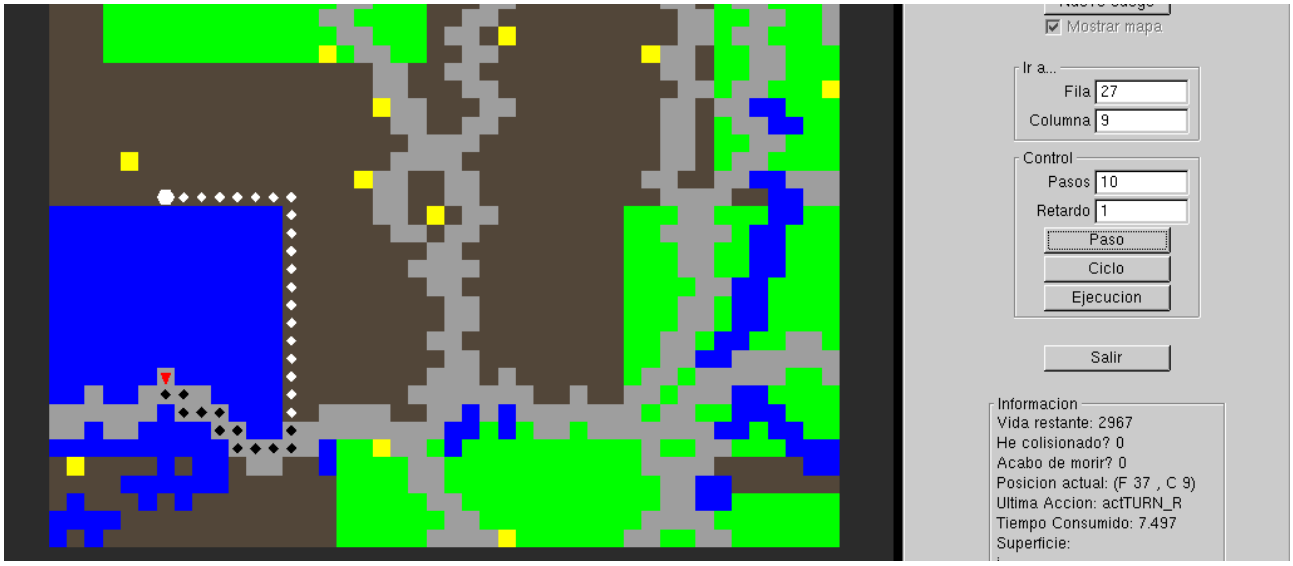
En resumen, la búsqueda en anchura nos asegura encontrar el plan más corto (con menos acciones) hasta el destino, con un gran rendimiento para objetivos no muy lejanos. Aun siendo un objetivo lejano, ofrece un rendimiento no menospreciable. La búsqueda en profundidad resulta muy efectiva para objetivos cercanos a los extremos del mapa, pero en el resto es preferible la búsqueda en anchura.

Nivel 1. Busqueda de costo uniforme

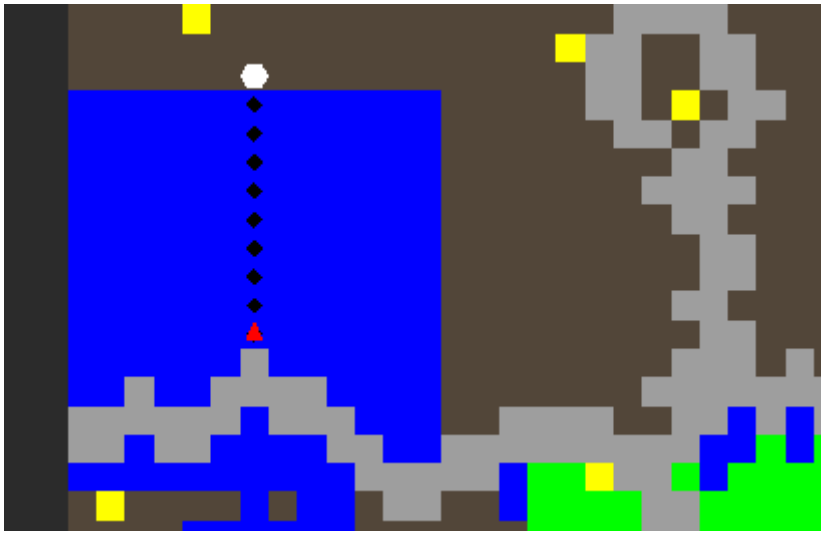
La principal diferencia de esta búsqueda con las anteriores es que permite buscar un camino teniendo en cuenta el tipo de terreno del mapa, que repercute en encontrar el camino más rápido al objetivo. La búsqueda en anchura nos aseguraba encontrar si existía el camino, aquel más corto EN TÉRMINOS DE NÚMERO DE ACCIONES. La búsqueda de costo uniforme encuentra el camino más corto EN TÉRMINOS DE PASOS (TIEMPO) hasta el objetivo, es decir, el más rápido para el jugador. Junto con la información vista en clase y la descripción del algoritmo en las diapositivas, implementamos esta búsqueda. La estructura de nodo en el árbol de búsqueda cuenta con el coste asociado hasta llegar a ese estado(casilla) desde el origen, teniendo en cuenta que el coste de todas las acciones es 1u excepto en el caso de avanzar por los siguientes terrenos especiales: Tierra (2u), Bosque (5u) y Agua (10u). Este algoritmo explora aquellos nodos con menor coste, lo cual garantiza, al igual que con el algoritmo de Dijkstra, que la solución obtenida refleja el camino de menor coste, el más rápido, al destino. Esto se aproxima bastante al comportamiento esperado del jugador, pues encuentra el camino más rápido al objetivo y así puede alcanzar el mayor número posible de objetivos. Su único inconveniente es el tiempo que emplea en buscar el camino más rápido, que puede quedarse estancado si el coste de todos los nodos a explorar es semejante o el mismo (distancias muy lejanas donde el terreno a explorar es uniforme).

Con algunas modificaciones podemos adaptarlo para la parte deliberativa del nivel 2. Este algoritmo es un caso particular del algoritmo de búsqueda A* si la heurística de éste fuera una función constante.

Ejemplo diferencia UNIFORME vs ANCHURA



Uniforme (tiene en cuenta el coste de avanzar por agua y rectifica por el borde)



Anchura (solo busca el camino más directo, se lanza a ahogarse al agua)

Ejemplo PROFUNDIDAD bordeando el río y con el camino más óptimo, pero 68,91s para elaborar el plan

Nivel 2. Agente reactivo/deliberativo

En este nivel nos enfrentamos al reto, donde no conocemos el mapa. Podemos dividir la actuación del agente en 2 partes: Buscar PK (punto de referencia) y encontrar un camino hasta el objetivo.

Nota: Al hablar de obstáculo nos referimos tanto a obstáculo del terreno como un aldeano.

Buscar PK: En esta parte implementamos un comportamiento reactivo donde mientras no tenga obstáculos delante avanza, y si tiene un osbtáculo gira a la derecha. Cuando tiene un PK en su sensor de visión del terreno, elabora un mini plan para llegar hasta él. Si en el transcurso de ese mini plan encuentra un obstáculo, gira a la derecha y prosigue con su actitud reactiva hasta encontrar de nuevo un PK en su sensor de visión del terreno. Si no, llega hasta PK y actualiza su posición en el mapa. Podríamos mejorarlo para aprovechar todo lo que el agente ve hasta que llega al PK y ya entonces actualizar correctamente el mapa, pero considero que no suele recorrer mucho terreno hasta encontrar el PK y aun así, una vez encontrado este primer PK, la función para ir actualizando el mapa a medida que se dirige al objetivo aprovecha TODO el campo de visión del agente, lo cual resulta muy efectivo. Lo que ganamos en terreno descubierto hasta encontrar el PK no proporciona gran ayuda frente a la búsqueda y actualización del terreno posterior. También podríamos mejorar la búsqueda del PK eligiendo, en vez de seguir de forma predeterminada recto, el siguiente paso de menor coste según la info del sensor del terreno.

Llegar al objetivo: Aquí utilizamos la búsqueda de costo Uniforme para encontrar la ruta más rápida hasta el objetivo. Está implementada de forma que da coste 1 a aquellos nodos desconocidos, de manera que puede elaborar el plan con partes del mapa desconocidas, incluso tenderá a explorarlas al tener un coste tan básico. Esto, junto con la función implementada que aprovecha, a medida que se dirige al destino, TODO el campo de visión del terreno del agente, resulta en que rápidamente conoce todo el mapa y puede implementar con aun más éxito esta búsqueda. Puede realizar un plan que no es el mejor o con obstáculos si desconoce parte del mapa, pero puede ajustarse ante cambios imprevistos comprobando antes de realizar la acción del plan si ésta supone colisionar con un obstáculo, borrandose el plan y girando a la derecha (por arbitrariedad) para acto seguido volver a calcular un plan. Al hacerlo, ahora conoce los nuevos obstáculos encontrados y realiza un plan mejor. Mejoramos esta implementación haciendo que, si tarda mucho la búsqueda Uniforme, realice una búsqueda en Anchura, no tan óptima pero bastante eficaz. Establecemos el tiempo límite en 10 segundos.

Anexo.

El código se encuentra en jugador.cpp y jugador.hpp