Búsqueda Heurística
Tarea 2: Akari Robot

Matías Bustos, Sasha Bravo

Descripción del juego

Akari Robot es una modificación al juego Akari, el cual consiste en colocar luces en un tablero que tiene casillas blancas y negras, las luces se ubican en las casillas blancas e iluminan en las 4 direcciones hasta encontrarse con el límite o una casilla negra. Para ganar es necesario iluminar todas las casillas blancas. Las luces en el juego deben seguir ciertas reglas para poder completar el tablero; una de ellas es que no puede haber 2 luces en la misma fila o columna con tal de que se iluminen entre sí.
La modificación consiste en que, ahora existe un robot que comienza en alguna de las casillas blancas del tablero, y este es una "ficha" que deberá moverse casilla por casilla y poner las luces. Además de conseguir completar el tablero, ahora también se debe completar en la menor cantidad de movimientos posibles.

Robot: Ficha que representa la posición actual del jugador. Coloca luces en la posición que se encuentra y puede moverse en las 4 direcciones (a menos que haya una casilla negra) una casilla a la vez.

Luces: Fichas que colocará el robot en el tablero. Iluminan toda la fila y columna en la que se encuentra (hasta llegar a una casilla negra). No pueden haber 2 luces en la misma fila ni columna.

Muros: Son las casillas negras sin número, estas casillas actúan como muros e impiden el movimiento del robot y la iluminación de las luces.

Muros Numerados: A diferencia de los muros normales, estas casillas tienen números que pueden ser de 0 a 4. Estos números indican la cantidad de luces que deben haber en las casillas adyacentes al muro (sin haber ni más ni menos). Es una de las condiciones que se deben cumplir para completar el juego.

Implementación

struct GameState: Estructura base de cada estado, donde se almacenan todas las variables correspondientes al tablero como: Robot, luces, casillas encendidas, etc. Además de contar con los “operator” correspondientes para utilizar en diversas comparaciones.

BFS: El código implementa un algoritmo BFS para encontrar un camino desde un estado inicial hasta un estado objetivo en el juego. Utiliza una cola para almacenar los estados a visitar, un conjunto para almacenar los estados visitados y un mapa para almacenar el estado padre de cada estado visitado. Se recorren los sucesores de cada estado actual, se agregan a la cola si no han sido visitados y se actualiza el estado padre. Si se encuentra un camino al estado objetivo, se reconstruye el camino desde el objetivo hasta el estado inicial.

Heuristic: Calcula la heurística para A*. Utiliza la distancia euclidiana entre la posición del Robot y el punto de destino más cercano como estimación del costo restante. También considera los waypoints no visitados y los obstáculos para ajustar la estimación. El resultado es una medida de la "proximidad" al objetivo, que se utiliza para guiar la búsqueda hacia soluciones más prometedoras.

A*: Utiliza una cola de prioridad para mantener los estados abiertos ordenados por su costo estimado, un conjunto para almacenar los estados visitados y un mapa para almacenar los estados padres. El algoritmo se ejecuta en un bucle mientras la cola de prioridad no esté vacía. En cada iteración, se selecciona el estado con el menor costo estimado. Se verifican condiciones de objetivo y se inserta el estado en el conjunto de estados visitados. Se generan los sucesores del estado actual y se actualiza el costo y el estado padre de cada sucesor si es necesario (Los sucesores se agregan a la cola de prioridad).

Search: Realiza una búsqueda recursiva para encontrar una solución en un juego. Calcula el valor heurístico basado en el costo acumulado y la función heurística, y realiza cortes si el valor supera el límite establecido. Verifica si se ha encontrado la solución o si el estado es una posición de luz. Genera los sucesores y realiza la búsqueda recursiva en cada uno, evitando estados duplicados. Actualiza el mínimo con el resultado de la búsqueda y retorna el mínimo si no se encuentra la solución.

IDA*: Calcula un límite inicial basado en la heurística y el estado inicial, y realiza iteraciones utilizando la función *Search* para explorar los sucesores y encontrar una solución. Actualiza el límite en cada iteración y retorna verdadero si se encuentra la solución, o falso si no se encuentra.

Resultados

	BFS			A*			IDA*		
LVL	Nodos Visitados	Tiempo (s)	Nodos /s	Nodos Visitados	Tiempo (s)	Nodos /s	Nodos Visitados	Tiempo (s)	Nodos /s
1	9775	0.01296	754243	470	0.00465	101075	/	/	/
2	2987728	8.36997	356958	1625	0.39482	4115	/	/	/
3	358847	1.31688	272497	205	0.08391	2443	/	/	/
4	704657	2.49346	282602	500	0.08343	5993	/	/	/
5	728912	2.68434	271542	545	0.16678	3267	/	/	/

Conclusiones

No logramos reconstruir el camino en el bfs, dado que la asignación del padre en el ciclo de reconstrucción no se asignaba, detalle que no logramos descubrir el porqué.

En el caso del IDA* el algoritmo mientras busca el camino llega a un punto donde se queda iterando hasta un cierto largo de movimiento y vuelve a empezar, por lo que pensamos que en algún punto deja de reconocer los vecinos del nodo actual, dado esto, quedó el algoritmo en el código pero sin ejecutarlo por el problema mencionado.

En el caso de Akari Robot, hemos implementado tanto el algoritmo de BFS como el algoritmo A* y al comparar los algoritmos, podemos concluir:

El algoritmo A* explorar menos nodos que el BFS debido a su estrategia heurística que guía la búsqueda hacia áreas más cercanas a la meta.

El algoritmo A* cuenta con el costo acumulado hasta el momento como una estimación del costo restante, lo que evita recorrer caminos costosos. El BFS no considera el costo y explora en todas las direcciones posibles.

También podemos tomar en cuenta el tiempo de ejecución de A* es relativamente menor que BFS, lo que en general permite realizar más pruebas con distintos estados iniciales y comprobar la efectividad del algoritmo y al reducir el espacio de búsqueda dependiendo de los costos asignado se presenta un mejor camino en un tiempo de búsqueda más razonable, sin embargo aún se tienen problemas con la función heurística y no es completamente razonable al dar el valor del estado actual por lo que se podría mejorar de alguna manera el rendimiento del algoritmo A* para mejores resultados.