NOMBRE: Manuel Alejandro Muñoz Carrasco

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 — Inteligencia Artificial — 2' 2022

Tarea 2

1.1 Para el desarrollo del algoritmo *Anytime* me base mucho en lo que es *A**, por lo que es bastante similares, además dejé en el código comentada cada línea sobre lo que hice y el porque.

Para correr el código se debe hacer lo siguiente:

```
python rushhour/run.py anytime <board> <heuristic> <expansions> <weight>
```

Además, el programa en caso de encontrar una solución te preguntará si deseas quedarte con ella o buscar otra. En caso de que encuentre una y después no encuentre más, imprimirá que no se encontraron más soluciones y el programa terminará.

1.2 En primer lugar es importante notar que es lo que hace la heurística BlockingCarsHeuristic. Esta heurística se encarga en revisar si existen vehículos con orientación vertical que podrían bloquear el paso del auto rojo, para lo cual revisa los vehículos que estánn por delante de la parte delante del auto rojo y que además se encuentran en por arriba de este, tales que puedan o no estar bloqueando el paso actualmente de este.

Tomando en consideración esto, podemos notar que la heurística solo está considerando los vehículos por sobre el y que esten en una dirección vertical.

1.3 En la siguiente tabla se puede apreciar la diferencia empírica existente entre ambas heurísticas utilizando el mismo algoritmo, A*

	Blockingcars		Zero	
Test	Time	Expansions	Time	Expansions
Test 1	0,010445	4	0,008924	4
Test 2	0,459302	1038	0,465453	1068
Test 3	0,298744	836	0,298622	850
Test 4	0,103775	258	0,105579	260
Test 5	1,857205	3812	1,92604	3905
Test 6	0,257345	591	0,66889	615
Test 7	1,84943	2883	1,94401	3069

Como se logra apreciar, la heurística BlockingCars siempre expande o menos o la misa cantidad que la heurística Zero, además de tardar en la mayoría de casos una menor cantidad de tiempo. Todas estas comparaciones se hicieron utilizando un peso de $1,5$. De esta forma podemos concluir que si nos interesa que la heurística expanda lo menos posible, debemos priorizar utilizar BlockingCars.

1.4

1.5 Desarrollo:

a) *Bidirectional Search* corresponde a un algoritmo de búsqueda que se enfoca en buscar el camino más corto que existe entre dos nodos de un grafo. La particularidad es como lleva a cabo este proceso. Lo que hace es ir desde el nodo objetivo al nodo de inicio y también para ir del nodo de inicio al objetivo. Esto lo hace con la finalidad de realizar una menor cantidad de expansiones, ya que en el momento en que ambas ejecuciones (de ahí el nombre Bidirectional) se encuentran, el camino ya fue descubierto, y los nodos explorados son mucho menores a los necesarios en caso de utilizar otro algoritmo de búsqueda. Por otra parte, este algoritmo también permite utilizar heurísticas, al igual que A*.

b) En las líneas 33 y 34 se logra apreciar la implementación de la Bidirectional Search, al estar guardando en generados el nodo objetivo y el nodo del que comienzo, y de esta forma hacer una búsqueda más eficiente del camino a seguir para comunicar los dos estados.

1.6