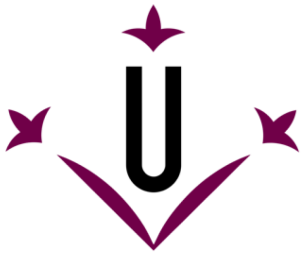

Universitat de Lleida

Pac-Man

Segura Paz, Aleix

Serrano Ortega, Aniol

Data: 06/11/2023

Pràctica 1

Intel·ligència Artificial

Escola Politècnica Superior

Índex

Índex de Taules

1. Introducció

En aquest informe, es detallen diferents algoritmes de cerca implementats per al joc del *Pac-Man*. A través de la implementació d'aquests algorismes (DFS, BFS, UCS i A*) es tracta de comprendre el seu funcionament i poder comparar-los entre si. D'altra banda, es proporciona una descripció i comparativa de rendiment de les heurístiques desenvolupades per a estimar el cost restant per arribar a l'objectiu.

2. Implementació Algorismes

2.1. *Depth-First Search* (DFS)

Per a la implementació del DFS s'han considerat els següents punts de disseny:

- S'utilitza una estructura LIFO (*Last In First Out*) com a *fringe*, això permet explorar en profunditat, ja que sempre s'explora l'últim node introduït en l'estructura quan s'ha acabat d'expandir un.
- Abans d'entrar al bucle es comproVa si el node inicial és ja solució.
- Quan obtenim els successors del node que estem explorant, es comprova que no estigui ni a *expanded* ni a *fringe*.
- Es comprova si un node és solució en el moment que el veiem per primer cop, és a dir, quan el crea el node pare.

2.2. *Breadth-First Search* (BFS)

Per a la implementació del BFS s'han considerat els següents punts de disseny:

- S'utilitza una estructura FIFO (*First In First Out*) com a *fringe*, això permet explorar en amplada ja es va explorant per nivells i no s'explora un node del nivell següent fins que s'ha explorar el actual.
- Abans d'entrar al bucle es comprova si el node inicial és ja solució.

- Quan obtenim els successors del node que estem explorant, es comprova que no estigui ni a *expanded* ni a *fringe*.
- Es comprova si un node és solució en el moment que el veiem per primer cop, és a dir, quan el crea el node pare.

2.3. *Uniform Cost Search* (UCS)

Per a la implementació del UCS s'han considerat els següents punts de disseny:

- S'utilitza una estructura *Priority Queue* com a *fringe*, ja que permet que sempre explorem el node amb millor cost acumulat.
- Es comprova si un node és solució just al fer *fringe.pop()* per assegurar que sigui la solució òptima (millor cost).
- Es comprova que el node a explorar no estigui en *expanded*.
- Es comprova que els nodes fill no estiguin en *expanded* quan es creen, si no ho estan s'introdueixen a *fringe*.
- La funció *update* de *fringe* s'encarrega de reemplaçar el mateix node si està a *fringe* i ara el visitem amb un cost menor.

2.4. *A-Star*

Per a la implementació del A-Star s'han considerat els següents punts de disseny:

- S'utilitza una estructura *Priority Queue* com a *fringe*, ja que permet que sempre explorem el node amb millor cost acumulat (cost del node + cost de l'heurística per a aquell node).
- Es comprova si un node és solució just al fer *fringe.pop()* per assegurar que sigui la solució òptima (millor cost).
- Es comprova que el node a explorar no estigui en *expanded*.

- Es comprova que els nodes fill no estiguin en *expanded* quan es creen, si no ho estan s'introdueixen a *fringe*.
- La funció *update* de *fringe* s'encarrega de reemplaçar el mateix node si està a *fringe* i ara el visitem amb un cost menor.

3. *Corners Problem*

En el cas de l'heurística per a les cantonades (*cornersHeuristic*), l'objectiu és proporcionar una estimació del cost restant per a arribar a la meta del problema. Més específicament, la meta és visitar totes les cantonades del mapa i retornar l'estimació d'aquest cost.

En primer lloc, s'han d'identificar les cantonades no visitades fent ús del segon paràmetre de l'estat. En cas que una cantonada no estigui visitada dins de la llista de *corners*, aquesta cantonada es guarda en una llista de les cantonades no visitades. En el cas que no existeixin cantonades no visitades, llavors totes les cantonades han estat visitades i l'heurística retorna 0.

Seguidament, es calculen les distàncies des de la posició actual a totes les cantonades no visitades emprat la funció de la distància *Manhattan* definida en l'arxiu `util.py`. D'aquesta manera, es troba la cantonada més propera i la seva distància.

Finalment, des de la cantonada més propera es calculen les distàncies cap a les altres cantonades no visitades, les quals es van acumulant en la variable `corners_heuristic` fins que es retorna aquest valor. Aquesta heurística es pot provar amb la següent comanda:

```
$ python3 pacman.py -l <layout> -p AStarCornersAgent -z 0.5
```

4. *Food Search Problem*

L'objectiu de l'heurística de *foodHeursitic* és estimar la distància des de la posició actual fins a tots els punts de menjar restants en el mapa. Primerament, es revisa si tot el menjar ha estat ja menjat, és a dir, no queda menjar i, per tant, es retorna 0. Seguidament, es calcula la distància *Manhattan* des de la posició actual fins a cada punt de i es guarda la menor distància.

A continuació, mentre la llista de menjar no sigui buida, es troba el menjar més proper i es calcula la distància acumulada des d'aquest menjar fins als menjars restants. Per a cada cantonada, aquesta s'elimina de la llista. Finalment, se sumen les distàncies en la variable *food_heuristic* per a cada iteració fins que es retorna com a solució. Per a provar aquesta heurística es pot fer mitjançant la següent comanda:

```
$ python3 pacman.py -l <layout> -p AStarFoodSearchAgent
```

5. Anàlisis *cornersHeuristic*

Per la heurística de les cantonades s'ha provat amb els mapes de *tinyCorners*, *mediumCorners* i *bigCorners*. Per a UCS (heurística trivial) i A* (*cornersHeuristic*). Per tant, s'ha elaborat la següent taula:

Algorismes	**Mapes**	**Cost**	**Nodes *expanded***
UCS	*tinyCorners*	28	252
	mediumCorners	106	1966
	bigCorners	162	7949
A*	*tinyCorners*	28	154
	mediumCorners	106	692
	bigCorners	162	1725

Taula 1: Comparativa de costos y nodes *expanded* entre UCS i A*

A partir d'aquesta taula, es pot veure que amb el *AStarCornersAgent* tant el cost com els nodes expandits han estat considerablement inferiors als de l'heurística trivial.

6. Anàlisis *foodHeuristic*

Per la heurística del menjar s'ha provat amb els mapes que no tenen cap fantasma i hi ha més d'un menjar. Per a mostrar el temps d'execució s'ha modificat una línia de `searchAgents.py` (*registerInitialState*) per a tenir més precisió del temps (`%.1f -> %.3f`).

A partir de l'execució de l'algorisme emprant les dues heurístiques, s'ha elaborat la següent taula:

Mapes	**Temps** (s)		**Nodes *expanded***	
	UCS	A*	UCS	A*
testSearch	0.002	0.0008	14	12
tinySearch	0.161	0.088	5057	2152
smallSearch	4.801	0.276	70726	3416
trickySearch	0.983	0.805	16688	9912

Taula 2: Comparativa de temps i nodes *expanded* entre UCS i A*

7. Conclusions

Per a realitzar la implementació dels algorismes primerament descrits, s'ha hagut de superar una sèrie d'obstacles, però el principal ha estat comprendre el context del joc i quines són eines de les quals es disposen. Amb l'ajuda del pseudocodi de les transparències s'ha pogut dur a terme una estimació de la solució per a cada algorisme, i a força de realitzar iteracions sobre el problema s'ha aconseguit implementar aquests algorismes.