

# MEMORIA PRÁCTICA 1

**Título: Búsqueda**

**Autores: Gregorio Blázquez Martínez y Diego Vicente Miguel**

## Sección 1 (1 punto)

### 1.1. Comentario personal en el enfoque y decisiones de la solución propuesta (0.5 puntos)

Para solucionar este problema de búsqueda hemos usado los conocimientos aprendidos en la teoría de la asignatura, donde se nos ha proporcionado un pseudocódigo de la búsqueda en grafo (con eliminación de estados repetidos). En este caso teníamos que implementar una búsqueda en profundidad; hemos empleado una pila (Stack) como lista de abiertos de forma que los nodos se vayan insertando y explorando los sucesores en orden y en cadena. Hemos implementado una función general (`generalSearch()`), apta y válida para ejercicios y algoritmos de búsqueda empleados en ejercicios posteriores. Por otro lado, para ir guardando la solución (camino correcto hasta llegar al nodo meta) hemos guardado en una lista los tripletes formados por el nodo, su lista de sus acciones y el coste hasta llegar a él.

#### 1.1.1. Lista & explicación de las funciones del framework usadas

Para la *Pila* hemos tenido que usar las funciones que se nos proporcionan *push*, *pop* e *isEmpty* para operar con la pila según el algoritmo.

Para trabajar con el problema/nodos hemos tenido que usar las tres funciones que se nos proporcionan *getStartState*, *getSuccessors* e *isGoalState*.

#### 1.1.2. Incluye el código añadido

```
# Implementación del algoritmo de búsqueda en grado sin estados repetidos mostrado en teoría
def generalSearch(problem, listaAbiertos):
    listaAbiertos.push((problem.getStartState(),[],0))
    listaCerrados=[]

    while(True):
        if listaAbiertos.isEmpty(): return []

        nodo=listaAbiertos.pop()
        if(problem.isGoalState(nodo[0])):
            return nodo[1]

        # Mantenemos una lista de los nodos ya explorados para no repetirlos
        if nodo[0] not in listaCerrados:
            listaCerrados.append(nodo[0])
            # Añadimos los sucesores para explorarlos en el orden dado por la funcion getSuccessors
            for successor in problem.getSuccessors(nodo[0]):
```

```

lista=nodo[1].copy()
lista.append(successor[1])
listaAbiertos.push((successor[0],lista,nodo[2]+successor[2]))

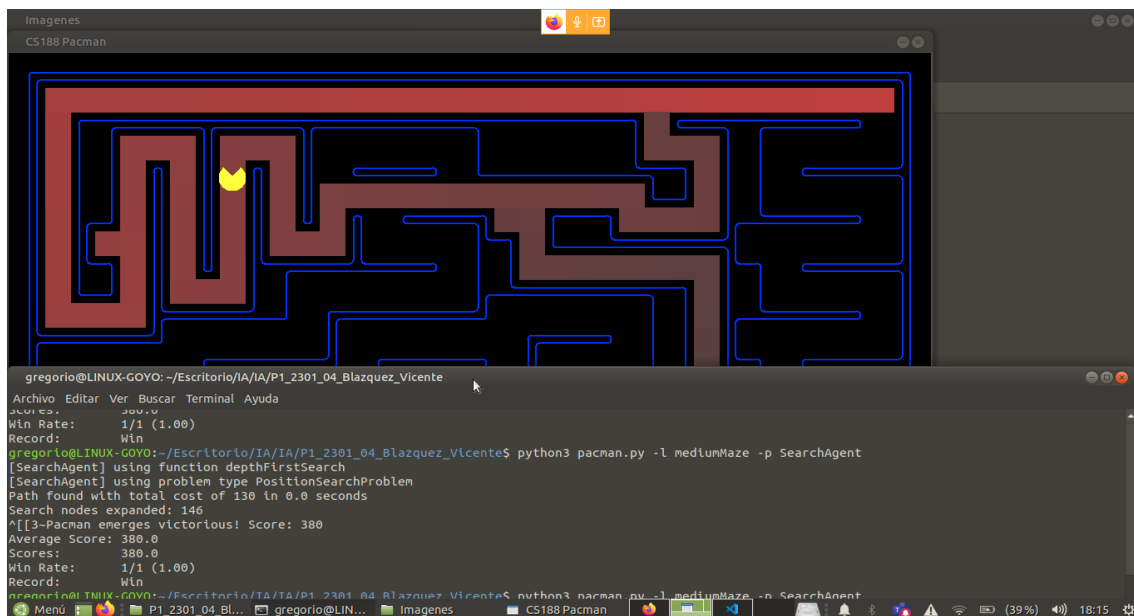
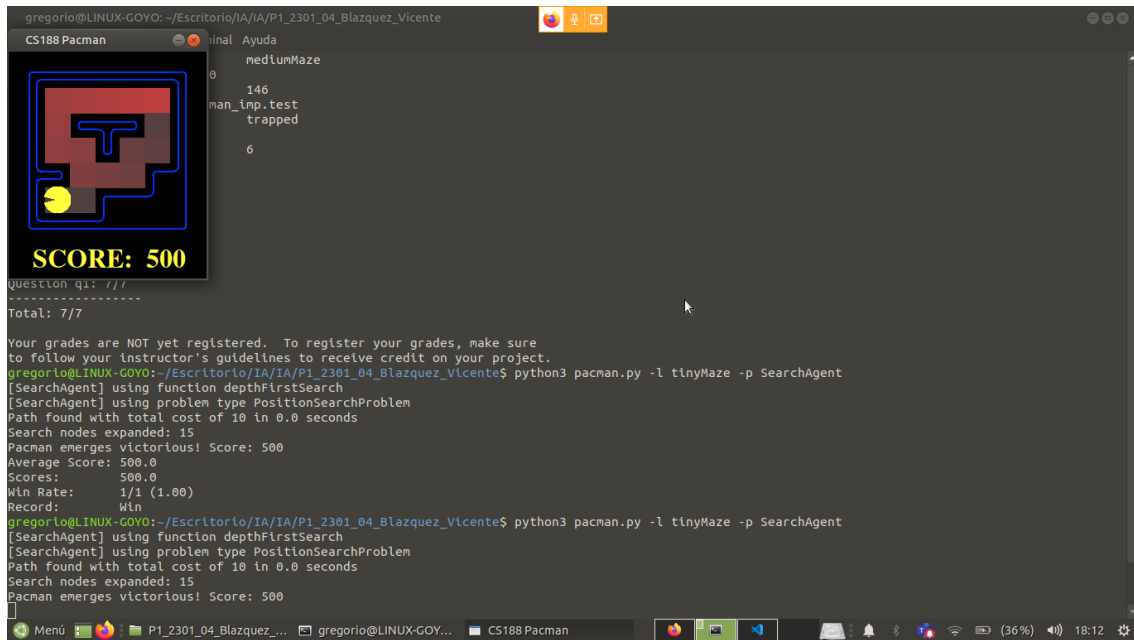
```

```

# Inicializamos la lista de abiertos como una cola LIFO para la búsqueda en profundidad
listaAbiertos=util.Stack()
return generalSearch(problem,listaAbiertos)

```

### 1.1.3. Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados



```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 p
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
^[[3-Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores: 380.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 p
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores: 380.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 p
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores: 380.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l bigMaze -z .5 -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
```



```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 autograder.py -q q1
Starting on 2-12 at 18:28:23

Question q1
=====
*** PASS: test_cases/q1/graph_backtrack.test
*** solution: ['A->C', 'B->C']
*** expanded_states: ['A', 'D', 'C']
*** PASS: test_cases/q1/graph_bfs_vs_dfs.test
*** solution: ['2:A->D', '0:D->G']
*** expanded_states: ['A', 'D']
*** PASS: test_cases/q1/graph_imp.test
*** solution: []
*** expanded_states: ['A', 'C', 'B']
*** PASS: test_cases/q1/graph_infinite.test
*** solution: ['0:A->B', '1:B->C', '1:C->G']
*** expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q1/graph_manypaths.test
*** solution: ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
*** expanded_states: ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases/q1/pacman_1.test
*** pacman_layout: mediumMaze
*** solution length: 130
*** nodes expanded: 146
*** PASS: test_cases/q1/pacman_imp.test
*** pacman_layout: trapped
*** solution length: 0
*** nodes expanded: 6

## Question q1: 7/7 ##

Finished at 18:28:23

Provisional grades
=====
Question q1: 7/7
-----
Total: 7/7

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$
```

Como podemos observar en las capturas de pantalla proporcionadas, el número de nodos expandidos en general no es muy elevado. Sin embargo, el coste del camino encontrado es claramente superior al camino óptimo (encontrado en búsqueda en profundidad).

## 1.2. Conclusiones en el comportamiento de Pacman, es óptimo (s/n), llega a la solución (s/n), nodos que expande, etc. (0.5 puntos)

El comportamiento de pacman siguiendo el algoritmo de búsqueda en profundidad no es óptimo dado que al explorar los nodos teniendo la lista de abiertos inicializada como una pila puede darse el caso que el algoritmo encuentre la solución en un camino con mayor coste pero que haya sido encontrado antes en su exploración en profundidad. El algoritmo siempre encuentra la solución y presenta un coste espacial lineal. Si bien esta opción puede ser la que menor tiempo computacional y menor número de expansiones de nodos

haga antes de encontrar una solución, generalmente no encontrará el camino de coste óptimo hasta llegar a la meta.

### **1.2.1. Respuesta a pregunta 1.1**

Sí, pues el orden de exploración es Oeste, Este, Sur y Norte; y se corresponde con las casillas exploradas que muestra por terminal la ejecución de los tres programas. En estas ejecuciones se ve como en las bifurcaciones se actúa según ese orden.

### **1.2.2. Respuesta a pregunta 1.2**

No, Pacman no va a todas las casillas exploradas, pues nuestro algoritmo no devuelve la lista con las acciones que hemos realizado para la exploración, sino que una vez hallada la solución (no tiene que ser la óptima) devuelve la lista de acciones que llevan al final (desde el estado inicial al estado meta o final) sin la necesidad de retroceder y estableciendo solo el camino encontrado por el algoritmo.

### **1.2.3. Respuesta a pregunta 2**

Como ya hemos comentado, vemos claramente en el laberinto de tamaño mediano que la solución no es la de menor coste. Ya hemos estudiado en clase que la búsqueda en profundidad no asegura encontrar una solución ni asegura que la solución encontrada sea óptima.

## **Sección 2 (1 punto)**

### **2.1. Comentario personal en el enfoque y decisiones de la solución propuesta (0.5 puntos)**

Para solucionar este problema de búsqueda hemos usado los conocimientos aprendidos en la teoría de la asignatura, donde se nos ha proporcionado un pseudocódigo de la búsqueda en grafo (con eliminación de estados repetidos). En este caso teníamos que implementar una búsqueda en anchura; hemos empleado una cola (Queue) como lista de abiertos de forma que el primer nodo que se haya insertado en la lista de abiertos sea el primero en salir y ser explorado. Nos hemos servido de la función general (`generalSearch()`), de forma que el código es muy similar al del ejercicio de búsqueda en profundidad. Para ir guardando la solución hemos guardado (al igual que en el ejercicio anterior) en una lista los tripletes formados por el nodo, su lista de sus acciones y el coste hasta llegar a él.

#### **2.1.1. Lista & explicación de las funciones del framework usadas**

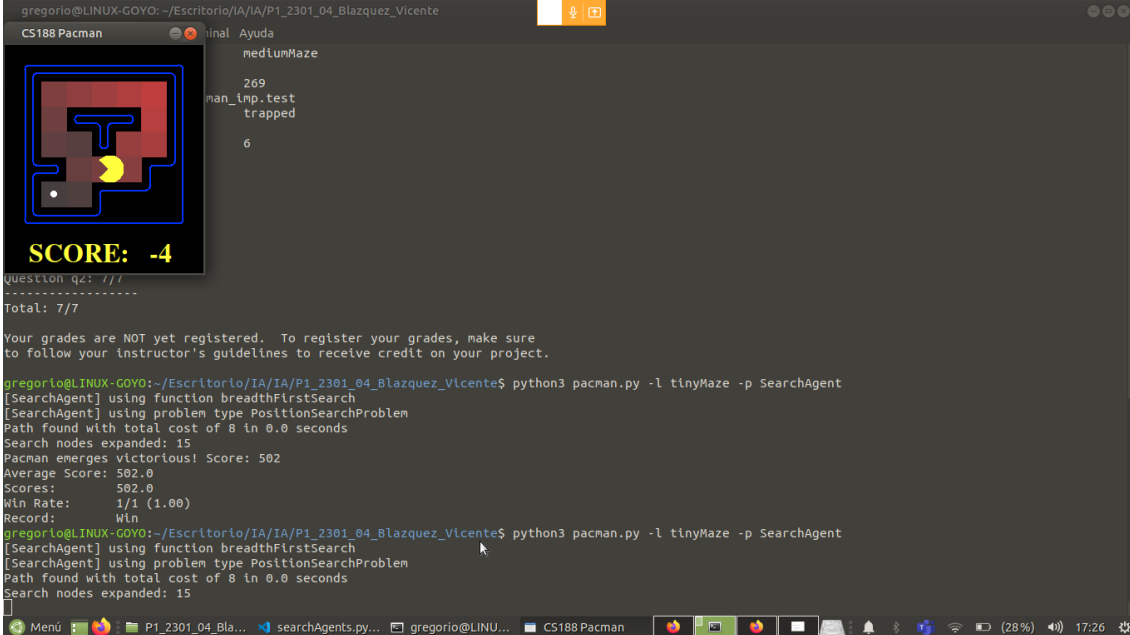
Para la Cola hemos tenido que usar las funciones que se nos proporcionan *push*, *pop* e *isEmpty* para operar con la cola (Queue) según el algoritmo.

Para trabajar con el problema/nodos hemos tenido que usar las tres funciones que se nos proporcionan *getStartState*, *getSuccessors* e *isGoalState*.

## 2.1.2. Incluye el código añadido

```
# Inicializamos la lista de abiertos como una cola FIFO para la búsqueda en profundidad
listaAbiertos=util.Queue()
return generalSearch(problem,listaAbiertos)
```

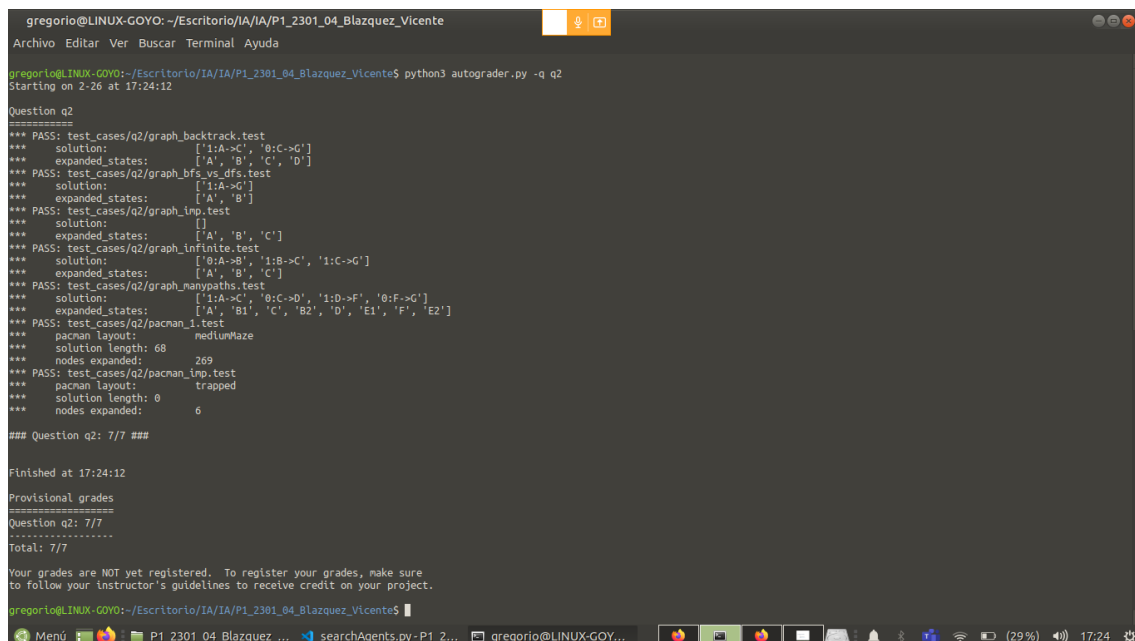
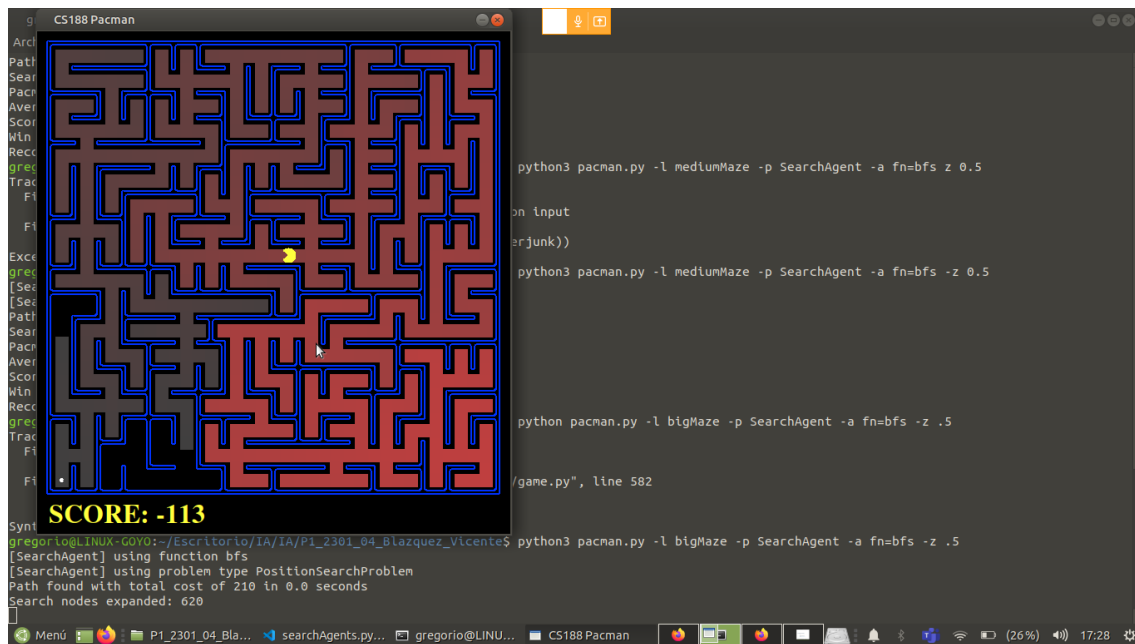
## 2.1.3. Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados



```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
CS188 Pacman
mediumMaze
269
man_tmp.test
trapped
6
SCORE: -4
Question q2: 7/7
Total: 7/7
Your grades are NOT yet registered. To register your grades, make sure to follow your instructor's guidelines to receive credit on your project.
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function breadthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores: 502.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function breadthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
```



```
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores: 502.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumMaze -p SearchAgent -a fn=bfs -z 0.5
Traceback (most recent call last):
  File "pacman.py", line 679, in <module>
    args = readCommand( sys.argv[1:] ) # Get game components based on input
  File "pacman.py", line 529, in readCommand
    raise Exception('Command line input not understood: ' + str(otherJunk))
Exception: Command line input not understood: ['z', '0.5']
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
SCORE: -57
```



Como podemos observar en las capturas de pantalla proporcionadas, el número de nodos expandidos es bastante elevado dado que antes de pasar al siguiente nivel de exploración el algoritmo expande todos los nodos del nivel actual. Sin embargo, el coste del camino encontrado es el ínfimo entre los posibles y se trata de un algoritmo óptimo en cuanto a su coste de búsqueda.

## 2.2. Conclusiones en el comportamiento de pacman, es óptimo (s/n), llega a la solución (s/n), nodos que expande, etc. (0.5 puntos)

El comportamiento de pacman siguiendo el algoritmo de búsqueda en profundidad sí es óptimo dado que al explorar los nodos teniendo la lista de abiertos inicializada como una cola, se van expandiendo en orden todos los nodos de un mismo nivel de profundidad hasta llegar a la solución. De esta forma devolveremos la lista con el menor número de acciones hasta llegar al nodo meta, y la primera solución encontrada (y la retornada) será

la óptima. El algoritmo siempre encuentra la solución y presenta un coste espacial exponencial. Esta opción no suele ser la de menor tiempo computacional y tampoco presenta el menor número de expansiones de nodos antes de encontrar una solución. No obstante, el camino encontrado hasta la solución será el de menor coste y por tanto es óptimo.

### 2.2.1. Respuesta a pregunta 3

Sí, aunque la expansión con este algoritmo de búsqueda en anchura es mayor respecto a búsqueda en profundidad (269 nodos frente a 146 en `mediumMaze` por poner un ejemplo), se puede ver una clara diferencia entre el coste en profundidad (130) y en anchura (68). Una vez más queda reflejado que se trata de un algoritmo óptimo, que encuentra la solución con el menor coste posible.

## Sección 3 (1 punto)

### 3.1. Comentario personal en el enfoque y decisiones de la solución propuesta (0.5 puntos)

En este caso teníamos que implementar una búsqueda de coste uniforme; hemos empleado una cola de prioridad (`PriorityQueueWithFunction`) como lista de abiertos de forma que los primeros nodos en salir de la lista de abiertos sean los de menor coste (del camino desde el nodo inicial). Para su implementación hemos utilizado una expresión lambda mediante la cual utilizamos el coste (tercer elemento de nuestro triplete de nodos) para ordenar los elementos a la hora de insertarlos en la lista de abiertos.

#### 3.1.1. Lista & explicación de las funciones del framework usadas

Para la *ColaDePrioridad* hemos tenido que usar las funciones que se nos proporcionan *push*, *pop* e *isEmpty* para operar con la cola de prioridad (`PriorityQueueWithFunction`) según el algoritmo.

Para trabajar con el problema/nodos hemos tenido que usar las tres funciones que se nos proporcionan *getStartState*, *getSuccessors* e *isGoalState*.

Cabe destacar el uso de la expresión lambda pasada a la cola de prioridad para gestionar la ordenación de sus elementos a la hora de hacer el push en función del coste de su camino desde el estado inicial.

#### 3.1.2. Incluye el código añadido

```
# Inicializamos la lista de abiertos como una cola de prioridad que ordena en funcion del tercer
elemento del nodo a insertar, es decir, el coste g
listaAbiertos=util.PriorityQueueWithFunction(lambda item: item[2])
return generalSearch(problem,listaAbiertos)
```

### 3.1.3. Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados

```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
*** PASS: test_cases/q2/graph_infinite.test
*** solution: ['0:A->B', '1:B->C', '1:C->G']
*** expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_manypaths.test
***
*** PA
***
*** PA
***
***
## Qui
Finish
Provis
=====
Questi
-----
Total:
SCORE: -48
Your g
to follow your instructor's guidelines to receive credit on your project.

gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumMaze -p SearchAgent -a fn=ucs -z 0.5
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
```

```
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda

Finished at 17:33:43
Provisional grades
=====
Question q2: 7/7
-----
Total: 7/7
Your grades a
to follow you

gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumDottedMaze -p StayEastSearchAgent
[SearchAgent]
[SearchAgent]
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores: 646.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumDottedMaze -p StayEastSearchAgent -z 0.5
[SearchAgent]
[SearchAgent]
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
```



```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumDottedMaze -p StayEastSearchAgent
CS188 Pacman
y -l mediumDottedMaze -p StayEastSearchAgent -z 0.5
-l mediumScaryMaze -p StayWestSearchAgent
82
SCORE: -36

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores: 418.0
Win Rate: 1/1 (1.00)
Record: Win

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumScaryMaze -p StayWestSearchAgent -z 0.5
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
```

```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
*** expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q3/graph_manypaths.test
*** solution: ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
*** expanded_states: ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q3/pacman_imp.test
*** pacman layout: trapped
*** solution length: 0
*** nodes expanded: 0
*** PASS: test_cases/q3/ucs_0_graph.test
*** solution: ['Right', 'Down', 'Down']
*** expanded_states: ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q3/ucs_1_problemC.test
*** pacman layout: mediumMaze
*** solution length: 68
*** nodes expanded: 269
*** PASS: test_cases/q3/ucs_2_problemE.test
*** pacman layout: mediumMaze
*** solution length: 74
*** nodes expanded: 260
*** PASS: test_cases/q3/ucs_3_problemW.test
*** pacman layout: mediumMaze
*** solution length: 152
*** nodes expanded: 173
*** PASS: test_cases/q3/ucs_4_testSearch.test
*** pacman layout: testSearch
*** solution length: 7
*** nodes expanded: 14
*** PASS: test_cases/q3/ucs_5_goalAtDequeue.test
*** solution: ['1:A->B', '0:B->C', '0:C->G']
*** expanded_states: ['A', 'B', 'C']

### Question q3: 7/7 ###

Finished at 17:42:35

Provisional grades
=====
Question q3: 7/7
-----
Total: 7/7

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

En nuestra primera ejecución podemos observar que el coste y el número de nodos expandidos son exactamente lo mismo que la búsqueda en anchura. Esto se debe a que con todos los costes de las acciones iguales la búsqueda de coste uniforme es igual a la de anchura.

En el caso de la ejecución con StayEastSearchAgent, la función de coste de las acciones del camino se asocian de la siguiente manera: el coste para avanzar a la posición (x,y) es  $\frac{1}{2}^x$ . De esta manera el coste hasta las posiciones más al este será menor que el resto de las posiciones y Pacman explorará siempre todo lo que sea posible hacia el este. El coste exponencial con un factor menor que 1 supone costes de desplazamiento (acciones) entre distintos estados muy bajo, podemos observar que el coste del camino óptimo también es muy bajo: 1.

En el caso de la ejecución con StayWesttSearchAgent, la función de coste de las acciones del camino se asocian de la siguiente manera: el coste para avanzar a la posición (x,y) es  $2^x$ . De esta manera el coste hasta las posiciones más al este será mayor que el resto de las posiciones y Pacman explorará siempre todo lo que sea posible hacia el oeste. El coste exponencial con un factor mayor supone costes de desplazamiento (acciones) entre distintos estados muy alto, podemos observar que el coste del camino óptimo también tiene un coste abismal.

### **3.2. Conclusiones en el comportamiento de pacman, es óptimo (s/n), llega a la solución (s/n), nodos que expande, etc. (0.5 puntos)**

El comportamiento de pacman siguiendo el algoritmo de búsqueda de coste uniforme sí es óptimo dado que al explorar los nodos teniendo en cuenta el coste de cada acción y ordenando la exploración por coste del camino (explorando primero el nodo que suponga el menor coste del camino). De esta forma devolveremos la lista con el menor coste hasta llegar al nodo meta, y la primera solución encontrada (y la retornada) será la óptima. El algoritmo siempre encuentra la solución y presenta un coste espacial exponencial. Esta opción no suele ser la de menor tiempo computacional y tampoco presenta el menor número de expansiones de nodos antes de encontrar una solución. Sin embargo, el camino encontrado hasta la solución será el de menor coste y por tanto es óptimo.

## **Sección 4 (2 puntos)**

### **4.1. Comentario personal en el enfoque y decisiones de la solución propuesta (1 punto)**

En este caso teníamos que implementar una búsqueda de coste uniforme; hemos empleado una cola de prioridad (PriorityQueueWithFunction) como lista de abiertos de forma que los primeros nodos en salir de la lista de abiertos sean los de menor coste  $f(n)=g(n)+h(n)$  que suma el coste actual desde el nodo inicial hasta el actual con el coste estimado por la heurística que le pasamos a la función aStar desde el nodo actual hasta el meta. Para su implementación hemos utilizado una expresión lambda mediante la cual utilizamos la suma del coste actual del nodo con la función de heurística (ordenamos en función del valor  $f=g+h$ ) para ordenar los elementos a la hora de insertarlos en la lista de abiertos.

#### **4.1.1. Lista & explicación de las funciones del framework usadas**

Para la *ColaDePrioridad* hemos tenido que usar las funciones que se nos proporcionan *push*, *pop* e *isEmpty* para operar con la cola de prioridad (PriorityQueueWithFunction) según el algoritmo.

Para trabajar con el problema/nodos hemos tenido que usar las tres funciones que se nos proporcionan *getStartState*, *getSuccessors* e *isGoalState*.

Cabe destacar el uso de la expresión lambda pasada a la cola de prioridad para gestionar la ordenación de sus elementos a la hora de hacer el push en función de la suma coste de su camino desde el estado inicial hasta el actual con la evaluación de la función de heurística.

## 4.1.2. Incluye el código añadido

```
# Inicializamos la lista de abiertos como una cola de prioridad que ordena en funcion
# del tercer elemento del nodo a insertar mas la heuristica, es decir, el coste f=g+h
listaAbiertos=util.PriorityQueueWithFunction(lambda item: item[2]+heuristic(item[0],problem))
return generalSearch(problem,listaAbiertos)
```

## 4.1.3. Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados



```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vic
Archivo Editar Ver Buscar Terminal Ayuda
*** solution length: 152
*** nodes expanded: 173
*** PASS: test_cases/q3/ucs_4_testSearch.test
*** pacman layout: testSearch
*** solution length: 7
*** nodes expanded: 14
*** PASS: test_cases/q3/ucs_5_goalAtDequeue.test
*** solution: ['1:A->B', '0:B->C', '0:C->G']
*** expanded_states: ['A', 'B', 'C']

### Question q3: 7/7 ###

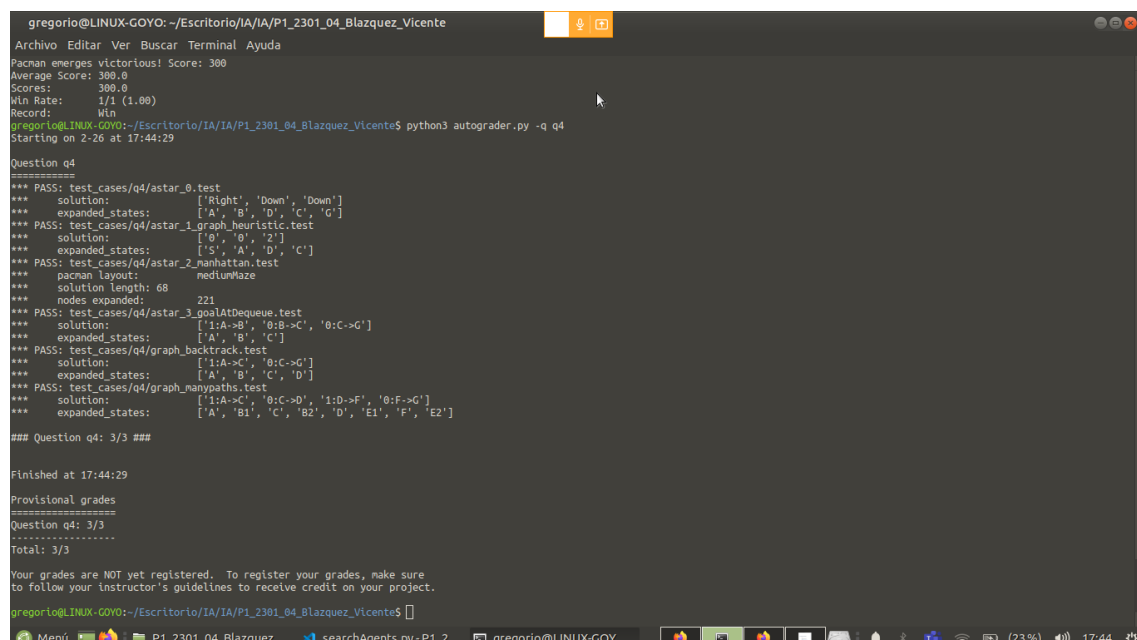
Finished at 17:42:35

Provisional grades
=====
Question q3: 7/7
-----
Total: 7/7

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your proje

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ py
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ py
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores: 300.0
Win Rate: 1/1 (1.00)
Record: Win

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l bigMaze -z 5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
```



```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores: 300.0
Win Rate: 1/1 (1.00)
Record: Win

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 autograder.py -q q4
Starting on 2-26 at 17:44:29

Question q4
=====
*** PASS: test_cases/q4/astar_0.test
*** solution: ['Right', 'Down', 'Down']
*** expanded_states: ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q4/astar_1_graph_heuristic.test
*** solution: ['0', '0', '2']
*** expanded_states: ['0', 'A', 'D', 'C']
*** PASS: test_cases/q4/astar_2_manhattan.test
*** pacman layout: mediumMaze
*** solution length: 68
*** nodes expanded: 221
*** PASS: test_cases/q4/astar_3_goalAtDequeue.test
*** solution: ['1:A->B', '0:B->C', '0:C->G']
*** expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q4/graph_backtrack.test
*** solution: ['1:A->C', '0:C->G']
*** expanded_states: ['A', 'B', 'C', 'D']
*** PASS: test_cases/q4/graph_manypaths.test
*** solution: ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
*** expanded_states: ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###

Finished at 17:44:29

Provisional grades
=====
Question q4: 3/3
-----
Total: 3/3

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$
```

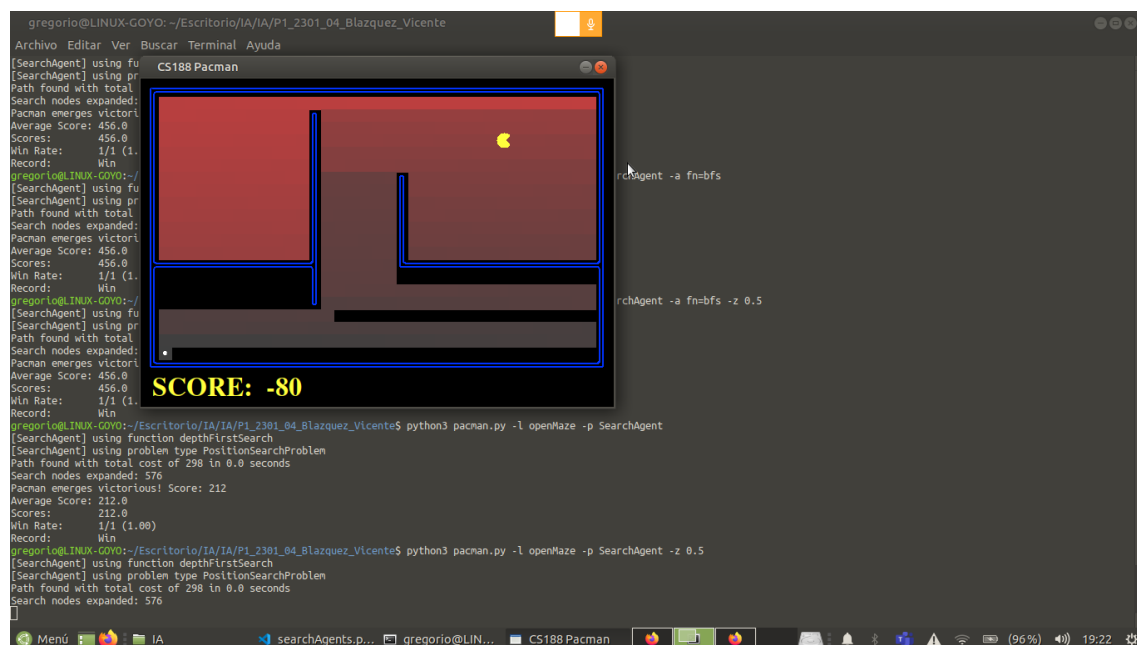
Tras la ejecución del algoritmo Astar con la heurística de Manhattan podemos apreciar que el número de nodos expandidos corresponde con el valor esperado en la presentación de la práctica (549), que es ligeramente inferior a la implementación dada, utilizando la heurística nula (todas las acciones tienen coste 0). Esto se debe a que la heurística de Manhattan es siempre superior a la heurística nula en todo nodo, y al ser admisible

podemos determinar que esta domina a la heurística nula (es mejor). Como consecuencia, el número de nodos expandidos será generalmente menos que en algoritmo con heurísticas que sean inferiores.

#### 4.2. Conclusiones en el comportamiento de pacman, es óptimo (s/n), llega a la solución (s/n), nodos que expande, etc (1 punto)

Se ha notado una mejoría dado que al añadir la heurística de Manhattan primero se exploran los nodos que están más cerca de la meta. Aunque esto no garantiza mejoría, generalmente es una buena idea explorar primero los nodos en dirección a la meta como se está haciendo y esto se refleja en que al igual que dicen en el enunciado hemos expandido 71 nodos menos. También hay que destacar que al ser claramente la distancia Manhattan una heurística admisible y consistente (viene de la relajación del problema quitando los muros) tenemos garantizado con A\* una solución óptima con un coste de búsqueda óptimo.

##### 4.2.1. Respuesta a pregunta 4



The screenshot shows a terminal window with the following content:

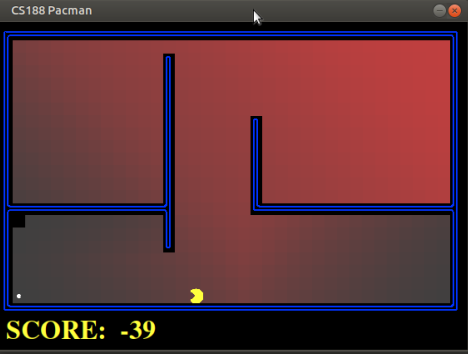
```
gregorio@LINUX-GOVO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 298 in 0.0 seconds
Search nodes expanded: 576
Pacman emerges victorious! Score: 212
Average Score: 212.0
Scores: 212.0
Win Rate: 1/1 (1.00)
Record: Win

gregorio@LINUX-GOVO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l openMaze -p SearchAgent -z 0.5
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 298 in 0.0 seconds
Search nodes expanded: 576
```

The terminal also displays a Pacman game window titled "CS188 Pacman". The game board shows Pacman (yellow dot) and a ghost (blue dot). The score is -80. The terminal output shows that the game was won with a score of 456.0.

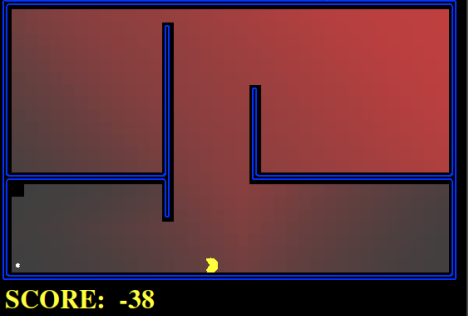
En el caso de la búsqueda en profundidad, como ya hemos comentado con anterioridad, tiene poco coste computacional (encuentra la solución con relativamente pocos nodos expandidos), sin embargo, el coste obtenido es muy elevado respecto al coste óptimo que veremos en ejecuciones posteriores (algoritmo de búsqueda en anchura).

```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
Average Score: 456.0
Scores: 456.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacnan.py -l openMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores: 456.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacnan.py -l openMaze -p SearchAgent -a fn=ucs -z 0.5
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacnan.py -l openMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores: 456.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacnan.py -l openMaze -p SearchAgent -a fn=ucs -z 0.5
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
```

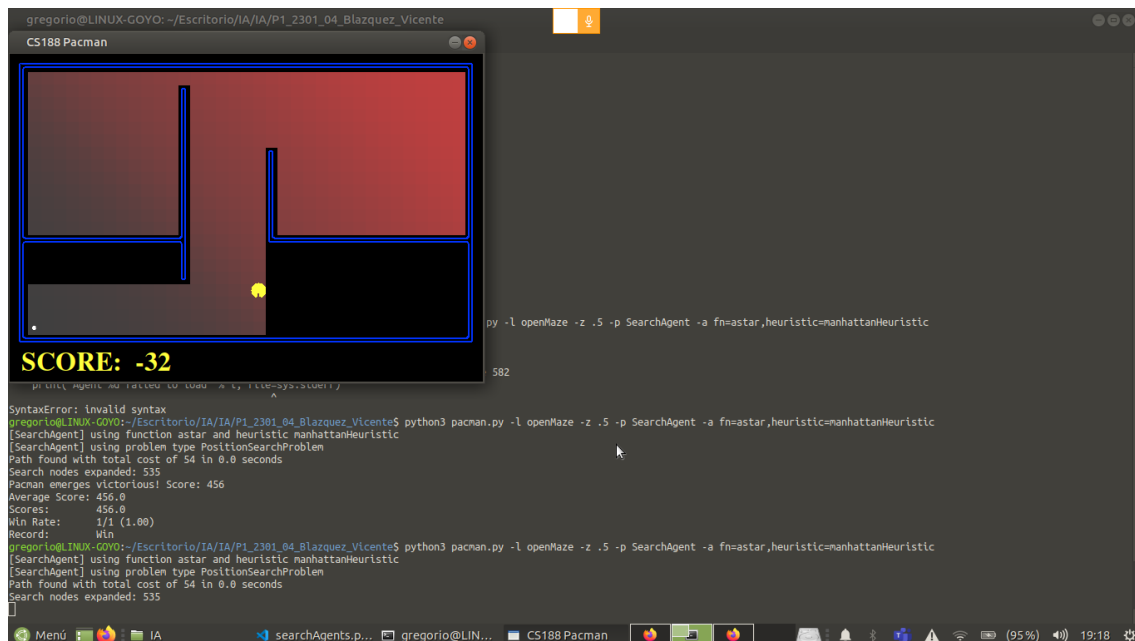


En el caso de la búsqueda en anchura podemos observar que el número de nodos expandidos es muy elevado, lo que corresponde a un coste computacional muy elevado. Por contrapartida, el coste total del camino es el óptimo (y mucho menor que el obtenido en profundidad).

```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacnan.py -l openMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores: 456.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacnan.py -l openMaze -p SearchAgent -a fn=ucs -z 0.5
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacnan.py -l openMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores: 456.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacnan.py -l openMaze -p SearchAgent -a fn=bfs -z 0.5
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
```



En el caso de la búsqueda de coste uniforme podemos observar que es exactamente la misma solución y los mismos resultados que en búsqueda en anchura, dado que todas las acciones tienen el mismo coste.



```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
CS188 Pacman

SCORE: -32

python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
582

SyntaxError: invalid syntax
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path Found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores: 456.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path Found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
```

En el caso de la búsqueda Astar empleando la heurística de Manhattan, podemos observar que dentro de nuestras opciones se corresponde con la de mejores resultados relativo a que garantiza la solución óptima con el menor coste de nodos expandidos respecto a los otros algoritmos óptimos.

## Sección 5 (2 puntos)

### 5.1. Comentario personal en el enfoque y decisiones de la solución propuesta (1 punto)

Hemos formalizado el problema, ya pensando en la implementación que íbamos a realizar, definiendo los estados como la posición de pacman en el tablero dada por una dupla de coordenadas cartesianas en el mapa y un conjunto de las esquinas visitadas. De esta manera no cometemos el error de considerar que es el mismo estado pacman en una casilla habiendo visitado dos esquinas que en esa misma casilla sin haber visitado ninguna esquina. Las acciones siguen siendo las de antes (moverse al SUR, NORTE, ESTE y OESTE) y los costes también se mantienen como costes fijos de valor 1 a no ser que se especifique lo contrario. Así pues al implementar el código consideramos que hemos solucionado el problema cuando ya se han visitado las 4 esquinas y al generar los sucesores copiamos el conjunto de esquinas visitadas por el padre y en caso de ser el hijo una esquina añadimos ésta al conjunto.

#### 5.1.1. Lista & explicación de las funciones del framework usadas

#### 5.1.2. Incluye el código añadido

```
class CornersProblem(search.SearchProblem):
    """
    This search problem finds paths through all four corners of a layout.
    You must select a suitable state space and successor function
    """
```

```

def __init__(self, startingGameState):
    """
    Stores the walls, pacman's starting position and corners.
    """
    self.walls = startingGameState.getWalls()
    self.startingPosition = startingGameState.getPacmanPosition()
    top, right = self.walls.height-2, self.walls.width-2
    self.corners = ((1,1), (1,top), (right, 1), (right, top))
    for corner in self.corners:
        if not startingGameState.hasFood(*corner):
            print('Warning: no food in corner ' + str(corner))
    self._expanded = 0 # DO NOT CHANGE; Number of search nodes expanded
    # Please add any code here which you would like to use
    # in initializing the problem
    """*** YOUR CODE HERE ***"""

def getStartState(self):
    """
    Returns the start state (in your state space, not the full Pacman state
    space)
    """
    """*** YOUR CODE HERE ***"""
    return (self.startingPosition, set())

def isGoalState(self, state):
    """
    Returns whether this search state is a goal state of the problem.
    """
    """*** YOUR CODE HERE ***"""

    if len(state[1]) == 4:
        return True

    return False

def getSuccessors(self, state):
    """
    Returns successor states, the actions they require, and a cost of 1.
    As noted in search.py:
    For a given state, this should return a list of triples, (successor,
    action, stepCost), where 'successor' is a successor to the current
    state, 'action' is the action required to get there, and 'stepCost'
    is the incremental cost of expanding to that successor
    """
    """*** YOUR CODE HERE ***"""
    successors = []
    for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:
        x,y = state[0]
        dx, dy = Actions.directionToVector(action)
        nextx, nexty = int(x + dx), int(y + dy)
        if not self.walls[nextx][nexty]:
            nextState = (nextx, nexty)
            conjuntoEsquinas=state[1].copy()
            if nextState in self.corners:
                if nextState not in state[1]:
                    conjuntoEsquinas.add(nextState)

```

```

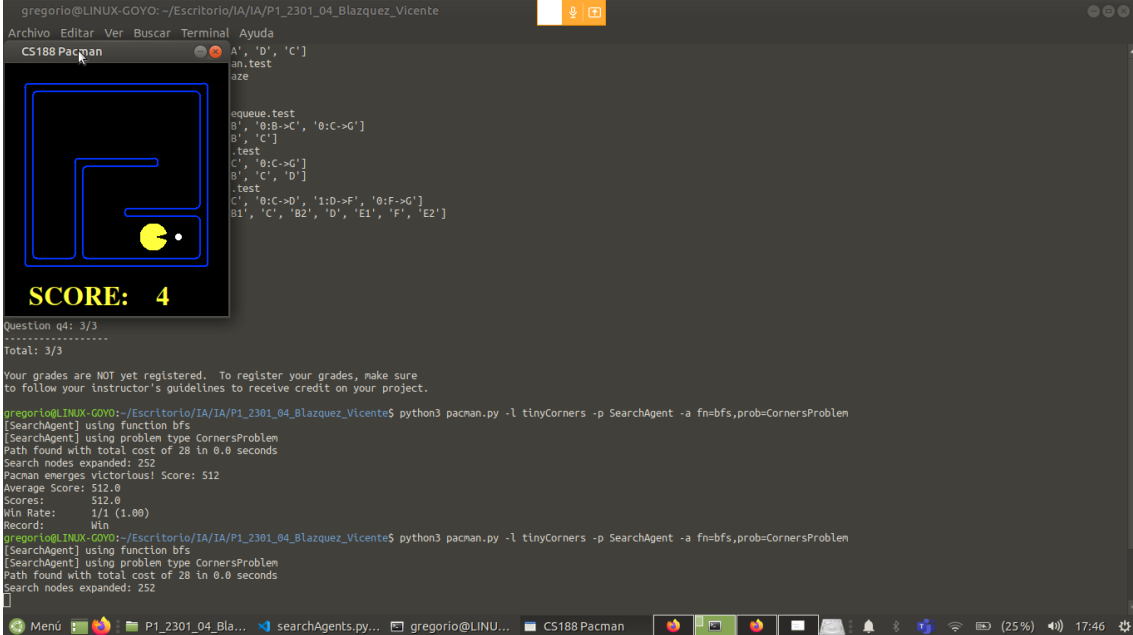
        successors.append( ( (nextState,conjuntoEsquinas), action, 1) )

    self._expanded += 1 # DO NOT CHANGE
    return successors

def getCostOfActions(self, actions):
    """
    Returns the cost of a particular sequence of actions. If those actions
    include an illegal move, return 999999. This is implemented for you.
    """
    if actions == None: return 999999
    x,y= self.startingPosition
    for action in actions:
        dx, dy = Actions.directionToVector(action)
        x, y = int(x + dx), int(y + dy)
        if self.walls[x][y]: return 999999
    return len(actions)

```

### 5.1.3. Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados



```

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
CS188 Pacman
A', 'D', 'C']
an.test
aze

equeue.test
B', '0:B->C', '0:C->G']
B', 'C']
.test
C', '0:C->G']
B', 'C', 'D']
.test
C', '0:C->D', '1:D->F', '0:F->G']
B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

SCORE: 4

Question q4: 3/3
-----
Total: 3/3

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores: 512.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO:~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252

```

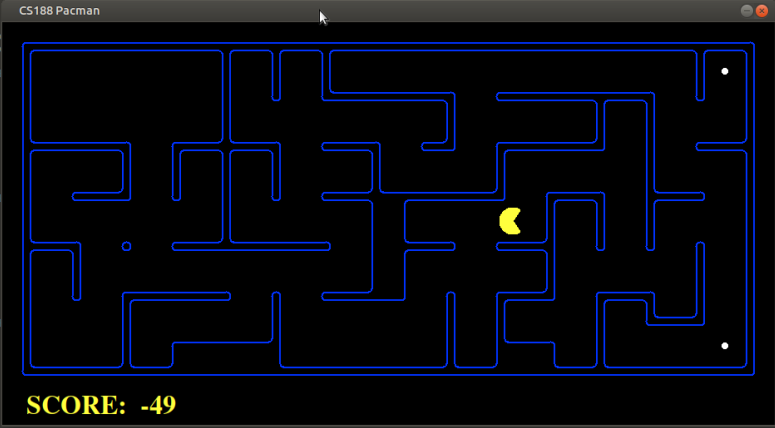
Observamos que el coste es óptimo (28 como se indica en el enunciado) y que se encuentra con 252 nodos expandidos, una cantidad razonable.



```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
Provisional grades
=====
Question q4: 3/3
Total: 3/3

Your grades are NOT yet registered. To register y
to follow your instructor's guidelines to receive y

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores: 512.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores: 512.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.2 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores: 434.0
Win Rate: 1/1 (1.00)
Record: Win
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 1966
]

CS188 Pacman

SCORE: -49
```

Encuentra la solución correctamente y coincidimos con en el número de nodos expandidos que se indica en el enunciado, algo menos de 2000.

```
gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
*** solution: ['1:A->G']
*** expanded_states: ['A', 'B']
*** PASS: test_cases/q2/graph_ino.test
*** solution: []
*** expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_infinite.test
*** solution: ['0:A->B', '1:B->C', '1:C->G']
*** expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_manypaths.test
*** solution: ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
*** expanded_states: ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q2/pacman_1.test
*** pacman layout: mediumMaze
*** solution length: 68
*** nodes expanded: 269
*** PASS: test_cases/q2/pacman_ino.test
*** pacman layout: trapped
*** solution length: 0
*** nodes expanded: 6

## Question q2: 7/7 ##

Question q5
=====
*** PASS: test_cases/q5/corner_tiny_corner.test
*** pacman layout: tinyCorner
*** solution length: 28

## Question q5: 3/3 ##

Finished at 17:47:39

Provisional grades
=====
Question q2: 7/7
Question q5: 3/3
Total: 10/10

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$
```

## 5.2. Conclusiones en el comportamiento de pacman, es optimo (s/n), llega a la solución (s/n), nodos que expande, etc (1 punto)

Hemos observado que el comportamiento de Pacman es óptimo. Esto se debe a que al utilizar una heurística trivial, como la que se está utilizando, estamos realizando una búsqueda en anchura que garantiza la solución óptima pero con un coste de ejecución muy alto que se refleja en los nodos expandidos. Más adelante con nuestra heurística vamos a lograr reducirlo a más de la mitad.

## Sección 6 (3 puntos)

### 6.1. Comentario personal en el enfoque y decisiones de la solución propuesta (1.5 puntos)

Para solucionar este apartado y hallar una heurística admisible y consistente hemos utilizado dos ideas: la relajación del problema al no tener en cuenta los muros, y la idea de solucionar el problema tratando de reducirlo a la solución una vez alcanzas una esquina. Tratamos esto en más detalle en el apartado 6.2.1.

#### 6.1.1. Lista & explicación de las funciones del framework usadas

Para que el código quede más claro y sea más flexible hemos definido una función `disManhattan` que recibe dos puntos/duplas y calcula la distancia Manhattan entre esos puntos sin tener en cuenta muros u otros factores externos.

#### 6.1.2. Incluye el código añadido

```
def disManhattan(a,b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def cornersHeuristic(state, problem):
    """
    A heuristic for the CornersProblem that you defined.
    state: The current search state
           (a data structure you chose in your search problem)
    problem: The CornersProblem instance for this layout.
    This function should always return a number that is a lower bound on the
    shortest path from the state to a goal of the problem; i.e. it should be
    admissible (as well as consistent).
    """
    corners = problem.corners # These are the corner coordinates
    walls = problem.walls # These are the walls of the maze, as a Grid (game.py)

    """ YOUR CODE HERE """

    # Calculamos el minimo de las distancias de manhattan a las esquinas no exploradas
    minManhattan=-1
    for corner in corners:
        if corner not in state[1]:
            aux=disManhattan(state[0],corner)
            if (aux<minManhattan) or (minManhattan==-1):
                minManhattan=aux

    #Caso en el que todas las esquinas están encontradas
    if(minManhattan==-1):
        minManhattan=0

    height=abs(corners[0][1]-corners[1][1])
    width=abs(corners[0][0]-corners[2][0])

    lados=[height,width]
    NesquinasEncontradas = len(state[1])
```

```

if NesquinasEncontradas == 0:
    return minManhattan+2*min(lados)+max(lados)
elif NesquinasEncontradas == 1:
    return minManhattan+lados[0]+lados[1]
elif NesquinasEncontradas == 2:
    esquinas = [x for x in corners if x not in state[1]]
    return minManhattan+disManhattan(esquinas[0],esquinas[1])
else:
    return minManhattan

```

### 6.1.3. Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados

Los resultados se analizan más adelante en el apartado 6.2.1.

The screenshot shows a terminal window with the following output:

```

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
Traceback (most recent call last):
  File "gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente/Blazquez_Vicente/game.py", line 582
    (rr)
  File "gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente/Blazquez_Vicente/game.py", line 582
    (rr)

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
Traceback (most recent call last):
  File "gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente/Blazquez_Vicente/game.py", line 582
    (rr)
  File "gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente/Blazquez_Vicente/game.py", line 582
    (rr)

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 774
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores: 434.0
Win Rate: 1/1 (1.00)
Record: Win

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$ python3 pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 774

```

Overlaid on the terminal is a Pacman game window titled "CS188 Pacman". It shows a maze with a yellow Pacman character and a score of -69.

The screenshot shows a terminal window with the following output:

```

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente
Archivo Editar Ver Buscar Terminal Ayuda

*** solution: ['1:A->B', '0:B->C', '0:C->G']
*** expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q4/graph_backtrack.test
*** solution: ['1:A->C', '0:C->G']
*** expanded_states: ['A', 'B', 'C', 'D']
*** PASS: test_cases/q4/graph_manypaths.test
*** solution: ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
*** expanded_states: ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###

====
Question q6
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
paths ['North', 'East', 'East', 'East', 'North', 'North', 'West', 'West', 'West', 'West', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'West', 'South', 'South', 'East', 'East', 'East', 'East', 'South', 'South', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'West', 'West', 'East', 'East', 'North', 'North', 'North', 'North', 'East', 'East', 'East', 'East', 'East', 'East', 'North', 'East', 'North', 'No
rth', 'East', 'East', 'North', 'North', 'East', 'East', 'East', 'South', 'South', 'South', 'South', 'East', 'East', 'East', 'East', 'North', 'East', 'North', 'No
rth', 'East', 'East', 'North', 'North', 'East', 'East', 'East', 'South', 'South', 'South', 'South', 'East', 'East', 'North', 'East', 'East', 'South', 'South', 'South', 'South', 'South', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'North', 'East', 'East', 'North', 'North']
path length: 106
*** PASS: Heuristic resulted in expansion of 774 nodes
path: ['East', 'South', 'South', 'West', 'West', 'South', 'East', 'East', 'East', 'West', 'West', 'West', 'West', 'West', 'North', 'North', 'North', 'North']
path length: 22
*** PASS: Heuristic resulted in expansion of 24 nodes

### Question q6: 6/6 ###

Finished at 17:32:43

Provisional grades
====
Question q4: 3/3
Question q6: 6/6
-----
Total: 9/9

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

gregorio@LINUX-GOYO: ~/Escritorio/IA/IA/P1_2301_04_Blazquez_Vicente$

```

## **6.2. Conclusiones en el comportamiento de pacman, es óptimo (s/n), llega a la solución (s/n), nodos que expande, etc. (1.5 puntos)**

Hemos comprobado que el comportamiento de Pacman es óptimo (coste total de 106), pues sabemos de teoría que una heurística admisible y consistente garantiza exactitud en búsqueda en grafo como estamos realizando. Por supuesto el camino óptimo obtiene la solución y además estamos muy contentos con el rendimiento de pacman con nuestra heurística. Pues en un principio logramos una heurística admisible que nos daba la solución óptima con algo menos de 1100 nodos expandidos y decidimos mejorarla al mismo tiempo que nos aseguramos de que fuera consistente llegando a lograr el resultado final en el que expandimos tan solo 774 nodos.

### **6.2.1. Respuesta a pregunta 5: heurística**

Para hallar una heurística consistente para el problema de las cuatro esquinas hemos seguido el consejo del enunciado y primero hallamos una heurística admisible. Para esto tuvimos desde el primer momento claro que el primer paso era relajar el problema quitando los muros sabiendo que el coste real con los muros sería mayor o igual que sin ellos. Ya sin muros pensamos en que la solución claramente consiste en ir a una esquina (en principio no sabemos cuál) lo más directo posible y una vez llegada a esa esquina ir al resto de esquinas en línea recta por ser el camino más corto. Por tanto, en un primer momento planteamos una heurística que consistía en la distancia a la esquina más cercana (sabiendo que posiblemente esta no sería la esquina óptima que visitar en primer lugar pero que seguro que tenía menor o igual coste). Para que la heurística fuera mejor le sumamos el número de esquinas que faltaban por visitar menos uno por el mínimo entre el largo y ancho del rectángulo que es el tablero. Así pues, teníamos una heurística admisible que se acercaba bastante a la solución sin muros, pero que no era consistente pues en un rectángulo podía cambiar de sumar el largo por el ancho y cambiar mucho la estimación de un cuadrado a otro contiguo. Por eso terminamos la heurística finalmente como la suma de la distancia Manhattan a la esquina más cercana y según las esquinas que queden suma:

-Si solo queda una esquina no suma nada.

-Si quedan dos esquinas la distancia Manhattan de las dos esquinas restantes (el largo o el ancho).

-Si quedan 3 esquinas dos veces el mínimo entre el ancho y el largo y una vez el máximo.

De esta manera hemos separado por casos metiendo la solución óptima para conseguir que nuestro algoritmo sea lo más eficiente posible. Podíamos haber sido más generales, pero entonces la cantidad de nodos expandidos habría sido mayor como habíamos comprobado en las versiones anteriores y por eso decidimos ser muy específicos obteniendo así unos resultados que nos parecen excelentes.

## **Sección 7**

### **-Comentarios personales de la realización de esta práctica:**

Hemos decidido destacar y agradecer el enunciado de la práctica que consideramos completo, bien estructurado y claro; favoreciendo un eficiente desarrollo de la práctica.

En cuanto al contenido de la práctica nos ha parecido interesante, aunque la memoria quita bastante tiempo por su extensión. Ha supuesto un reto entretenido y divertido la búsqueda de una heurística óptima para la resolución de la Sección 6.