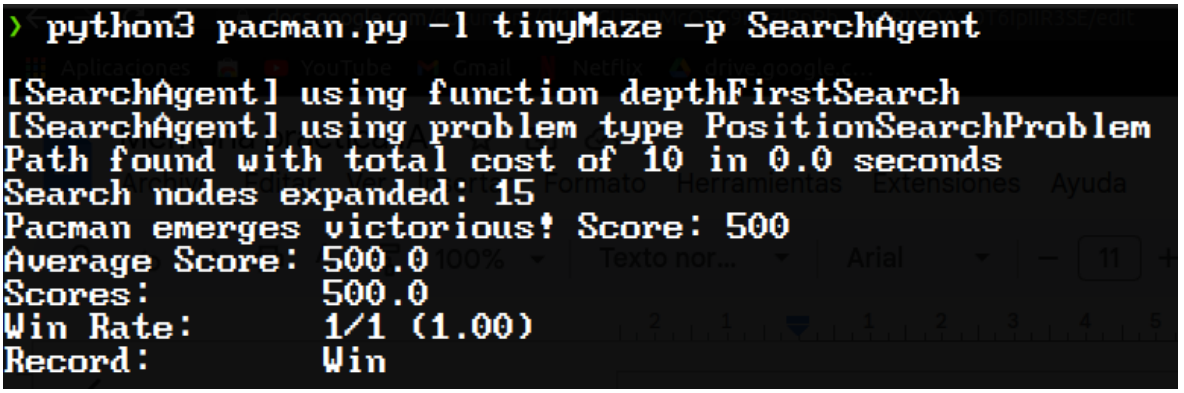
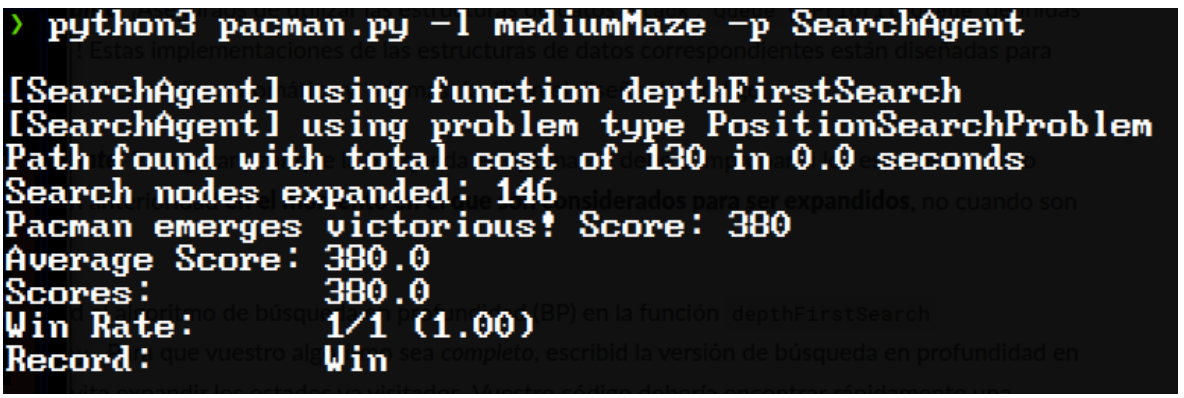
# Práctica 1: Búsqueda

Sergio Hidalgo Gamborino y Alexis Canales Molina

## Sección 1: Encontrar un punto de comida fijo usando la búsqueda primero en profundidad

Este apartado se decidió abarcar realizando un bucle que itera mientras no se encuentre el estado ganador, al iniciar el bucle obtendrá los sucesores del nodo en el que se encuentra (expandiendo el mismo) y si estos no han sido visitados, se pusheará la información del nodo y el recorrido para llegar a él en la estructura de datos correspondiente. En un inicio se realizó la función y en este apartado y luego se copió y pegó en el resto de búsquedas modificando la estructura de datos correspondiente, pero tras una observación del profesor se decidió realizar una función genérica para tener el código más comprimido.
Las pruebas realizadas han sido las siguientes:



```
❯ 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
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Comando: python3 autograder.py -q q1

```
### Question q1: 7/7 ###

Finished at 16:08:46

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

**Pregunta 1.1: ¿El orden de exploración es el que esperabais? ¿Pacman realmente va a todas las casillas exploradas en su camino hacia la meta?**
Sí, pues va en orden en la lista de los sucesores, en caso de cambiar esta parte de código, realizará el camino contrario (línea 105)

```
        for i in reversed(actual_sucessors):
```

No va a todas las casillas exploradas porque los caminos que no llegan a ningún lugar se descartan, para que fuese a todas las casillas exploradas, habría que hacer que una vez llega a un camino sin salida, recorra sobre sus pasos hasta el inicio o hasta la anterior bifurcación, conservando todo el recorrido desde el inicio.

**Pregunta 1.2: ¿Es esta una solución de menor coste? Si no es así, pensad qué está haciendo mal la búsqueda en profundidad.**
En el caso del tiny maze no es la de menor coste (a no ser que se aplique el cambio anteriormente mencionado), no obstante en el medium maze pasa lo contrario. Esto se debe a que al explorar en profundidad, se condiciona el camino de pacman al inicio (izquierda o derecha en el tiny maze o izquierda o abajo en el medium)

# Sección 2: Búsqueda en anchura

Debido a lo mencionado anteriormente, no hay mucho más que comentar respecto a la implementación de esta función (está comentado en genericSearch), salvo que se cambia la estructura de datos de una pila a una cola.

```
> python3 pacman.py -l tinyMaze -p SearchAgent -a fn=bfs

[SearchAgent] using function bfs
[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
```

```
> python3 pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

[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
```

```
> python3 pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

comando: python3 autograder.py -q q2

```
Finished at 16:26:44

Provisional grades
==================
Question q2: 7/7
----------------
Total: 7/7
```

**Pregunta 2.1: ¿BA encuentra una solución de menor coste? Si no es así, verificad vuestra implementación.**

Si que la encuentra.

# Sección 3: Variar la función de coste

De la misma manera que las anteriores funciones, salvo que al llamar a genericSearch se introducirá el parámetro costfn (que es o None o una función que reciba el search_problem y el camino realizado) que devolverá el coste.

El siguiente test realiza lo mismo que el bfs.

```
› 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
```

En el siguiente test se irá por el camino en forma de zig zag porque habrá comida en él (y esto reduce el coste del camino)

```
› python3 pacman.py -l mediumDottedMaze -p StayEastSearchAgent

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
```

En el siguiente test se irá por el camino en de arriba, dado que, de forma análoga al anterior test, pues esquiva a los fantasmas de abajo (dado que aumentarán el coste de la búsqueda)

```
› 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
```

comando: python autograder.py -q q3

```
Finished at 16:36:48

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

# Sección 4: Búsqueda A*

De la misma manera que las anteriores funciones, salvo que al llamar a genericSearch se introducirá el parámetro costfn y heuristic (pasa una función de la heurística o None).

```
❯ python3 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhat
tanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 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
```

### Pregunta 4.1: ¿Qué sucede en `openMaze` para las diversas estrategias de búsqueda?

DFS -> no es óptimo porque recorre de forma horizontal el  laberinto hasta llegar a la fruta

```
❯ python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=dfs

[SearchAgent] using function dfs
[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
```

BFS -> es óptimo (realiza el mismo recorrido que el A* pero expandiendo más nodos

```
❯ python3 pacman.py -l openMaze -z .5 -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
```

UCS -> mismo caso que el bfs

```
❯ python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=ucs

[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.1 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
```

A*-> llega de forma óptima con un coste de 54 y explorando menos nodos

```
> python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manha
ttanHeuristic
[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
```

# Sección 5: Encontrar todas las esquinas

Para implementar este problema de búsqueda se ha usado parte del código de la clase PositionSearchProblem, para las partes visuales del pacman y la funcionalidad básica, la diferencia principal de este problema está en la manera en la que se envían los distintos nodos a la estrategia de búsqueda, por ejemplo, startingPosition, en vez de devolver solo el nodo con la ubicación, lo devolverá con una lista inicializada de las distintas corners vistidadas. Esto se usará tanto en isGoalState, pues se comprobará si se hayan explorado todas las corners, como en getSucessors, en el que se actualizará la lista de las esquinas, de manera que al llegar a una esquina se actualizará dicha lista y el pacman podrá dar media vuelta, pues la estrategia de búsqueda no detectará como estado repetido el estado recibido (pues a pesar de ser el mismo nodo, tendrá una lista de esquinas distintas).

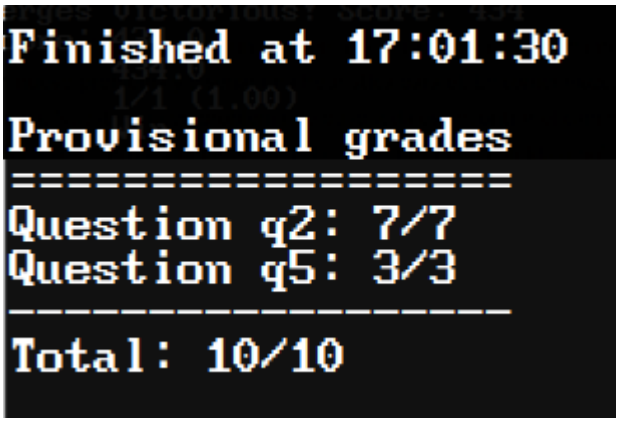
En el siguiente test recorrerá el mapa pasando por todas las esquinas de manera óptima.

```
> 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: 435
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```

En el siguiente recorre todas las esquinas también.

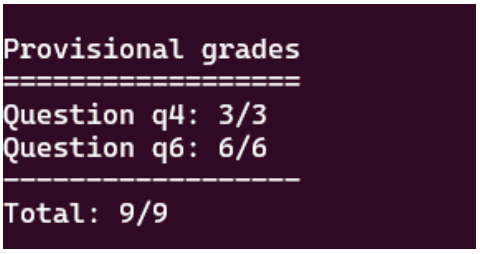
```
> python3 pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProble
m

[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.2 seconds
Search nodes expanded: 2448
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

comando: python autograder.py -q q5




## Sección 6: Problema de las esquinas: heurística

comando: python3 autograder.py -q q6




**Pregunta 6.1: Describe el proceso que se ha seguido para diseñar la heurística y explica la lógica de la misma.**
Primero se ha planteado una versión relajada del problema en la que no hay muros, y después se han obtenido todas las posibles permutaciones entre las esquinas y se ha ido calculando el coste aproximado con la distancia de Manhattan (suma del valor absoluto de la resta de los puntos de los nodos) de cada permutación (incluyendo al inicio de cada combinación el estado actual). De esta manera se sabe cuál va a ser el coste mínimo del camino. En el caso en el que no queden esquinas sin visitar, se habrá cumplido el objetivo y la heurística retornará 0.
Esta heurística es admisible porque es optimista y no sobrestima el resultado (pues no tiene en cuenta los muros y eso le ahorra mucho coste) y además es consistente porque para llegar a todas las esquinas su coste va a ser mayor que el coste estimado de llegar a una sola esquina.

comando: python3 autograder.py