به نام خدا

PACMAN

نویسنده : روژینا کاشفی

شماره دانشجویی: ۹۸۳۱۱۱۸

۱) پیدا کردن یک نقطه ثابت غذا با استفاده از جستجو اول عمق

برای اطمینان از درستی کار کرد searchAgent دستور زیر را با استفاده از tinyMazeSearch اجرا میکنیم.

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
[SearchAgent] using function tinyMazeSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

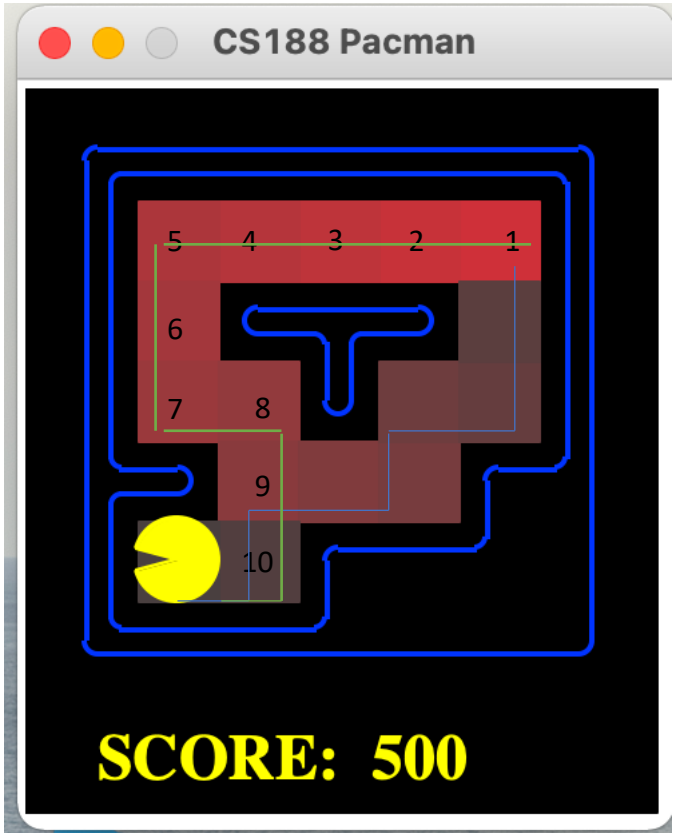
سپس پس از پیاده سازی تابع dfs دستورات زیر را اجرا میکنیم که تفاوت آنها در اندازه زمین مورد بررسی است.

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l mediumMaze -p SearchAgent
[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
rojina@Rojinakashefis-MacBook-Pro Packman AI % python 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
```

طبق راهنمایی سوال اگر الگوریتم را به نحوی پیاده سازی کنیم که به ترتیبی که getSuccessor مشخص کرده به انتهای fringe اضافه کنیم باید طولی برابر ۱۳۰ داشته باشد که دارد

سوال) آیا ترتیب کاوش شده همان ترتیبی بود که انتظار داشتید؟ آیا پکمن در راه رسیدن به هدف، به همه مربع های کاوش شده می‌رود؟

بله همان ترتیب مورد نظر بود زیرا در تابع problem.getSuccessor() ترتیب پیمایش successor ها از شمال، جنوب، شرق، غرب است . چون از پشته استفاده می‌کنیم کسی که آخر اضافه می‌شود بالای پشته است و اول از همه بیرون میرود که اگر حالتی ۴ تا successor داشته باشد جهت غرب اخرین جهت اضافه شده است و اولین عنصر خارج شده.(last in first out)
مطابق شکل زیر که برای tinyMaze است :



مسیر سبز رنگ (به ترتیب getSuccessor)
در مرحله اول در fringe به ترتیب جنوب و غرب وجود دارد که غرب خارج میشود.
از مرحله دو تا ۴ در fringe به ترتیب شرق و غرب وجود دارد که غرب استفاده میشود.
در مرحله ۵ در fringe به ترتیب جنوب و شرق وجود دارد اما چون شرق قبلا بررسی شده بود آن را در نظر نمیگیریم و به پایین میرویم
در مرحله ۶ در fringe به ترتیب شمال و جنوب وجود دارد که جنوب از fringe بیرون می آید
در مرحله ۷ در fringe به ترتیب شمال و شرق وجود دارد که شرق از fringe بیرون می آید.
در مرحله ۸ در fringe به ترتیب جنوب و غرب وجود دارد که چون غرب قبلا بررسی شده بوده به پایین میرویم و به همین منوال ادامه میدهیم .

خیر به همه مربع های کاوش شده نمیرود. به عنوان مثال در شکل فوق مسیر آبی نیز کاوش شده است(حتی زودتر از سبز چون کمرنگ است و همینطور هزینه کمتری از مسیر سبز دارد) اما در الگوریتم dfs چون یکی از مربع های کاوش شده را انتخاب میکند بسته به ترتیب که از fringe خارج میشود و تا عمق آن میرود دیگر به بقیه مربع های کاوش شده نمیرود.

سوال) آیا این راه‌حل کمترین هزینه را دارد؟ اگر نه فکر کنید که جستجوی اول عمق چه کاری را اشتباه انجام می‌دهد؟

خیر مسیر دوم که با آبی نشان داده شده است هزینه کمتر دارد به دلیل ترتیب push و pop از پشته اما در حالت کلی در dfs به هزینه اهمیت داده نمیشود چون تا عمق و رسیدن به آخرین برگ میرود و اگر درست نباشد و به هدف نرسد مجبور است برگردد و اگر هزینه انتقال بین حالت های مختلف فرق کند اصلا هزینه را در نظر نمی‌گیرد و فقط به رسیدن عمق فکر میکند (بدون در نظر گرفتن هزینه و موقعیت هدف که آیا درست است یا نه).

۲) جستجوی اول سطح

با پیاده سازی الگوریتم bfs با استفاده از صف خروجی های زیر را برای زمین های مختلف مشاهده می‌کنیم.
الگوریتم bfs تمام حالت با عمق یکسان را بررسی میکند و سپس در صورت نبود جواب به عمق بعدی میرود به همین دلیل هدف را سریعتر پیدا میکند.
اگر مقایسه کنیم در حالت mediumMaze در dfs با هزینه ۱۳۰ هدف را پیدا کرد اما در bfs با هزینه ۶۸.
و در حالت bigMaze در dfsو bfs هردو با هزینه ۲۱۰.

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python 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
rojina@Rojinakashefis-MacBook-Pro Packman AI % python 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
```

۳) تغییر تابع هزینه

در این نوع جستجو حالت های مختلف هزینه های مختلفی دارد و عامل با توجه به آنکه کدوم هزینه کمتری دارد حرکت میکند.

در حالات زیر در تابع هزینه استفاده شده متفاوت هستند.

در حالت اول تمام هزینه ها ۱ در نظر گرفته شده است که مانند bfs عمل میکند چون اولویت همه یکسان است priority queue همان queue میشود که مشاهده میکنیم تمامی موارد با bfs برابری میکند.

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python 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
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l bigMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 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
```

و برای حالت stayEastAgent هزینه بسیار کمی در نظر گرفته شده است.

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python 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
```

و در حالت stayWestSearch هزینه بسیار زیادی مصرف کرده است.

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python 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
```

۴) جستجوی A استار

برای محاسبه a استار از رابطه روبه‌رو استفاده میکنیم f(n) = g(n) + h(n)
جستجوی A استار با استفاده از heuristic کار میکند به عنوان پیش فرض heuristic در نظر گرفته شده برابر null است و مقداری برابر صفر دارد .
اگر مقدار heuristic صفر باشد همان ucs میشود.
در حالت اول از heuristic پیش فرض استفاده کردیم که همان nullHeuristic است که برای همه صفر است و همان ucs است (f(n) = g(n) + 0) و نتایج آن با ucs برابری میکند.

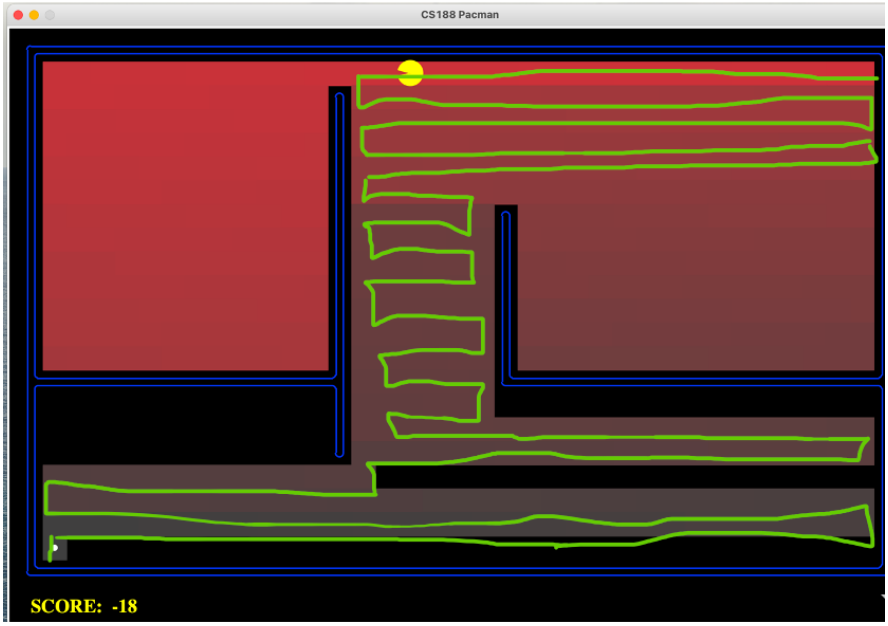
```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l mediumMaze -p SearchAgent -a fn=astar
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

اما اگر heuristic آن را به Manhattandistance تغییر دهیم مشاهده میکنیم برای bigMaze نسبت به usc حالت های کمتری گسترش داده است.

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l bigMaze -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 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
```

سوال) الگوریتم های جستجویی که تا این مرحله پیاده‌سازی کرده‌اید را روی openMaze اجرا کنید و توضیح دهید چه اتفاق می افتد؟

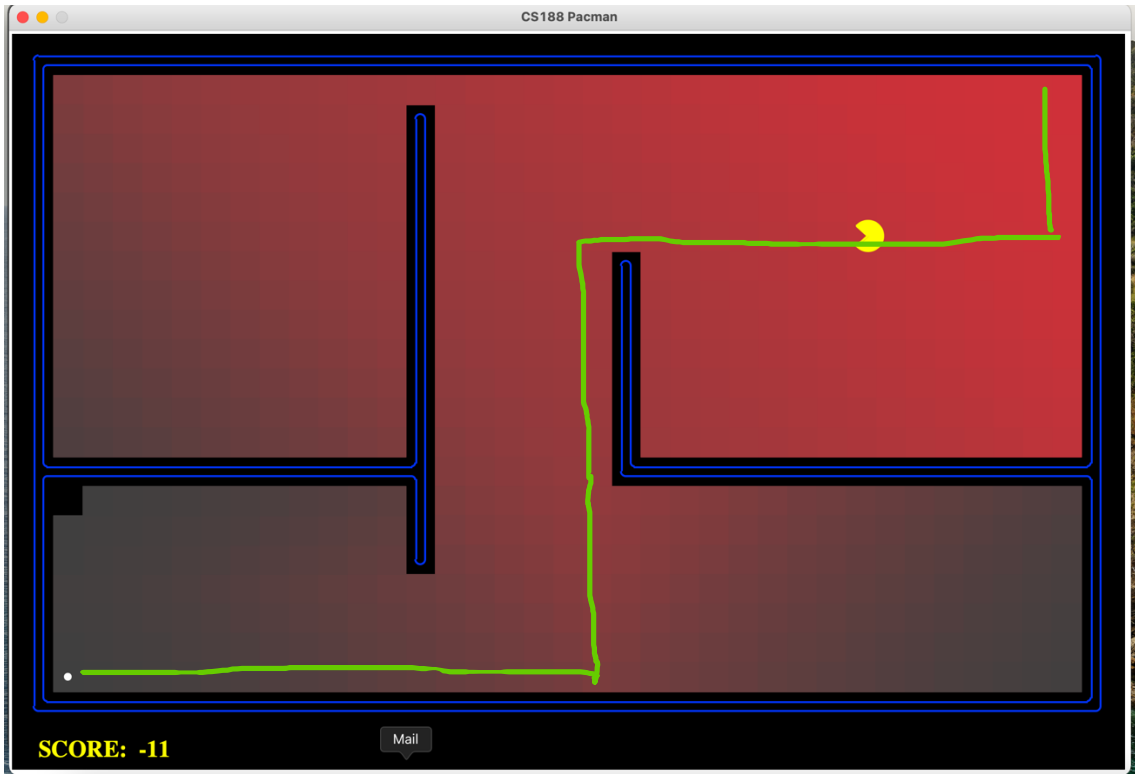
- **الگوریتم DFS**




```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l openMaze -p SearchAgent
[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
```

همانطور که در شکل مشاهده میکنیم ناحیه سمت چپ قرمز تر است آن‌هم به دلیل سیاست LIFO در پشته و نحوی پیمایش successor ها در getSuccessor است که به ترتیب successorهای بالا، پایین، راست، چپ به پشته اضافه میشوند و در pop کردن ابتدا مسیر چپ استفاده می‌شود. همانطور که مشاهده میکنید در dfs یک مسیر را تا عمق میرویم بدون اینکه هیچ دیدی داشته باشیم که به سمت هدف میرویم یا نه (ناآگاهانه) و اگر نباشد مجبور به بازگشت هستیم.(هزینه در نظر نمیگیرد). پس این الگوریتم کامل است اما بهینه نیست. این الگوریتم حالاتی که گسترش میدهیم نسبت به بقیه کمتر است اما هزینه بیشتری دارد.

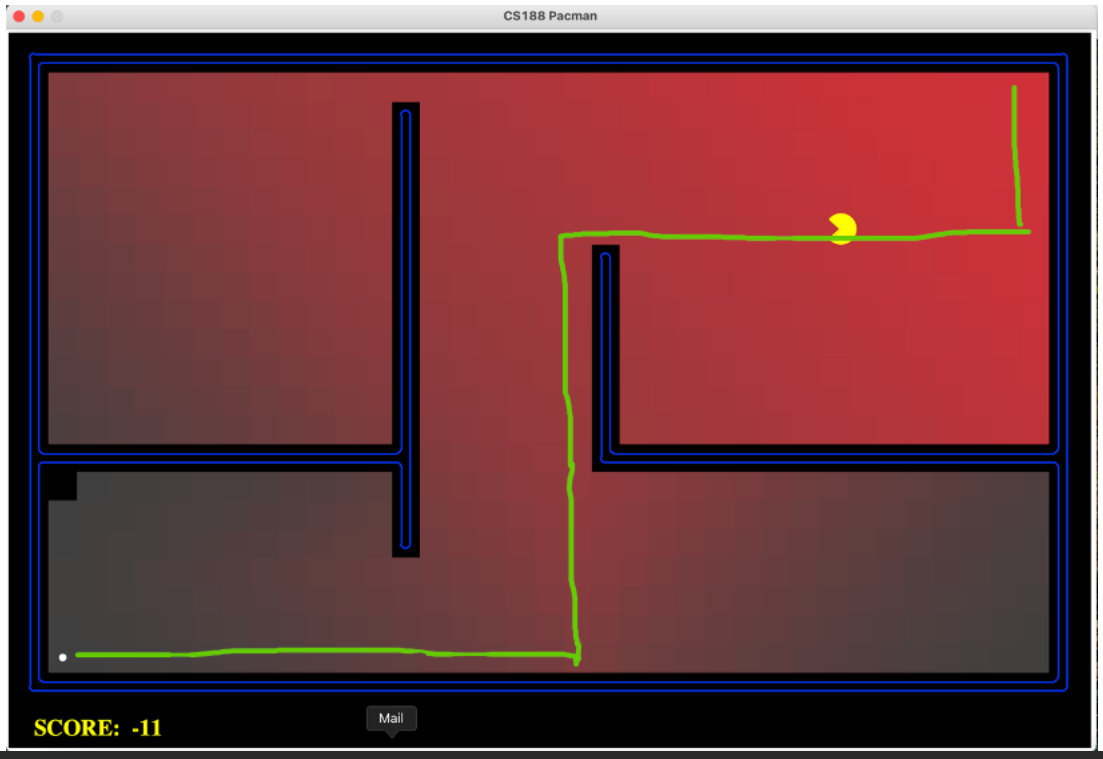
- **الگوریتم BFS**




```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.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
```

در الگوریتم bfs ابتدا تمامی حالاتی که داری یک عمق هستند بررسی میشوند و چون از queue استفاده میشود آنهایی که دارای یک عمق کمتری هستند مورد بررسی بیشتری قرار میگیرند به همین دلیل است که سمت راست شکل پررنگ تر است. همچنین میسر پیدا شده سریعتر است چون حالت های بیشتری را مورد بررسی قرار میدهد و چون از مسیر با عمق کمتر به عمق بیشتر میرود هزینه کمتری مصرف میکند. این الگوریتم نیز ناآگاهانه است و هزینه را اگر هزینه successor ها متفاوت باشد در نظر نمیگیرد.(در این جا هزینه تمامی successor ها ۱ فرض شده است)
بنابراین نسبت به dfs هزینه کمتری دارد اما حالت های بیشتری گسترش داده است.
این الگوریتم کامل است اما بهینه نیست (در شرایطی تفاوت هزینه ها)

- **الگوریتم UCS**

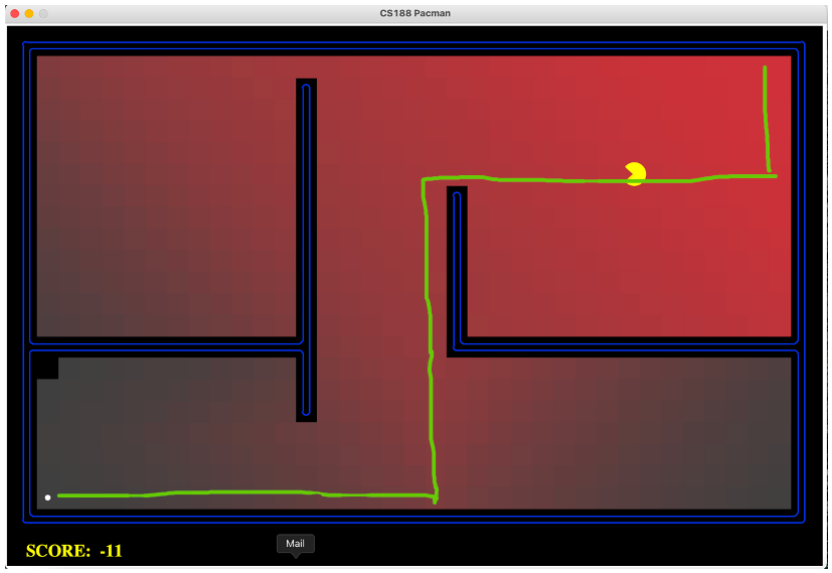



```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l openMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.1 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
```

این الگوریتم در اینجا شبیه به bfs است.
چون در bfs نیز هزینه را یک در نظر گرفتیم و اینجا نیز هزینه تمامی حالات را ۱ در نظر گرفتیم اما اگر هزینه ها متفاوت بود الگوریتم bfs دیگر مسیر خوبی به ما پیشنهاد نمی‌داد. و الگوریتم ucs با در نظر گرفتن آنکه کی هزینه کمتری دارد آن مسیر را ابتدا پیمایش میکند اما همچنان نسبت به هدف دیدی ندارد (نا آگاهانه)
این الگوریتم نیز کامل است اما بهینه نیست (چون دیدی نسبت به هدف ندارد).

- **الگوریتم A***

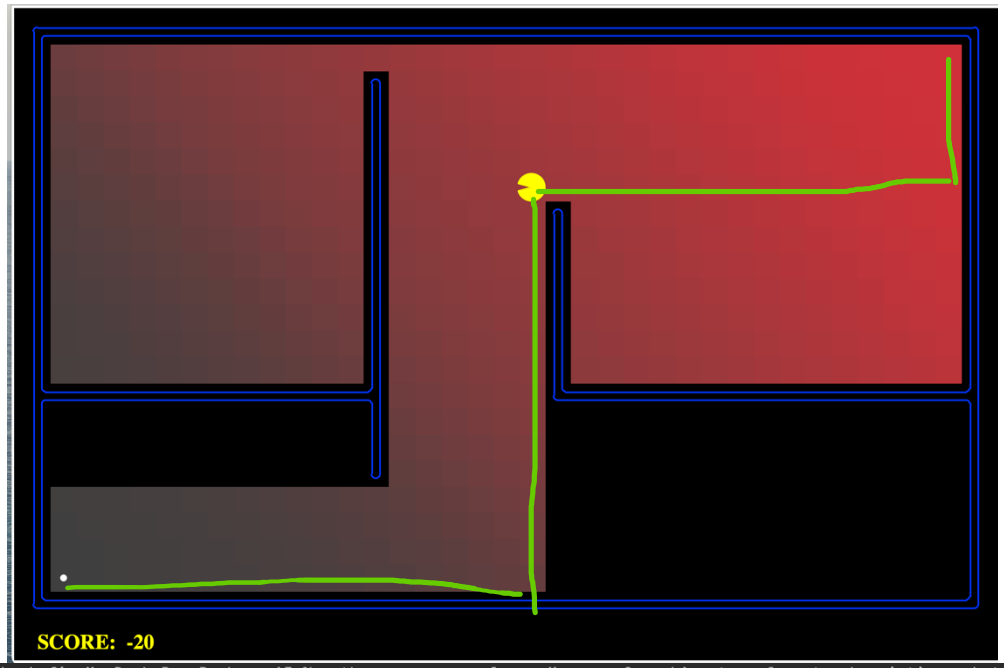
با nullHeuristic :




```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l openMaze -p SearchAgent -a fn=astar
[SearchAgent] using function astar and heuristic nullHeuristic
[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
```

طبق فرمول $F(n) = g(n) + h(n)$ که اگر nullHeuristic در نظر بگیریم مقدار $h(n)=0$ است و رابطه به $F(n) = g(n)$ تبدیل میشود و با الگوریتم ucs و bfsبرابری میکند. اما این تفاوتی ایجاد نمیکند چون ما هنوز مشکل ناآگاهانه بودن نسبت به هدف را به دلیل صفر بودن heuristic داریم .
اگر heuristic را با ManhattanDistance قرار دهیم مسئله را relax کردیم و علاوه بر هزینه یک دیدی نسبت به موقعیت قرار گیری هدف داریم (آگاهانه).
این روش کامل و بهینه است.

با ManhattanDistance heuristic:




```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l openMaze -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
Ending graphics raised an exception: 0
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
```

همانطور که در شکل مشاهده میکنیم استفاده از manhattanHeuristic موجب شده است دیدی نسبت به هدف داشته باشیم و یک سری ناحیه ها را در نظر نگیریم(ناحیه مشکی) ، در واقع گسترش حالات کمتری داریم همچنین باعث شده مسیر پیمایش شده هزینه کمتری شامل شود و دیگر نواحی مشکی رنگ را پیمایش نمیکند.

برای مقایسه کلی :

هزینه : astar(manhattandistance) < bfs,ucs,astar(null heuristic) < dfs

نود های گسترش داده شده : astar(manhattanDis) < dfs < bfs,ucs,astar(null heuristic)

۵) پیدا کردن همه گوشه ها

مطابق راهنمایی سوال برای حالت tinyCorner هزینه برابر ۲۸ شد و گره های گسترش داده شده برای برای mediumCorner حدودا برابر ۲۰۰۰ تاست.

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python 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
rojina@Rojinakashefis-MacBook-Pro Packman AI % python 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 0.2 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
```

اگر از *a و هیوریستیک استفاده کنیم ، میتوان مقدار حالات گسترش داده را کاهش داد.

۶)هیوریستیک برای مسئله گوشه‌ها

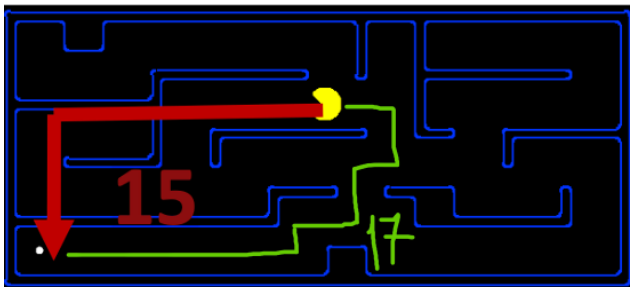
سوال) هیوریستیک خود را توضیح دهید و سازگاری آن را استدلال کنید

هیوریستیک در نظر گرفته شده برابر است با :

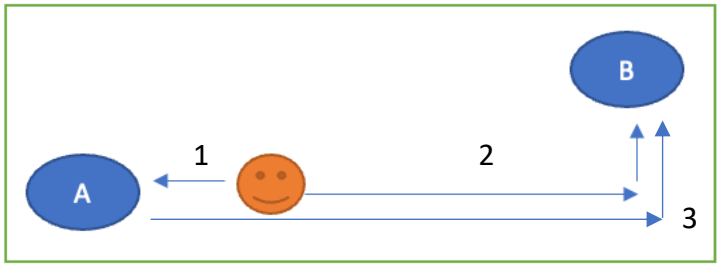
ماکسیمم فاصله منهتن از موقعیت فعلی پک من تا گوشه های بررسی نشده

این هیوریستیک غیربدیهی است زیرا مسئله را ریلکس کرده و از وجود دیوار ها چشم پوشی کرده و درون بازه $0 < h(n) < h^*(n)$ قرار دارد. اگر صفر و $h^*(n)$ باشد یعنی بدیهی شده است و در حالت صفر همان ucs است و موجب ناآگاهی و اگر $h^*(n)$ باشد یعنی زمان محاسبه طولانی است.
فاصله منهتن کوتاه ترین فاصله بین مسیر است که با جهت حرکت پک‌من هم‌خوانی دارد به همین علت از محاسبه زمانی زیادی برخوردار نیست و همینطور صفر نیست و بنابراین غیربدیهی است.

در مثال زیر دیوار ها در نظر گرفته نشده و فاصله منهتن به عنوان heuristic گرفته شده که از هزینه واقعی کمتر است.



برای سازگاری نیز میدانیم f(n) نباید در طول مسیر کاهش پیدا کند و چون ماکسیمم فاصله منهتنی در حال افزایش است چون پک من به سمت f(n) کمتری حرکت میکند (در الگوریتم a استار هست اونی که f(n) کمتری دارد زودتر از صف اولویت خارج میشود چون هزینه کلی کمتری دارد.) و موجب میشود فاصله منهتنی زیاد شود و همینطور اگر آن که f(n) کمتری دارد زودتر خارج شود یعنی cost کمتری داشته است و پک من به سمت آن حرکت میکند و موجب میشود فاصله پک من از بقیه نقاط بیشتر شود در نتیجه cost نیز افزایش می‌یابد.



به عنوان مثال در اینجا ماکسیمم منهتن برابر میسر ۲ است . و همچنین هزینه مسیر دو بیشتر است پس محاسبه a استار به روش زیر میشود :

F(a) = g(a) + manhattan(2)

F(b) = g(b)+ Manhattan(2)

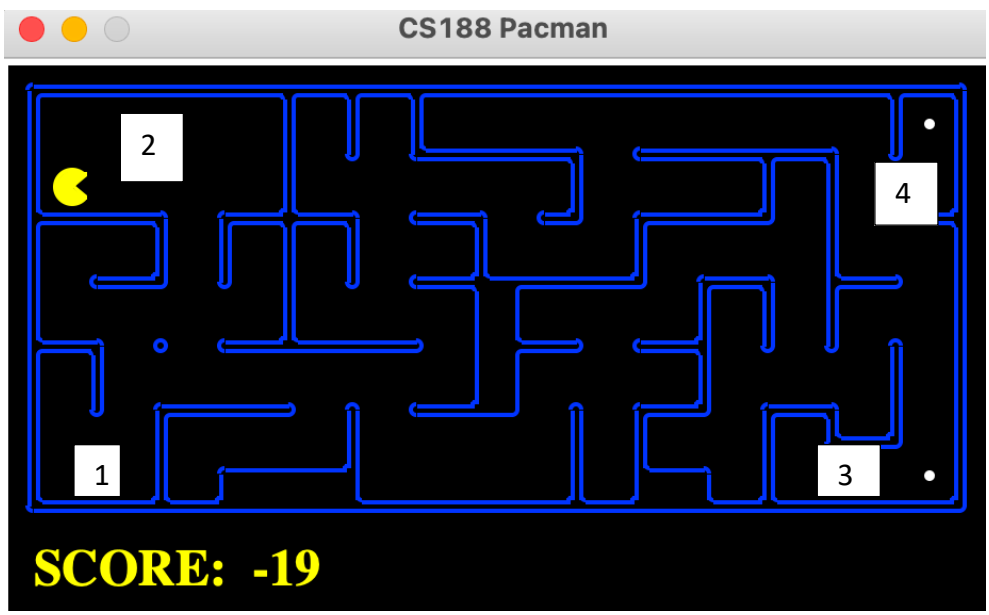
چون f(a) کمتر است سریعتر بیرون میآید و ابتدا به آن سمت میرود.

در حالت بعد برای رسیدن به b هم فاصله منهتن هم هزینه مسیر بیشتر شده است در نتیجه F (b) بیشتر شده است و f(n) افزایش پیدا کرده است.

در صورت سوال گفته شده گر usc و *A مسیر هایی با طول برابر بدهند هیوریسیک سازگار است که مشاهده میکنیم برابر هستند:

```
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l mediumCorners -p SearchAgent -a fn=ucs,prob=CornersProblem,heuristic=cornersHeuristic
[SearchAgent] using function ucs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.3 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
rojina@Rojinakashefis-MacBook-Pro Packman AI % python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 1136
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

مسیر پیمایش شده به صورت زیر است:



ترتیب فاصله منهتن : 4> 3> 2 > 1
ترتیب هزینه بیشتر : 4 > 3> 2 > 1
ترتیب f(n) : 4> 3 > 2 > 1
ترتیب خروج از صف اولویت : ۱، ۲، ۳، ۴ (اونی که f(n) کمتری دارد زودتر خارج میشود)
همانطور که مشاهده میکنید f(n) از ۱ تا ۴ در حال افزایش است چون هم منهتن هم هزینه بیشتری میشود و در صف الویت با الویت پایین تر برای خارج شدن قرار میگیرد، بنابراین سازگار است.

۷) خوردن همه نقطه‌ها

سوال) هیوریستیک خود را توضیح دهید و سازگاری آن را استدلال کنید

هیوریستیک در نظر گرفته شده برابر است با :

ماکسیمم mazeDistance از موقعیت فعلی پک من تا نقاط خورده نشده

mazeDistance فاصله بین دو نقطه را با استفاده از bfs که اگر هزینه های مسیر ۱ باشد بهینه است حساب میکند.
غیربدیهی است زیرا با درنظر گرفتن دیوار ها جستجوی bfs را ریلکس کرده و حالات جستجو را کاهش داده است و موجب شده از h*(n) کمتر شود و چون فاصله بین دو نقطه صفر نمیشود پس ماکسیمم مقدار mazeDistance که همان هیوریستیک است صفر نیست بنابراین رابطه 0<h(n)<h*(n) برقرار است و غیربدیهی است.

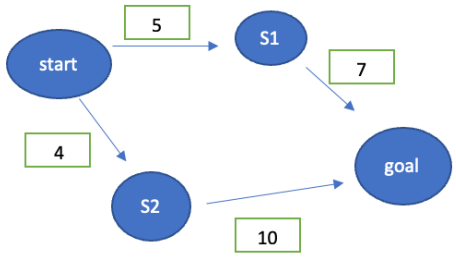
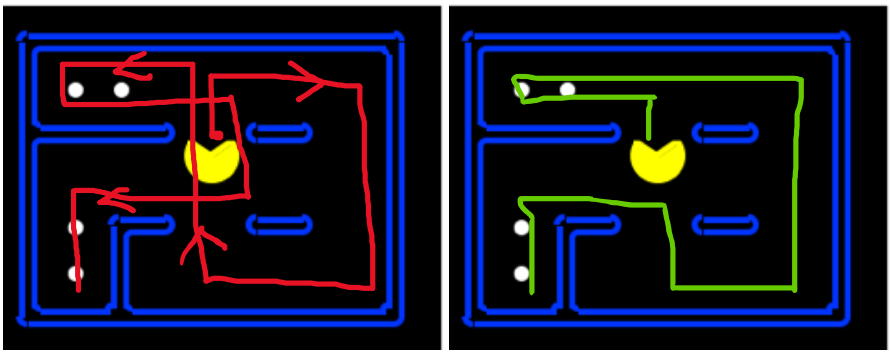
سازگار است به همان علت هیوریستیک manhattanDistance .
چون pacman به آن که f(n) کمتری دارد حرکت میکند موجب میشود هم فاصله منهتن هم cost در طول مسیر زیاد شود و باعث افزایش f(n) در طول مسیر شود.
بین هزینه های بدست آمده بزرگترین آخر از همه از صف اولویت خارج شود و این موجب میشود در طول مسیر کاهش f(n) نداشته باشیم و به مرور افزایش f(n) را محاسبه کنیم.

همچنین در صورت سوال گفته شده که اگر سازگار باشد برای حالت mediumSearch در زمان طولانی حل میکند که اگر دستور را اجرا کنیم میبینیم که صفحه قفل میشود بنابراین سازگار است.

۸)جستجوی نیمه بهینه

سوال) closestDotSearchAgent شما، همیشه کوتاه‌ترین مسیر ممکن در ماز را پیدا نخواهد کرد.مطمئن شوید که دلیل آن را درک کرده‌اید و سعی کنید یک مثال کوچک بیاورید که در آن رفتن مکرر به نزدیکترین نقطه منجر به یافتن کوتاهترین مسیر برای خوردن تمام نقاط نمی‌شود.

چون هزینه همه ۱ است الگوریتم bfs و ucs و astar میشود استفاده کرد.
از bfs استفاده شده چون سریعتر است.
چون در greedy نسبت به لحظه عمل میکنیم و آینده نمیدانیم چه میشود منجبر به کوتاه ترین مسیر نمیشود.
به عنوان مثال)
چون ۴ کمتر از ۵ است به سمت شاخه پایین میرود ولی در آینده مسیری با طول ۱۰ داریم که هزینه را زیاد میکند و چون در greedy دیدی نسبت به آینده و در لحظه نگاه میکنیم مسیر بهینه انتخاب نمیکنیم.



یا اگر بخواهیم مثالی از pacman بزنیم در شکل سمت چپ مشاهده میکنیم مسیر پیمایش شده مسیر تکراری زیاد دارد و موجب افزایش هزینه میشود(۳۱) در صورتی که اگر از الگوریتم a استار استفاده کنیم میسر سمت راست نتیجه میشود و بهینه است.