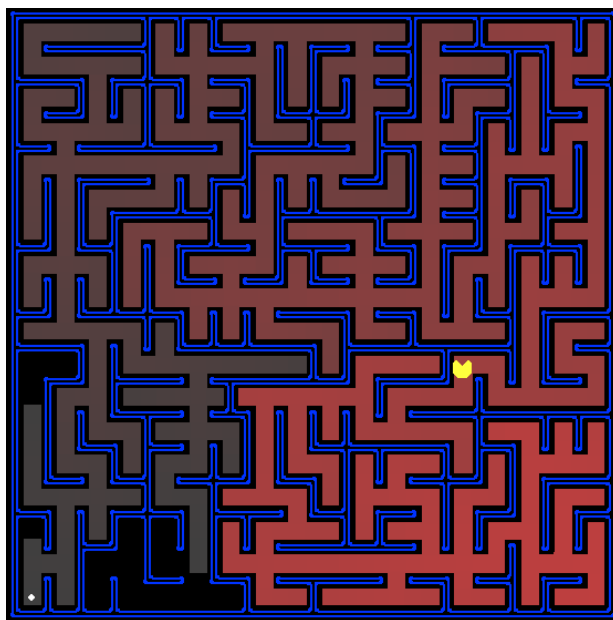


بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

تمرین سری اول برنامه نویسی : سرچ در پکمن



معرفی:

در این پروژه ، عامل پکمن شما از بین جهان هزارتوی خود هم برای رسیدن به مکانی خاص و هم برای جمع آوری بهینه غذا ها مسیری پیدا خواهد کرد . شما باید الگوریتم های عمومی جست و جو را پیاده سازی و آن ها را به جهان پکمن اعمال کنید. این پروژه شامل یک سیستم نمره دهی خودکار است تا بتوانید بعد از اتمام کد خود نمره خود را مشاهده کنید. این عمل با دستور زیر قابل اجراست :

```
python autograder.py
```

نکته : اگر در ترمینال **pycharm** کد میزنید در تمامی دستور های گفته شده کلمه **python** اول دستور را حذف کنید

کد این پروژه از چندین فایل پایتون تشکیل شده است ، بعضی از آن ها را باید بخوانید و متوجه شوید و از بعضی از آن ها میتوانید چشم پوشی کنید. شما میتوانید همه ی کد ها و فایل های کمکی را به صورت یک فایل زیپ از [اینجا](#) دانلود کنید.



بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

فایل هایی که باید ویرایش کنید :	
search.py	همه ی الگوریتم های جست و جوی شما اینجا خواهند بود
searchAgents.py	همه ی عامل های جست و جوی شما اینجا خواهند بود
فایل هایی که باید بخوانید ولی ویرایش نکنید :	
pacman.py	فایل اصلی که بازی پکمن را اجرا میکند.
game.py	قانونی که پشت نحوه کار دنیای پکمن است.
util.py	ساختار داده ای های مفید برای پیاده سازی الگوریتم های جست و جو
فایل های کمکی که میتوانید از خواندن آن ها چشم پوشی کنید :	
graphicsDisplay.py	گرافیک پکمن
graphicsUtils.py	فایل کمکی برای گرافیک پکمن
textDisplay.py	گرافیک ASCII برای پکمن
ghostAgents.py	عامل هایی برای کنترل روح ها
keyboardAgents.py	واسط کیبورد برای کنترل پکمن
layout.py	کدهایی برای خواندن فایل های لایوت و ذخیره ی محتوای آن ها
autograder.py	سیستم نمره دهی خودکار پروژه
testParser.py	تجزیه آزمون نمره دهی خودکار و فایل های راه حل
testClasses.py	آزمون نمره دهی عمومی کلاس ها
test_cases/	مسیری که شامل موارد آزمون برای هر سوال است
searchTestClasses.py	آزمون نمره دهی کلاس های مخصوص پروژه اول

فایل هایی که باید ویرایش و ارسال کنید : شما بخش هایی از [search.py](#) و [searchAgents.py](#) را که مشخص شده است در طول این تمرین پر خواهید کرد . لطفا سایر قسمت ها را دست نخورده باقی بگذارید. پس از تکمیل این دو فایل آن ها را زیپ کرده و نام فایل فشرده خود را هم نام با شماره دانشجویی خود قرار دهید و آن را ارسال کنید.

ارزیابی : کد شما برای ارزیابی صحت فنی توسط سیستم نمره دهی خودکار بررسی خواهد شد. لطفا نام هیچ کدام از توابع و یا کلاس های داخل کد ها را تغییر ندهید. در غیر این صورت سیستم نمره دهی خودکار کد شما را رد خواهد کرد. طبق نمرات حاصل شده از سیستم نمره دهی خودکار نمره ی سوال ۵ شما به عنوان نمره ی سوال ۸ درج شده و سوال ۵ و ۶ و ۷ این سیستم برای شما حذف شده است. با این حال نمره ی حاصله از سیستم نمره دهی خودکار ، نمره نهایی شما نخواهد بود و صحت پیاده سازی کد ها پایه و اساس نمره دهیست.



بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

به دنیای پکمن خوش آمدید

بعد از دانلود کد ([search.zip](#)) خارج کردن آن از حالت فشرده و باز کردن آن با نرم افزار pycharm شما باید بتوانید پکمن را با تایپ کردن دستور زیر در ترمینال pycharm اجرا کنید :

```
python pacman.py
```

پکمن در دنیایی شمال راهرو های تو در تو و میوه های خوش مزه زندگی میکند. هدایت کارآمد پکمن در این جهان قدم اول در تسلط به آن خواهد بود.

ساده ترین عامل در searchAgents.py عامل GoWestAgent است که همیشه به سمت غرب میرود. این عامل هر از گاهی میتواند برنده شود :

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

اما وقتی نیاز به چرخش باشد اوضاع برای این عامل سخت میشود :

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

اگر پکمن جایی گیر کرد شما میتوانید با تایپ کردن CTRL-C در ترمینال خود از بازی خارج شوید.

به زودی عامل شما قادر خواهد بود نه تنها tinyMaze بلکه هر مازی را حل کند.

نکته اینکه pacman.py از چندین گزینه پشتیبانی میکند که هر کدام میتوانند به روش طولانی (e.g., --layout) و کوتاه

(e.g., -l) بیان شوند. شما میتوانید لیست همه ی این گزینه ها و مقادیر پیش فرضشان را با دستور زیر مشاهده کنید :

```
python pacman.py -h
```

همچنین ، همه ی دستوراتی که در این پروژه به کار گرفته میشوند در فایل commands.txt موجودند که میتوانید به راحتی آن ها را کپی و پیست کنید.



بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

سوال ۱: پیدا کردن یک نقطه ثابت غذا با استفاده از DFS

در فایل `searchAgents.py` شما یک `SearchAgent` که به طور کامل پیاده سازی شده را پیدا خواهید کرد که مسیری در دنیای پکمن پیدا میکند و سپس قدم به قدم آن را طی میکند اما الگوریتم های جست و جو برای کوتاه کردن یک مسیر پیاده سازی نشده اند که این کار شماست.

ابتدا با دستور زیر امتحان کنید که آیا `SearchAgent` به درستی کار میکند یا خیر :

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

دستور بالا به `SearchAgent` میگوید که از `tinyMazeSearch` به عنوان الگوریتم جست و جوی خود استفاده کند که در فایل `search.py` پیاده سازی شده است. پکمن باید با موفقیت در هزار تو حرکت کند. حالا زمان آن رسیده که تابع عمومی جست و جوی تکامل یافته را برای کمک به مسیر یابی پکمن بنویسید! الگوریتم کلی جست و جو برای درس مطابق زیر است :

```
function GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe, strategy)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
      end
    end
  end
```

نکته مهم : به یاد داشته باشید که یک گره جست و جو باید علاوه بر حالت خود اطلاعات لازم برای بازسازی مسیری که به آن حالت ختم میشود را داشته باشد.

نکته مهم : همه ی توابع جست و جوی شما باید لیستی از اعمالی که عامل را از نقطه ی شروع به نقطه ی هدف میرسانند برگردانند. این اعمال همگی باید حرکات قانونی باشند (جهت دهی های صحیح نه حرکت از بین دیوار ها)



بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

نکته مهم : حتما از ساختار داده های Queue ، Stack و PriorityQueue که در فایل util.py برای شما آماده شده است استفاده کنید! این ساختار داده های پیاده سازی شده مشخصه های ویژه ای دارند که جهت سازگاری با سیستم نمره دهی خودکار نیاز اند.

راهنمایی : الگوریتم های مربوط به DFS, BFS, UCS و A* خیلی به هم شبیه اند و از شبه کد الگوریتم جست و جوی کلی بالا مشتق شده اند. پس فقط روی پیاده سازی صحیح DFS تمرکز کنید و بعد از آن قادر خواهید بود سایر توابع را به سادگی بنویسید.

الگوریتم (DFS) depth-first search را در تابع depthFirstSearch موجود در فایل search.py پیاده سازی کنید. برای کامل کردن الگوریتم خود نسخه ی جست و جوی گرافی (graph search) این الگوریتم را بنویسید که مانع از بسط گره هایی میشود که قبلا مشاهده شده اند .

کد شما باید به سرعت راه حلی برای دستور های زیر پیدا کند :

```
python pacman.py -l tinyMaze -p SearchAgent
python pacman.py -l mediumMaze -p SearchAgent
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

صفحه ی پکمن گره های بسط داده شده و ترتیب بسط دادن آن ها را نشان خواهد داد (قرمز روشن تر به معنی بسط دادن زودتر آن گره است). آیا ترتیب بسط گره ها همانی بود که انتظار داشتید ؟ آیا پکمن واقعا به تمام گره های بسط داده شده در مسیرش تا هدف میرود ؟

راهنمایی : اگر شما از Stack به عنوان ساختار داده ای خود استفاده کردید ، راه حل پیدا شده با الگوریتم DFS برای mediumMaze باید طولی برابر با ۱۳۰ داشته باشد. آیا این کم هزینه ترین راه حل است ؟ اگر نه درباره اینکه چرا جست و جوی DFS جواب بهینه را پیدا نمیکند فکر کنید.



بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

سوال ۲: پیدا کردن یک نقطه ثابت غذا با استفاده از BFS

الگوریتم (BFS) `breadth-first search` را در تابع `breadthFirstSearch` موجود در فایل `search.py` پیاده سازی کنید. مجدد الگوریتم گراف سرچ آن را بنویسید که بسط گره های دیده شده خودداری میکند. کد خود را به همان روشی که در سوال قبل بود تست کنید.

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```

آیا BFS کم هزینه ترین راه حل را پیدا میکند؟ اگر نه باید پیاده سازی خود را مجدداً چک کنید.

راهنمایی: اگر پکمن شما خیلی آرام حرکت میکند گزینه ی `--frameTime 0` را امتحان کنید.

نکته: اگر شما کد جست و جوی خود را به صورت کلی نوشته اید، کد شما باید برای مساله ی جست و جوی `eight-puzzle` هم به خوبی کار کند و نیازی به هیچگونه تغییر نخواهد داشت.

```
python eightpuzzle.py
```



بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

سوال ۳: تغییر تابع هزینه

در حالی که BFS کوتاه ترین مسیر را تا هدف به دست می آورد ، ممکن است ما به دنبال پیدا کردن مسیر هایی باشیم که از جنبه های دیگر بهترین اند. ما های `mediumDottedMaze` و `mediumScaryMaze` را در نظر بگیرید.

با عوض کردن تابع هزینه ما میتوانیم پکمن را ترغیب کنیم تا مسیر های متفاوتی را پیدا کند. برای مثال ، ما میتوانیم هزینه ی بیشتری برای گام های خطرناک در مناطقی که روح ها حرکت میکنند و یا هزینه ی کمتر برای گام هایی در مناطق نزدیک به غذا ایجاد کنیم و یک عامل هوشمند پکمن باید رفتار خود را در پاسخ به آن تنظیم کند.

الگوریتم گراف جست و جوی UCS را در تابع `uniformCostSearch` موجود در فایل `search.py` پیاده سازی کنید . ما به شما توصیه میکنیم تا با نگاه به فایل `util.py` به دنبال ساختار داده ای بگردید که در کدتان موثر خواهد بود. اکنون شما باید رفتاری موفق را در هر سه لایه مشاهده کنید، جایی که عامل تمامی آن ها عامل UCS است که تنها در تابع هزینه ای که استفاده میکنند متفاوت اند. (عامل ها و توابع هزینه برای شما نوشته شده اند) :

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

نکته : شما باید مسیر های خیلی کم هزینه و بسیار پر هزینه ای را به ترتیب به ازای `StayEastSearchAgent` و

`StayWestSearchAgent` طی توابع هزینه ی نمایی آن ها به دست آورید (برای جزئیات بیشتر فایل `searchAgents.py` را

بینید)



بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

سوال ۴: جست و جوی A*

جست و جوی گراف A* را در تابع خالی aStarSearch در فایل search.py پیاده سازی کنید. A* یک تابع هیوریستیک را به عنوان ورودی خود میگیرد. توابع هیوریستیک دو ورودی میگیرند: یک حالت در مساله ی جست و جو (آرگومان اصلی) و خود مساله (به عنوان مرجع اطلاعات). تابع هیوریستیک nullHeuristic در فایل search.py یک مثال جزئی از این نوع توابع است.

شما میتوانید پیاده سازی A* خود را روی مساله ی اصلی یافتن مسیری در بین هزارتو تا یک موقعیت ثابت با استفاده از هیوریستیک فاصله های منهتن (Manhattan distance heuristic) که با عنوان manhattanHeuristic در فایل searchAgents.py موجود است، امتحان کنید.

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a  
fn=astar,heuristic=manhattanHeuristic
```

پس از اجرای آن باید مشاهده کنید که A* راه حل بهینه را کمی سریع تر از UCS پیدا میکند. برای استراتژی های جست و جوی گوناگون چه اتفاقی روی openMaze می افتد؟



بسم الله الرحمن الرحيم

تمرین برنامه نویسی سری اول هوش مصنوعی

پاییز ۱۳۹۶

سوال ۵: جست و جوی نیمه بهینه

گاهی اوقات، حتی با A^* و یک هیوریستیک خوب، یافتن مسیر بهینه در بین همه نقاط سخت است. در این موارد، ما هنوز به دنبال یافتن مسیری خوب و منطقی هستیم. در این بخش شما عاملی خواهید نوشت که همیشه حریصانه نزدیک ترین نقطه به خود را بخورد. `ClosestDotSearchAgent` برای شما در فایل `searchAgents.py` پیاده سازی شده است اما یک تابع کلیدی برای پیدا کردن مسیری به نزدیک ترین نقطه در آن جا مانده است.

تابع `findPathToClosestDot` را در فایل `searchAgents.py` پیاده سازی کنید. عامل ما این هزارتو را به صورت نیمه بهینه زیر یک ثانیه با مسیری به هزینه ی ۳۵۰ حل خواهد کرد:

```
python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
```

راهنمایی: سریع ترین راه برای کامل کردن `findPathToClosestDot` این است که `AnyFoodSearchProblem` را کامل کنید که هدف آن جا افتاده است. سپس مساله را با یک تابع مناسب جست و جو حل کنید. جواب کوتاه خواهد بود `ClosestDotSearchAgent` همیشه کوتاه ترین مسیر ممکن در هزارتو را پیدا نخواهد کرد. بررسی کنید که چرا این اتفاق می افتد و مثال ساده ای از جایی که خوردن نزدیک ترین نقاط به کوتاه ترین مسیر ختم نمیشود ارائه کنید.

* تمرینها باید از طریق ایمیل تحویل داده شود. موضوع ایمیل ارسالی حتما به فرم مثال زیر باشد در غیر اینصورت ۲۰

درصد نمره تمرین کسر خواهد شد:

<StudentNo>-<FisrtName>-<LastName>-HW<No>-Kharazmi-AI-Fall96
83405307-Hadi-Asheri-HW1-Kharazmi-AI-Fall96

* تمرینها به آدرس `ai.kharazmi.fall96@gmail.com` ارسال شود.

* مهلت ارسال: ۷/۲۹ - ساعت ۱۸