# گزارش پروژه اول هوش مصنوعی


## امیررضا نوری ۹۷۲۶۰۸۳

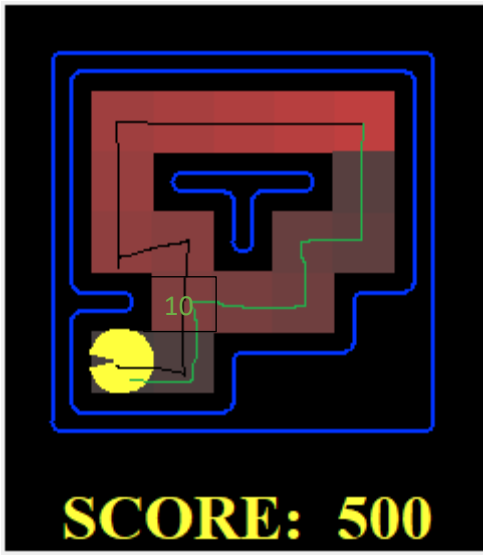

**پاسخ سوال ۱ =**

بله ترتیب کاوش مورد انتظار بود. از آنجایی که ترتیب پوش شدن successor ها در تابع getSuccessor شمال-جنوب-شرق-غرب است و روش پیاده سازی آن Stack (LIFO) است هنگام پاپ شدن آن ها از آخرین successor که وارد شده شروع به پاپ شدن میکند و در واقع ترتیب پاپ شدن غرب-شرق-جنوب-شمال است و طبق این ترتیب رفتار پکمن مورد انتظار و درست بود. در اولین نود انتخاب بین غرب و جنوب است که به دلایل گفته شده شرق انتخاب میشود و در خانه شماره ۱۰ انتخاب بین جنوب و شرق است که ابتدا شرق انتخاب میشود و پس از اینکه به هدف نمیرسد و به نود تکراری می رسد، جنوب در آن خانه انتخاب می شود و جستجو ادامه می یابد.

خیر پکن به تمام مربع های کاوش شده نمیرود بلکه اولین مسیری که به هدف منتهی می شود را انتخاب میکند و فقط در خانه های ان مسیر می رود. مسیر پکمن در تصویر زیر با خط مشکی نشان داده شده است.

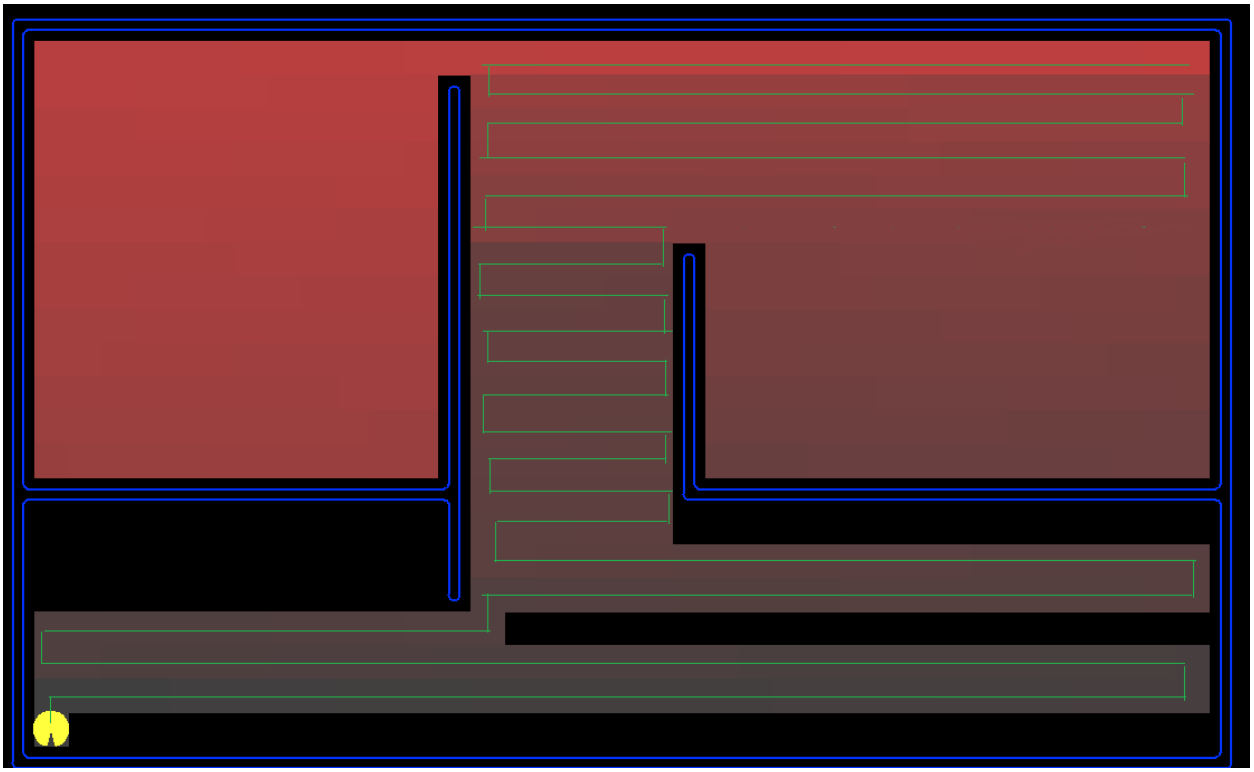
**پاسخ سوال ۲-**

خیر زیرا مسیری که باز گردادنده میشود لزوما کمترین هزینه را ندارد برای مثال در ماز زیر مسیر با کمترین هزینه با رنگ سبز نشان داده شده است که هزینه ان۸ خانه است ولی مسیر برگردانده شده ۱۰ خانه را باید طی کند و مسیر هینه نیست. علت این است که جستجوی DFS ناآگاهانه است و بدون در نظر گرفتن موقعیت دیوار ها و هدف یک مسیر را پیش میگید و تا عمق میرود تا به هدف برسد و اولین مسیر منتهی به هدف را بر میگرداند و این مسیر الزاما بهینه ترین مسیر نیست.



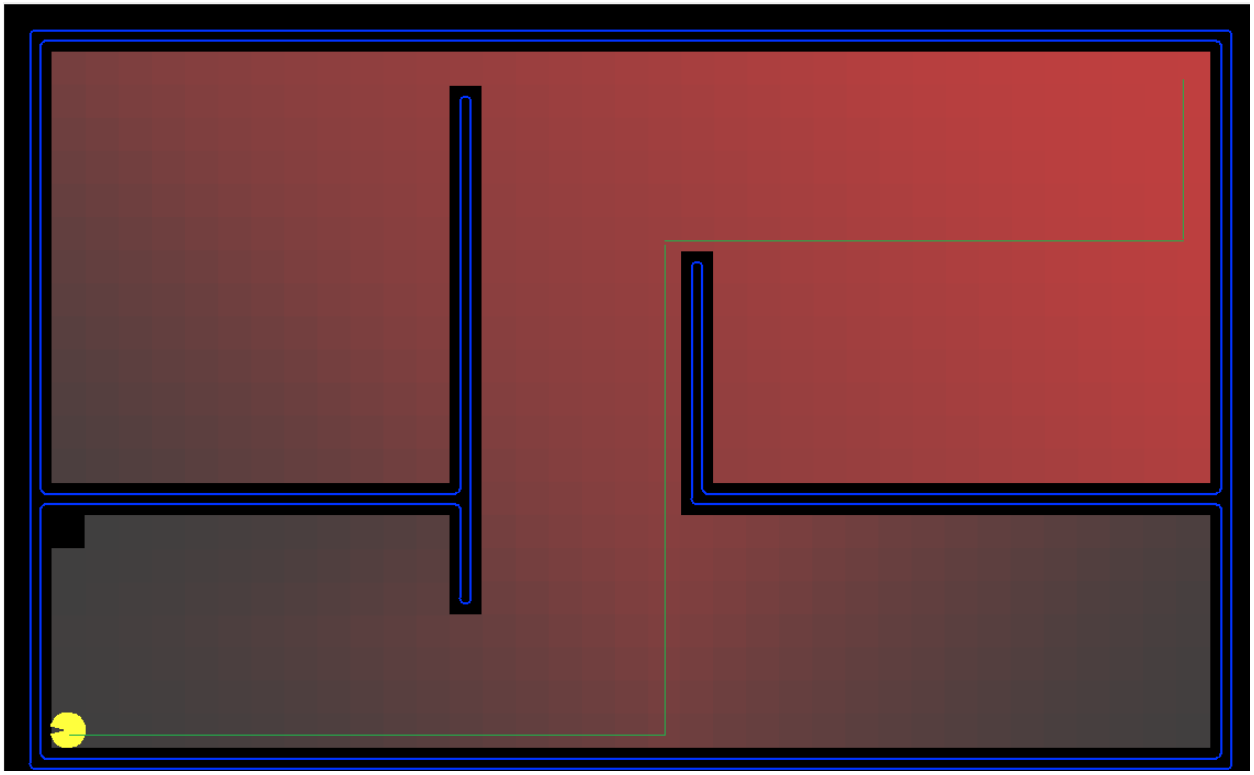
## پاسخ سوال ۳-

**نحوه رفتار عامل با جستجوی DFS در openMaze:**



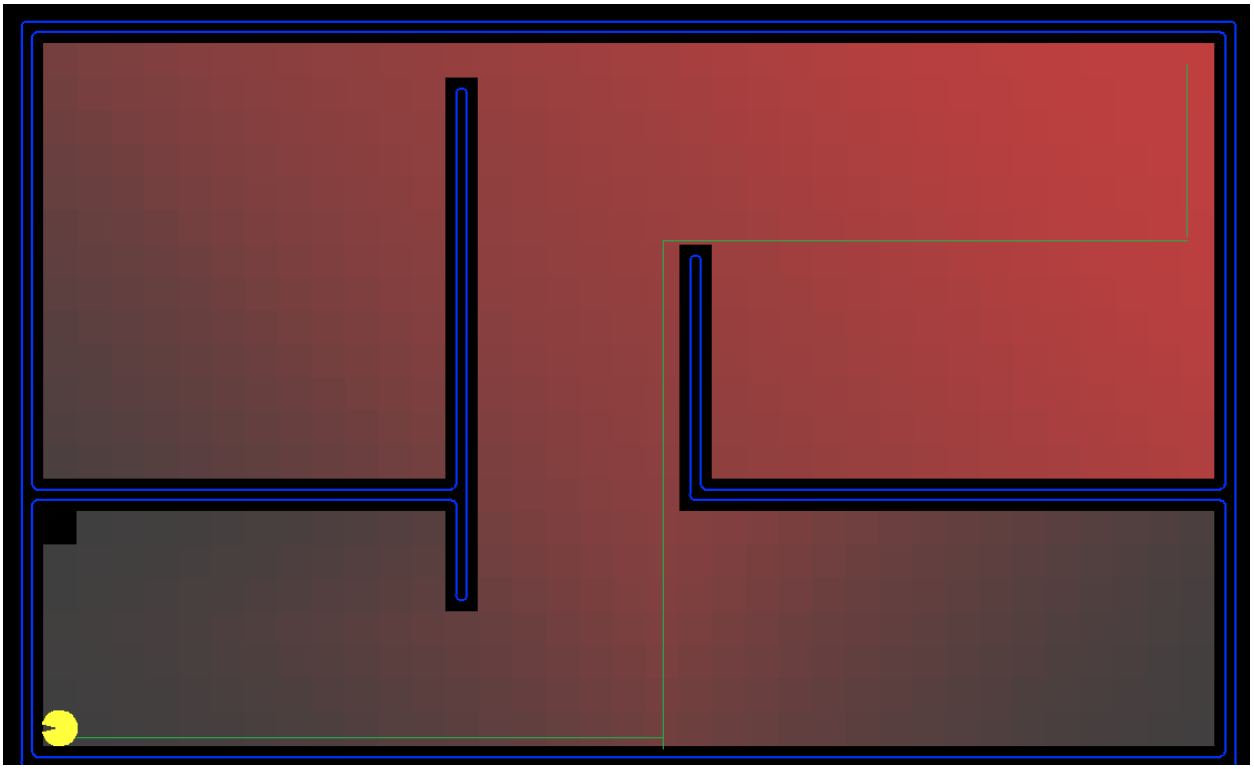
**مسیر پیموده شده با خط سبز نشان داده شده است. با توجه به نحوه پیاده سازی فرنج به صورت stack است ورودی ها (پوش) به ترتیب شمال-جنوب-شرق-غرب است ولی خوانده شدن از فرینج از اخر به اول غرب-شرق-جنوب-شمال است (LIFO). به همین علت است که نود های سمت غرب نقشه زودتر کاوش شده است و همچنین قسمت های کاوش نشده که با مشکی نشان داده شده اند به این معنی است که نود هایی با عمق کمتر یا برابر اولین نود هدف پیدا شده در جستجو دارند.**

**نحوه رفتار عامل با جستجوی BFS در openMaze:**



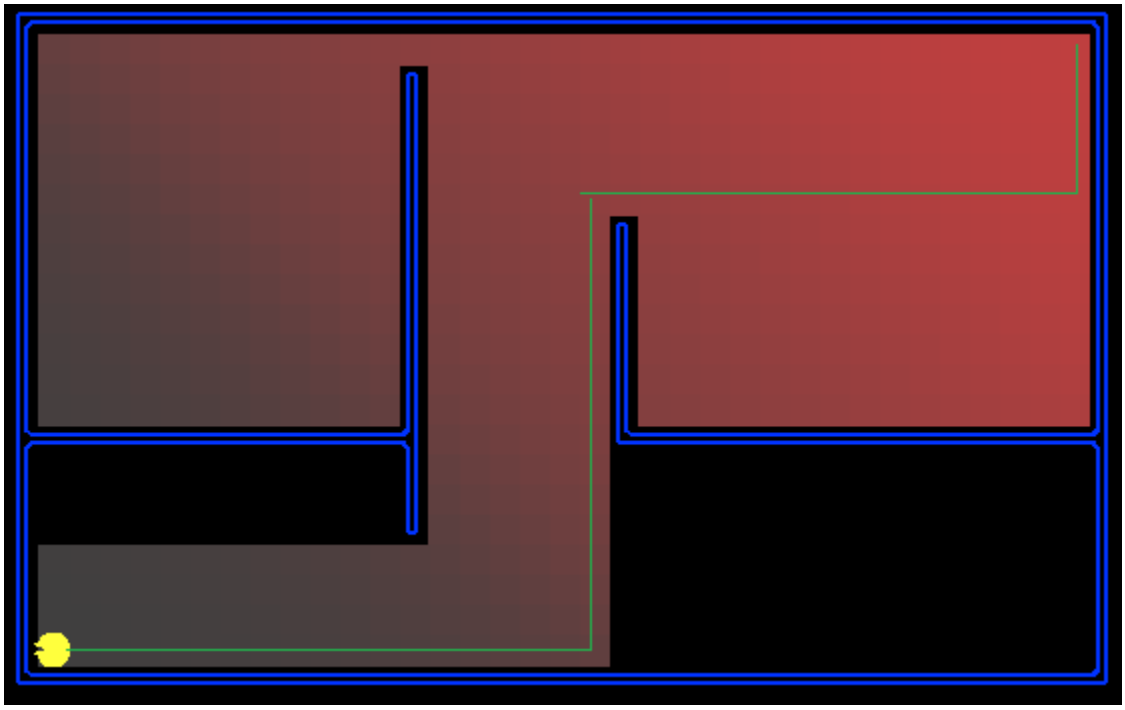
**مسیر پیموده شده با خط سبز نشان داده شده است. جستجوی bfs لزوما مسیر بهنیه را بر نمیگرداند ولی در این مثال بهینه ترین مسیر را برگردانده است. در این نوع جستجو از فرینج Queue که نحوه وارد شدن و خواندن از آن به صورت FIFO است استفاده شده و هم ورودی و هم خروجی به ترتیب برابر شمال-جنوب-شرق-غرب، است. در این نوع جستجو همانطور که در تصویر بالا مشاده میشود نود های نزدیک تر به نود شروع زودتر کاوش میشوند و هرچه فاصله از نود شروع بیشتر شود رنگ آن تیره در میشود که به معنی دیرتر کاوش شدن است. یک نقطه اس کل محیط بازی کاوش نمیشود که علت آن این است که در همه مسیر های منتهی به آن عمقی بیشتر یا برابر با کوتاه ترین مسیر رسیدن به نود هدف دارد.**

**نحوه رفتار عامل با جستجوی UCS در openMaze:**



**مسیر پیموده شده با خط سبز نشان داده شده است. این جستجو مانند BFS است دلیل انی امر این است که تابع هزینه برابر هزینه واقعی رسیدن تا نود هدف است و از آنجایی که فرینج با priorityQueue پیاده سازی میشود نود هایی که فاصله شان از مبدا کمتر است زودتر کاوش میشوند و قرمز تر اند و این دلیل شباهت آن با BFS است.**
**جستجتوی UCS میسر بهینه را بر میگرداند.**

**نحوه رفتار عامل با جستجوی Astar در openMaze با هیوریستیک منتهن:**



**مسیر حرکت با خط سبز مشخص شده است. فرینج با priorityQueue پیاده میشود. هیوریستیک در این جستجو فاصله منهتنی است. اولیت در این جستجو جمع فاصله منهتنی مکان فعلی و هدف یا همان هیوریستیک با هزینه هزینه واقعی رسیدن از نقطه شروع تا تا نقطه فعلی است. به همین دلیل نود های نزدیک تر زودتر کاوش میشوند و قرمز تر هستند و از یک جایی به بعد به دلیل افزایش فاصله متنهتی تا هدف آن نقطه ها دیگر کاش نمیشود و این علت سیاه بودن قسمت های سمت راست پایین ماز و بالای هدف است. به بیان بهتر جمع هیورستیک و فاصله واقعی آنها تا نقطه شروع بیشتر از تمام نقطه های کاوش شده در ماز میباشد و اولویت کمتری داشته اند.**

## پاسخ سوال ۴-

استیت شامل مکان فعلی و لیست از تمام گوش های ملاقات نشده است. اگر لیست خالی بود یعنی که تمام گوشه ها ملاقات شده و به هدف رسیده ایم و هیوریستیک ۰ برگردانده میشود. در غیر این صورت به محاسبه هیوریستیک میپردازیم. فرض میکنیم که مکان ما یک نقطه دلخواه در ماز باشد، ابتدا فاصله منهتنی تمام گوشه ها تا مکان فعلی ما محاسبه میشود و کمترین آن انتخاب میشود و اگر قبلا ملاقات نشده بود و به آن گوشه میرویم و از لیست گوشه های ملاقات نشده حذف میشود و مکان فعلی ما آن گوشه میشود و فاصله منهتی طی شده با هیوریستیک قبلی که در ابتدا صفر بود جمع میشود. روند گفته شده دوباره تکرار میشود تا زمانی که لیست گوشه های ملاقات نشده خالی شود و آن زمان به هدف رسیده ایم و هیوریستیک که مجموع تمامی فاصله های منهتی گفته شده است برگردانده میشود.

دلیل admissible بودن: چون از فاصله منهتی برای اندازه گیری فاصله و مسیر رفتن تا گوشه ها استفاده شده و دیوار ها در نظر گرفته نشده است قطعا از مسیر واقعی کوتاه تر یا برابر آن است.

دلیل consistent بودن: از آنجایی که در هر بار ملاقات کردن یک گوشه فاصله منهتی آن با هیوریستیک نود قبلی جمع میشود همیشه مقدار هیوریستیک نود بعدی بیشتر از قبلی است و چون همگی از فاصله منهتنی استفاده میکنند که از فاصله واقعی کمتر یا برابر آن است شرط consistent بودن ارضا میشود.

**پاسخ سوال ۵-**

**در صورتی ک تعداد غذا های باقی مانده صفر باشد یعنی استیت به هدف رسیده و مقدار هیوریستیک ۰ را برمیگرداند در غیر این صورت روند زیر طی میشود. متغیری تحت عنوان check تعریف میکنیم که مکان فعلی و مکان یک به یک غذا هارا جمع میکند، در صورتی که این مقدار در problem.heuristicInfo بود فاصله را برابر آن مقدار قرا میدهد در غیز این صورت فاصله را از طریف فاصله واقعی مکان فعلیو غذا ها محاسبه میکند و آن را در problem.heuristicInfo جایزگین میکند. در صورتی که فاصله از هیوریستیک بیشتر باشد مقدار هیوریستیک برا بر مقدار فاصله میشود. در نهایت وقتی تمام غذا ها چک شد مقدار هیوریستیک برگردانده میشود.**

دلیل admissible بودن: با هیوریستیک گفته شده بهترین حالت مسیر برای خوردن نزدیک ترین و دورترین غذا این است که سایر غذا ها بر روی کوتاه ترین مسیر برای رفتن از نزدیک ترین غذا به حالت فعلی و دورترین غذا قرار گرفته باشند. به همین دلیل مقدار هیوریستیک برگردانده شده همیشه کمتر یا برابر حالت واقعی خواهد بود.

**پاسخ سوال ۶-**

**جستجوی حریصانه فقط در لحظه تصمیم میگید و بلند مدت را نگاه نمیکند برفرض مثال در حالت ساده زیر مسیر بهینه و حریصانه با میبینیم:**

| | | | | | |
|---|---|---|---|---|---|
| • | pacman | • | • | • | • |

حریصانه:۹

| | | | | | |
|---|---|---|---|---|---|
| • | pacman | • | • | • | • |

آ استار:۸