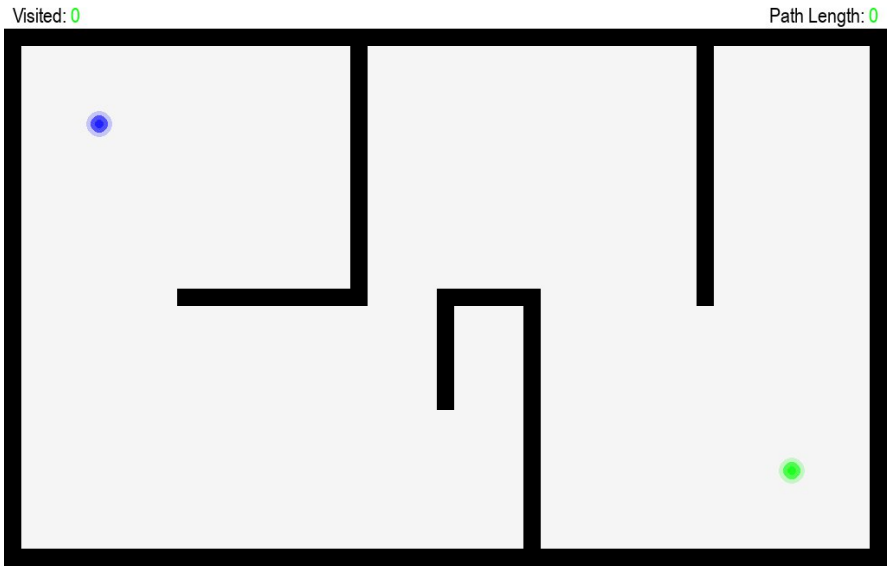
به نام خدا

تمرین کامپیوتری اول
مهلت تحویل:**۰۹/۰۲**

هوش مصنوعی و سیستم‌های خبره
دکتر سلیمی
پاییز ۱۴۰۳

مقدمه

در این تمرین، از شما می‌خواهیم که سه عامل مختلف را بر اساس الگوریتم‌های جستجوی BFS، DFS و AStar پیاده‌سازی کنید تا بتوانند مسئله‌ی زیر را حل کنند:



شرح

در این مسئله، عامل از نقطه‌ی شروع (نقطه‌ی آبی‌رنگ) حرکت خود را آغاز می‌کند و تلاش می‌کند تا به مقصد (نقطه‌ی سبزرنگ) برسد. عامل می‌تواند به صورت خطی حرکت کند یعنی قابلیت حرکت به صورت افقی، عمودی و

مورب را دارا می‌باشد. توجه کنید که، خانه‌های سیاه‌رنگ موجود در محیط مانع هستند و عامل قابلیت حرکت برروی آنها را ندارد.

پیش‌نیازها

پیاده‌سازی این تمرین را باید با پایتون انجام دهید و درنتیجه نیاز است که آن را نصب داشته باشید. برای نصب آن می‌توانید از این لینک استفاده کنید. همچنین برای اجرای تمرین به کتابخانه‌ی pygame نیاز دارید که می‌توانید توضیحات و نحوه‌ی نصب آن را از این لینک مشاهده کنید.

جزئیات مسئله و پیاده‌سازی

در ادامه به توضیح فایل‌ها و نحوه‌ی پیاده‌سازی عامل‌ها می‌پردازیم.

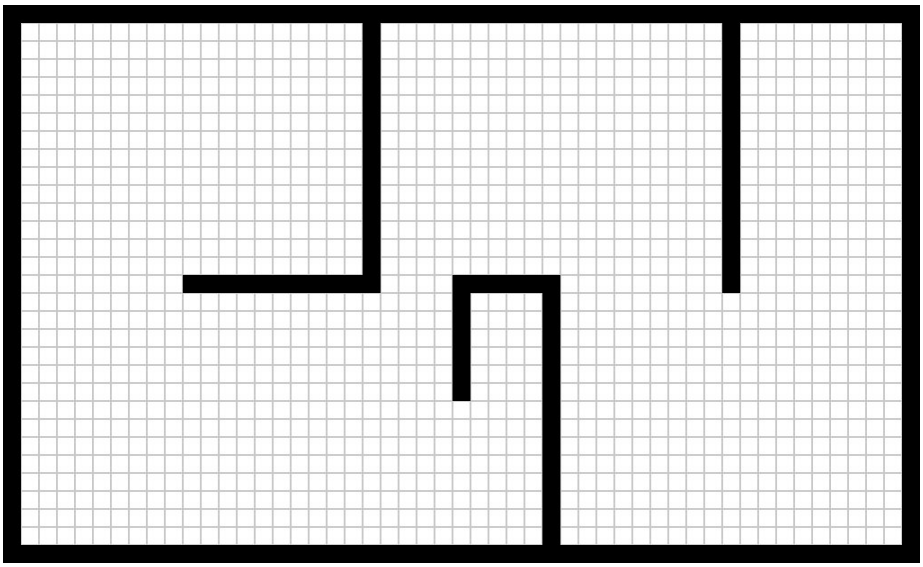
۱. ساختار کد و معرفی فایل‌ها

این تمرین شامل چند فایل مختلف است که هر یک نقش مشخصی در پیاده‌سازی و نمایش نتایج دارند:

۱. **main.py**: فایل اصلی تمرین می‌باشد و با اجرای این برنامه می‌تواند نتایج را مشاهده کنید. پیاده‌سازی عامل‌ها در این فایل انجام می‌شود. در این فایل می‌توانید تنظیمات مربوط به اجرا و محیط مسئله را نیز تغییر دهید:

```python
if __name__ == "__main__":
    s_start = (5, 5) # Starting point
    s_goal = (45, 25) # Goal

    FPS = 60
    generate_mode = False # Turn to True to change the map
    map_name = 'default'

    if generate_mode:
        gn.main(map_name)

    else:
        agent = DFS_Agent(s_start, s_goal, map_name) # Choose the agent here
        path, visited = agent.searching()

        # Plotting the path
        plot = Plotting(s_start, s_goal, map_name, FPS)

        plot.animation(path, visited)
```

- در دو خط اول می‌توانید نقاط شروع و مقصد را مشخص کنید.
- با تغییر FPS میتوانید سرعت به‌روزرسانی صفحه نمایش را تغییر دهید. پیشنهاد می‌شود این مورد را روی ۶۰ نگه دارید.
- با تغییر متغیر generate_mode به مقدار True و اجرای فایل وارد حالت تولید نقشه می‌شود که به شما این امکان را می‌دهد تا محیط مدنظر خودتان را تولید کنید:



در اینجا با کلیک بر روی هر خانه می‌توانید در آن مانع اضافه و یا حذف کنید.

- با بستن برنامه نقشه‌ی جدیدی که ساخته‌اید در فایلی با نامی که در map_name مشخص شده است ذخیره می‌شود.
- در بخش else کد بالا می‌توانید عامل انتخابی خود را مشخص کنید.

در این فایل سه کلاس مختلف معادل با هرکدام از عامل‌ها قرار دارد. هر عامل دارای یک تابع به نام searching می‌باشد که وظیفه‌ی شما تکمیل‌سازی آن‌ها است.

```
10  class BFS_Agent(AbstractSearchAgent):
11      def searching(self):
12          """
13          BFS search algorithm
14
15          Returns:
16          * path (list): The planned path from start to goal
17
18          * visted (list): list of visited nodes
19          """
20
21          # TODO
```

این تابع با جستجو در محیط مسیری به مقصد پیدا کرده و در نهایت نقاط بررسی شده و همچنین مسیر پیدا شده را برمی‌گرداند.

فایل‌های موجود در پوشه‌ی utils جهت کمک به شما و اجرای تمرین پیاده‌سازی شده‌اند. شما نیازی به ایجاد تغییر در این فایل‌ها ندارید.

کاربرد و کارکرد هرکدام از آن‌ها به صورت زیر است:

۲. **agent.py**: این فایل شامل کلاس پایه‌ی AbstractSearchAgent برای عامل‌های جستجو است. این کلاس شامل تابع‌هایی برای یافتن همسایگان یک گره (get_neighbor) و استخراج مسیر نهایی (extract_path) می‌باشد.

تابع get_neighbor، همسایه‌های یک خانه را بررسی میکند و لیستی از همسایه‌ها که مانع نیستند را برمی‌گرداند.

```python
def get_neighbor(self, s):
    """
    Find neighbors of the state that are not obstacles.
    :param s: current state
    :return: list of neighbors
    """

    return [(s[0] + u[0], s[1] + u[1]) for u in self.u_set if (s[0] + u[0], s[1] + u[1]) not in self.obs]
```

تابع extract_path، دیکشنری PARENT را به عنوان ورودی می‌گیرد و مسیر از مقصد به مبدا را محاسبه می‌کند و برمی‌گرداند.

```python
def extract_path(self, PARENT):
    """
    Extract the path from the start to the goal based on the parent dictionary.
    :return: the planned path
    """

    path = [self.s_goal]
    s = self.s_goal

    while s != self.s_start:
        s = PARENT[s]
        path.append(s)

    return list(reversed(path))
```

این کلاس دارای فیلدهای زیر می‌باشد:

```
    def __init__(self, s_start, s_goal, map_name):
        self.s_start = s_start
        self.s_goal = s_goal

        self.Env = env.Env(map_name)
        self.u_set = self.Env.motions
        self.obs = self.Env.obs

        self.OPEN = []
        self.CLOSED = []
        self.PARENT = dict()
        self.g = dict()

```

تمام عامل‌های موجود در فایل main.py از این کلاس ارث‌بری می‌کنند و شما می‌توانید از ویژگی‌ها و توابع این کلاس استفاده کنید.

۳. **env.py: این فایل محیط جستجو را ایجاد می‌کند. محیط شامل یک نقشه و مجموعه‌ای از موانع است که به عنوان محدودیت در جستجو عمل می‌کنند.**

```
    def __init__(self, map_name):
        self.x_range = 51
        self.y_range = 31
        self.motions = [(-1, 0), (-1, 1), (0, 1), (1, 1),
                        (1, 0), (1, -1), (0, -1), (-1, -1)]
        self.obs = self.load_obstacles(map_name)
```

حرکت‌های مجاز عامل و همچنین اندازه‌ی نقشه را در اینجا می‌توانید بررسی کنید.

۴. **generator.py: این فایل برای ساخت نقشه‌های دلخواه استفاده می‌شود. توابع موجود در این فایل وظیفه‌ی نمایش نقشه در حالت ویرایش و ذخیره‌ی آن بعد از بسته شدن را به عهده دارند.**

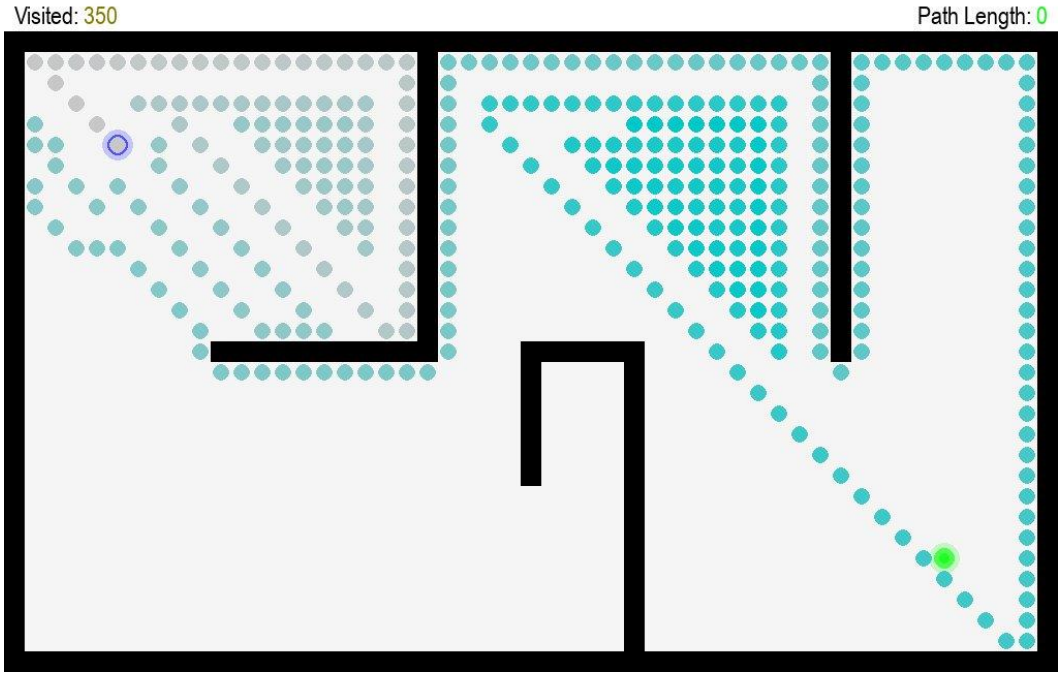
۵. **plotting.py: این فایل وظیفه نمایش مسیر و نقطه‌های بازدید شده در جریان جستجو را برعهده دارد. توابع این فایل به شما کمک می‌کنند تا نتایج جستجوی خود را به صورت بصری با استفاده از کتابخانه‌ی** pygame **مشاهده کنید.**

۲. وظیفه‌ی شما: پیاده‌سازی الگوریتم‌های جستجو

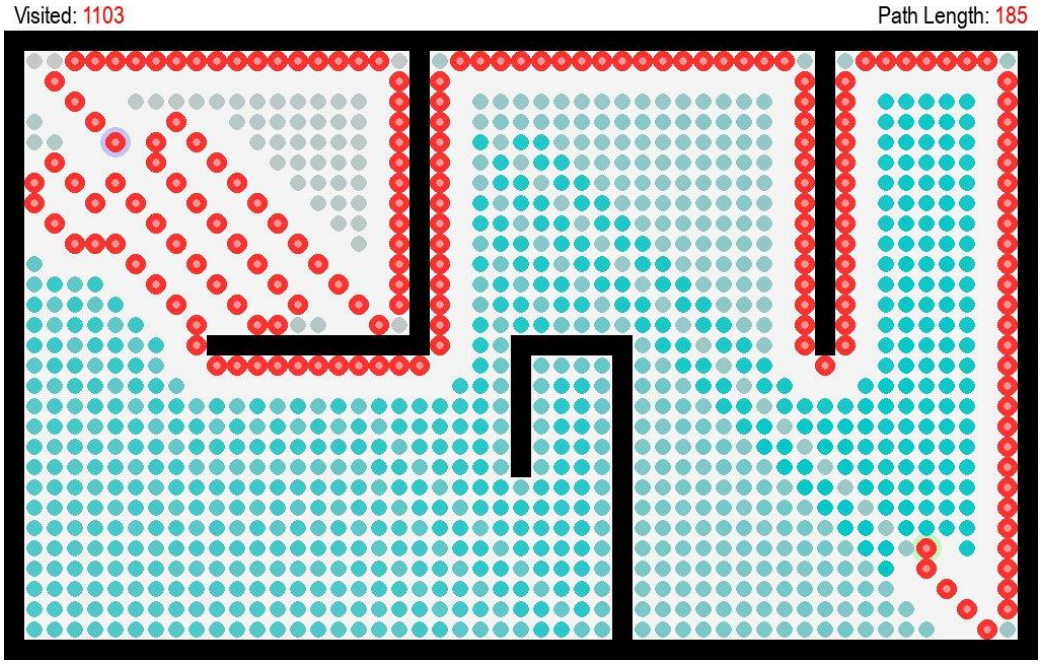
شما باید تابع searching **موجود در هرکدام از کلاس‌های** BFS_Agent، DFS_Agent **و** AStar_Agent **را که در فایل** main.py **قرار دارند، تکمیل کنید. این توابع باید به ترتیب لیست نقاطی که مسیر را تشکیل می‌دهند و لیستی که شامل تمام نقاط بازدیدشده است را برگردانند. شما می‌توانید از توابع کمکی و ویژگی‌های کلاس پایه که در بخش قبل توضیح داده شد استفاده کنید. همچنین در صورت نیاز می‌توانید برای هر کلاس تابع‌های دیگری تعریف کنید و از آن‌ها استفاده کنید. به طور مثال نیاز است که برای عامل** AStar **تابعی برای محاسبه‌ی تابع مکاشفه‌ای (heuristic) تعریف کنید تا بتوانید در تابع** searching **از آن برای محاسبه‌ی هزینه استفاده کنید. بعد از تکمیل کردن تابع جستجو می‌توانید عامل موردنظر را انتخاب کرده و با اجرای فایل** main.py **نتیجه‌ی جستجوی عامل را مشاهده کنید.**

۳. اجرا و مشاهده‌ی نتایج

با اجرای فایل main.py**، برنامه شروع به نمایش فرآیند جستجو می‌کند:**



بعد از اتمام جستجو و پیدا کردن مقصد، مسیر نهایی نمایش داده می‌شود:



مشاهده می‌شود که نقاط مربوط به مسیر نهایی با رنگ قرمز و دیگر نقاط جستجو شده با رنگ آبی نمایش داده شده‌اند. همچنین تعداد نقاط بررسی شده و طول مسیر نهایی در قسمت بالایی صفحه نمایش قابل مشاهده است.

۴. بخش امتیازی

فرض کنید که دیوارها خطرناک هستند و نباید به آن‌ها نزدیک شویم. می‌خواهیم عاملی داشته باشیم که در حرکت خود شرط ایمنی را در نظر بگیرد و مسیر نهایی‌ای که پیدا می‌کند بیشترین ایمنی را داشته باشد. یعنی سعی کند در حرکت خود تا جای ممکن از تمام دیوارها فاصله بگیرد.

عامل چهارمی بر مبنای AStar بسازید و سعی کنید برای آن یک تابع مکاشفه‌ای مناسب تعریف کنید که بتواند این شرط ایمنی را رعایت کند.

تحویل و ارزیابی

برای تحویل تمرین کافی است تا فایل main.py خود را که تکمیل کرده‌اید، بارگذاری نمایید.

تمرین دارای ارائه می‌باشد و بخش مهمی از نمره‌ی تمرین وابسته به ارائه‌ی شما خواهد بود. با توجه به این مورد نیازی به نوشتن گزارش برای تمرین ندارید اما سعی کنید کدهایتان تمیز و قابل فهم باشند و برایشان توضیح (comment) بگذارید. تسلط کافی به کد و پیاده‌سازیتان شرط لازم برای دریافت نمره‌ی کامل می‌باشد.

نکات تکمیلی

- سوالات و ابهامات خود را ترجیحاً در گروه تلگرامی درس مطرح کنید.
- در صورت بروز مشکل می‌توانید با @MohammadM۴۰۴ و یا @sadrabe در ارتباط باشید.
- در صورت مشاهده‌ی هرگونه تقلب نمره‌ی منفی صد برای تکلیف در نظر گرفته می‌شود.
- فرمت نام‌گذاری تکلیف به صورت زیر می‌باشد:

CHW1-[Student-ID]-[Student-Name]

موفق باشید.