مستندات پروژه هوش مصنوعی: سیستم مسیریابی هوشمند آمبولانس

درس: هوش مصنوعی
موضوع: جستجو (Search) و بهینه‌سازی تکاملی (Genetic Algorithm)
تهیه کننده: آرش عربی - آبتین ممشخانی

۱. مقدمه و شرح کلی سیستم

هدف این پروژه طراحی یک سیستم هوشمند برای مسیریابی آمبولانس‌ها در شرایط اضطراری است. این سیستم باید بتواند در یک محیط شبکه‌ای (Grid) که شامل موانع ترافیکی و چراغ‌های راهنمایی با زمان‌بندی مشخص است، بهترین مسیر را برای رسیدن به مصدومین پیدا کند.

پروژه در سه فاز (سناریو) انجام شده است:

1. **جستجوی ناآگاهانه (UCS):** یافتن مسیر بهینه بدون هیوریستیک.
2. *جستجوی آگاهانه (A)*: یافتن مسیر بهینه با استفاده از تابع هیوریستیک.
3. **الگوریتم ژنتیک (GA):** تخصیص بهینه مصدومین بین چند آمبولانس.

۲. معماری مشترک و محیط مسئله (`MapEnv`)

برای رعایت اصول مهندسی نرم‌افزار و جلوگیری از تکرار کد، یک کلاس پایه به نام `MapEnv` طراحی شده است که وظیفه مدل‌سازی محیط را بر عهده دارد. این کلاس در هر سه سناریو استفاده می‌شود.

ویژگی‌های محیط:

- **نقشه:** یک ماتریس که شامل `S` (شروع)، `G` (مصدوم)، `L` (چراغ راهنمایی) و اعداد (ترافیک) است.
- **هزینه حرکت پویا (Time-Dependent):** هزینه ورود به خانه‌ها ثابت نیست و وابسته به زمان رسیدن به آن خانه است.
  - **خانه‌های عددی:** هزینه برابر با عدد خانه.
  - **چراغ راهنمایی (L):** دارای یک چرخه ۲۰ دقیقه‌ای است:
    - دقیقه ۰ تا ۹: **سبز** (هزینه ۱).

    - دقیقه ۱۰ تا ۱۹: **قرمز** (هزینه ۱۰).
- **اقدامات مجاز:** بالا، پایین، چپ، راست و **ماندن (STAY)**. اقدام `STAY` دارای هزینه ۱ است و برای تنظیم زمان رسیدن به چراغ سبز حیاتی است.

۳. سناریو اول: جستجوی ناآگاهانه (UCS)

در این سناریو، آمبولانس هیچ دانش قبلی از فاصله تا هدف ندارد و باید با جستجوی سیستماتیک، کم‌هزینه‌ترین مسیر را بیابد. از آنجایی که گراف ما دارای یال‌هایی با وزن‌های متفاوت (۱، ۱۰، اعداد ترافیک) است، الگوریتم **Uniform Cost Search (UCS)** یا همان **Dijkstra** بهترین گزینه برای تضمین بهینگی است.

چالش اصلی: گراف وابسته به زمان

در گراف‌های معمولی، اگر یک بار به خانه `(x, y)` برسیم، رسیدن مجدد به آن با هزینه بیشتر بیهوده است. اما در اینجا، **زمان رسیدن** مهم است. رسیدن به یک چراغ قرمز در زمان ۱۵ (هزینه عبور ۱۰) بدتر از رسیدن به همان چراغ در زمان ۲۰ (هزینه عبور ۱) است.

راهکار پیاده‌سازی:

1. **تعریف فضای حالت (State Space):** هر گره در درخت جستجو شامل موارد زیر است:
   `State = (Row, Col, Remaining_Goals, Cycle_Phase)`
    - `Remaining_Goals` : لیست اهداف باقی‌مانده (چون ممکن است چند مصدوم داشته باشیم).
    - `Cycle_Phase` : زمان فعلی به پیمانه ۲۰ (`current_time % 20`). این پارامتر باعث می‌شود الگوریتم بفهمد که وضعیت چراغ در زمان‌های مختلف متفاوت است و حالت‌های "صبر کردن" را حذف نکند.
2. **مدیریت صف اولویت (Priority Queue):** صف بر اساس `g(n)` (هزینه طی شده از مبدأ) مرتب می‌شود.
3. **برخورد با چراغ قرمز:** الگوریتم به صورت خودکار و ناآگاهانه اقدام `STAY` را امتحان می‌کند. اگر ماندن پشت چراغ باعث شود در نهایت با هزینه کمتری (سبز شدن) عبور کنیم، UCS آن مسیر را انتخاب خواهد کرد.

۴. سناریو دوم: جستجوی آگاهانه (A*)

در این سناریو، ما از موقعیت مکانی مصدومین آگاه هستیم و باید با طراحی یک **تابع هیوریستیک (Heuristic)** مناسب، سرعت جستجو را افزایش دهیم در حالی که بهینگی (Optimality) حفظ شود.

طراحی هیوریستیک (Heuristic Function)

چالش اصلی این است که ممکن است **چندین مصدوم (Goal)** در نقشه وجود داشته باشد. هیوریستیک فاصله منهتن ساده (`Manhattan Distance`) تا نزدیک‌ترین هدف کافی نیست، زیرا هزینه رفتن به سایر اهداف را نادیده می‌گیرد.

ما از ترکیب زیر استفاده کردیم که هم **Admissible** (خوش‌بینانه) و هم **Consistent** است:

$$h(n) = \text{Dist}(Current, Nearest\_Goal) + \text{MST}(Remaining\_Goals)$$

1. **Dist(Current, Nearest_Goal):** فاصله منهتن تا نزدیک‌ترین مصدوم باقی‌مانده. این کمترین مسافتی است که قطعاً باید طی شود.
2. **MST(Remaining_Goals):** طول درخت پوشای کمینه (Minimum Spanning Tree) بین تمامی مصدومین باقی‌مانده. این مقدار، تخمینی از هزینه اتصال تمام مصدومین به یکدیگر است.

چرا MST؟

مسئله بازدید از چند نقطه شبیه به مسئله فروشنده دوره‌گرد (TSP) است. MST یک تقریب پایین‌دست (Lower Bound) برای TSP است و تضمین می‌کند که هیوریستیک هرگز هزینه واقعی را بیشتر از حد تخمین نمی‌زند (Admissible)، بنابراین A* پاسخ بهینه را تضمین می‌کند.

۵. سناریو سوم: الگوریتم ژنتیک (GA)

در این سناریو، هدف تخصیص بهینه M مصدوم بین N آمبولانس است به طوری که زمان اتمام عملیات (زمانی که آخرین آمبولانس به مقصد می‌رسد یا **Makespan**) کمینه شود.

اجزای الگوریتم ژنتیک:

۱. نمایش کروموزوم (Genotype)

هر راه حل (کروموزوم) یک لیست به طول تعداد مصدومین است.

- **ایندکس:** شماره مصدوم.
- **مقدار:** شماره آمبولانس مسئول.
- *مثال:* `[0, 1, 0]` یعنی مصدوم ۱ و ۳ با آمبولانس ۰، و مصدوم ۲ با آمبولانس ۱.

۲. تابع برازندگی (Fitness Function)

برای محاسبه کیفیت یک کروموزوم:

1. مصدومین را بر اساس آمبولانس گروه‌بندی می‌کنیم.

2. برای هر آمبولانس، باید مسیری را بیابیم که تمام مصدومین اختصاص‌یافته را ویزیت کند. از آنجا که تعداد مصدومین هر آمبولانس کم است، از **جایگشت کامل (Permutations)** برای یافتن بهترین ترتیب ملاقات استفاده می‌کنیم.
3. هزینه سفر بین نقاط با استفاده از تابع `A*` (مشابه سناریو ۲ ولی با Caching برای سرعت) محاسبه می‌شود.
4. **Makespan** برابر است با بیشترین زمانی که یک آمبولانس صرف کرده است.
5. $$Fitness = \frac{1}{Makespan}$$

۳. عملگرهای ژنتیک

- **انتخاب (Selection):** روش تورنمنت (Tournament Selection). دو والد تصادفی انتخاب شده و بهترین آن‌ها برگزیده می‌شود.
- **ترکیب (Crossover):** برش تک‌نقطه‌ای (Single-point crossover) روی لیست کروموزوم.
- **جهش (Mutation):** با احتمال ۱۰٪، آمبولانس مسئول یک مصدوم به تصادف تغییر می‌کند.
- **نخبه‌گرایی (Elitism):** بهترین جواب‌های هر نسل مستقیماً به نسل بعد منتقل می‌شوند تا کیفیت جواب هرگز افت نکند.

۶. نحوه اجرا

کدها به زبان پایتون و بدون وابستگی به کتابخانه‌های خارجی نوشته شده‌اند. برای اجرا:

1. ترمینال را باز کنید.
2. دستور زیر را وارد کنید:

```
python AI_Project_Scenario1.py
```

(برای سناریوهای دیگر نام فایل مربوطه را وارد کنید).
3. ورودی را به صورت کامل (شامل ابعاد و نقشه) کپی کرده و در ترمینال Paste کنید و Enter بزنید.

نمونه ورودی:

```
3 5
S9679
917LG
19999
```

۷. نتیجه‌گیری

این پروژه نشان داد که:

- برای محیط‌های پویا (مثل چراغ راهنمایی)، تعریف دقیق فضای حالت (شامل زمان) در الگوریتم‌های جستجو حیاتی است.
- استفاده از هیوریستیک‌های ترکیبی (مثل MST) در A* می‌تواند کارایی را در مسائل چندهدفه به شدت افزایش دهد.
- الگوریتم ژنتیک ابزاری قدرتمند برای حل مسائل NP-Hard مانند تقسیم وظایف (VRP) است، به شرطی که تابع برازندگی دقیقی تعریف شود.