به نام خدا

دانشکده مهندسی کامپیوتر

مبانی و کاربردهای هوش مصنوعی

پروژه دوم

مهلت تحویل 18 اردیبهشت ۱۴۰۳

فاز ۰

لینک پروژه : Map Coloring

در فاز ۰ این پروژه می‌خواهیم همان مسئله مثال درس، یعنی رنگ‌آمیزی نقشه را حل کنیم. در این بخش سعی داریم با مدل کردن این مسئله به شکل یک CSP و با استفاده از الگوریتم‌هایی که برای حل و بهبود عملکرد حل یاد گرفتیم، هر نقشه‌ای را بتوانیم بدون این که زمان زیادی ببرد، رنگ‌آمیزی کنیم. دقت کنید قبل از شروع کار پکیج‌های مورد نیاز پروژه که در فایل requirements.tx قرار دارند نصب کرده باشید. میدانیم طبق قضیه چهار رنگ فقط با استفاده از ۴ رنگ میتوانیم هر نقشه را طوری رنگ‌آمیزی کنیم که هیچ ۴ ناحیه مجاوری رنگ تکراری نداشته باشند.

با دو روش می‌خواهیم این CSP را حل کنیم:

۱- جستجوی عقبگرد (بک‌ترکینگ سرچ)

۲- بهبود تکرار شونده

به یاد بیاورید که دو روش عمومی برای بهبود بک‌ترکینگ یاد گرفتیم: فیلترینگ و اوردرینگ. در فیلترینگ با بررسی دامنه‌های متغیرها قبل از پیش بردن رنگ‌آمیزی، مقادیری که میتوانستیم متوجه بشویم قرار نیست در یک جواب برای مسئله باشند (در صورت مقداردهی شدن مطمئناً منجر به بک‌ترک خواهند شد) از دامنه حذف میشوند. در ابتدا بدون هیچ فیلترینگی اقدام به رنگ‌آمیزی نقشه‌ها خواهید کرد. در این روش تا ناسازگاری رخ نداده، بک‌ترکی رخ نخواهد داد؛ به عبارتی هرگاه دو ناحیه مجاور همرنگ مشاهده شد باید بعد از آن شاهد بک‌ترک باشید. ساده‌ترین نوع فیلترینگ **forward checking**، و بعد از آن **arc consistency** بود که هر دو را پیاده‌سازی خواهید کرد، با استفاده از فیلترینگ باید کاهش بسیار واضحی در تعداد بک‌ترکینگ‌ها مشاهده کنید. روش دوم برای بهبود الگوریتم **ordering** بود. در این روش با کمک هیوریستیک‌هایی که یاد گرفتید در در انتخاب بین متغیرهای باقیمانده و رنگ‌های فیلتر نشده در دامنه متغیرها هوشمندانه عمل خواهید کرد تا مجدداً سعی بر کاهش مقدار بک‌ترک داشته باشیم و در نتیجه مسئله سریعتر حل شود.

سه قطعه **map.py**، **solver.py**، و **utils.py** را مشاهده میکنید. کد **map.py** مربوط به پیش‌پردازش و پردازش‌های تصویری لازم روی تصاویر نقشه‌هاست و نیازی به تغییر دادن آن نیست. کد **solver** را با آرگومان نام نقشه‌ای که میخواهیم رنگ شود برای اجرای الگوریتم اجرا میکنیم (به همراه فلگ‌های مربوط به مد اجرا که در ادامه توضیح داده شده است) و تنها دو متود **solve** به دو روش اشاره شده، که توسط کامنت‌ها نیز مشخص شده‌اند را در آن کامل میکنید، ولی با خواندن میتوانید درک بهتری از متغیرهای ورودی متدهایی که قرار است بنویسید پیدا کنید. در نهایت تمامی الگوریتم‌های مورد استفاده در متدهای **solve** در **utils.py** مشخص شده‌اند. کامنت‌ها را به دقت بخوانید تا بتوانید بهتر و مطابق آنچه انتظار میرود به خوبی متودها را پیاده کنید. قسمت‌هایی که باید پر کنید در کد به وضوح مشخص شده‌اند. در نهایت با سه نقشه‌ی آماده شده ایران قدیم، استان تهران و کشور آمریکا که در فولدر قرار دارند کد خود را آزمایش کنید. اجرای کد به این شکل میباشد:

فلگ اول: بدون فیلترینگ **n**، با فوروارد چکینگ **fc**، و با آرک کانسیستنسی **ac**

فلگ دوم: استفاده از اوردرینگ برای انتخاب متغیر **t** و عدم استفاده **f**

فلگ سوم: استفاده از اوردرینگ برای انتخاب مقدار **t** و عدم استفاده **f**

برای مثال در ابتدایی ترین مد دستور به این شکل:

```
python solver.py iran.jpg -n -f -f
```

و در پیشرفته‌ترین مد دستور به این شکل در میاید:

```
python solver.py usa.png -ac -t -t
```

برای دیباگ کردن میتوانید مقدار **SLEEP_TIME_IN_MILLISECONDS** در خط هشتم **solver.py** را زیادتر کنید تا مراحل رنگ شدن و بک‌ترکینگ را بهتر مشاهده کنید. (و اگر پرینتی در کنسول اضافه کردید بتوانید شمرده‌تر آن را با خروجی گرافیکی تطبیق دهید.) دقت کنید با **Esc** میتوانید اجرا را متوقف کنید.

بسته به میزان استفاده از ابزارهای بهبود الگوریتم تعداد بک‌ترک نیز عوض میشود. انتظار میرود با استفاده از هر دو ابزار بتوانید با تعداد بک‌ترک صفر نقشه آمریکا را رنگ کنید.

راهنمایی) تعداد بک‌ترک‌ها بر اساس مد اجرا مطابق زیر خواهند بود:

ابتدایی ترین مد: ایران 11، استان تهران 39، و آمریکا 65

با forward-checking ولی بدون اوردرینگ: ایران 0، استان تهران 2، آمریکا 1

با هر دو اوردرینگ ولی بدون فیلترینگ: ایران 14، استان تهران 24، آمریکا ...

با هر نوع فیلترینگ و هر دو اوردرینگ: ایران 0، استان تهران 0، و آمریکا 0

میتوانید از راهنمایی نیز حدس بزنید که بعضاً تعداد بک‌ترک‌ها مطابق انتظار نخواهند بود. سعی کنید توجیهی برای این اتفاق بیابید.

امتیازی: سعی کنید راه حلی بیابید که در آن با ایجاد تغییری در نحوه اوردرینگ، حتی در صورت عدم وجود فیلترینگ، اوردرینگ عملکرد را بهتر کند. (سعی کنید با اوردرینگ و بدون فیلترینگ آمریکا را رنگ‌آمیزی کنید...)

روش دوم حل CSP با کمک بهبود تکرار شونده است. در این روش مقدار دهی اولیه رندومی به همه متغیرها انجام می دهیم و به صورت رندوم مقادیری که منجر به نقض محدودیت شده‌اند با مقدار دیگری جایگزین میکنیم. از هیوریستیکی که در درس برای انتخاب مقدار جدید یاد گرفتید استفاده کنید. این روش ضمانتی به رسیدن به جواب ندارد و وابسته به مقداردهی اولیه ممکن است بسیار سریع به جواب برسد یا هیچوقت به جواب نرسید و در یک حلقه بی‌انتها گیر کند، ولی اکثر اوقات برای نقشه‌هایی که در این پروژه رنگ میکنیم با سرعت خوبی به جواب میرسد. برای حل مسئله به این روش دستور را به این شکل اجرا کنید:

```
python solver.py usa.png -ii
```

فاز ۱:

مقدمه

در این فاز پروژه عاملی را برای بازی شطرنج طراحی خواهید کرد. در این مسیر عوامل جستجوی مینیماکس و مینیماکس احتمالی را پیاده سازی خواهید کرد. با هرس آلفا بتا آشنا خواهید شد و یک تابع ارزیابی را بررسی میکنید.

لینک پروژه : Chess

در ابتدا پروژه را از لینک بالا دریافت کنید و سپس با استفاده از دستورات گفته شد در readme پروژه را روی سیستم خود اجرا کنید.

شما باید بخش هایی از فایل Agent.py را تغییر دهید.

توجه : در صورت علاقه میتونید سایر بخش های پروژه را نیز تغییر بدهید ولی تمامی تغییرات میبایست در گزارش پروژه ثبت شود.

به شطرنج چند عاملی خوش آمدید !

در ابتدا میتونید بازی رو با استفاده از دستور :

```
python src/main.py
```

اجرا کنید در این جا شما با یک عامل که حرکات خود را به صورت رندوم انجام میدهد بازی میکنید. (امیدوارم که ببرید)

سپس اگر علاقه ای برای دیدن بازی دو عامل رندوم دارید بازی را با استفاده از دستور :

```
python src/main.py -p BotVsBot
```

اجرا کنید سپس مشاهده میکنید که دو عامل با یکدیگر به صورت رندوم مسابقه میدهند.

۱) مینیماکس :

در این سوال باید کلاس `MinimaxAgent` را کامل کنید. درخت مینیماکس شما باید تا عمق دلخواه گسترش یابد تا در آخرین سطح به برگ ها برسد برگها باید به وسیله تابع مناسب ارزیابی شوند. به این منظور باید از تابع ارزیابی `evaluate_board_state` استفاده که به عنوان ورودی board و turn رو میگیرد که turn برابر است با بازیکنی که میخواهید برای آن ارزیابی انجام شود.

برای ارزیابی عامل خود میتوانید از دستور :

```
python src/main.py -p BotVsBot -af MinimaxAgent -df 2
```

که عامل اول را Minimax و عامل دوم random است و عمق مینیماکس را نیز برابر با ۲ قرار دادیم.

گزارش ۰ : تابع `evaluate_board_state` را در فایل agent.py بررسی کنید.

گزارش ۱ : چندین بازی را با شرایط زیر اجرا کنید و برد و باخت هر عامل را در آن حالت بررسی کنید.

- عامل انسانی با عامل مینیماکس :

```
python src/main.py -p PlayerVsBot -af MinimaxAgent -df 2
```

- عامل مینیماکس با عامل رندوم :

```
python src/main.py -p BotVsBot -af MinimaxAgent -df 2
```

- عامل مینیماکس با عمق ۲ در مقابل عامل مینیماکس با عمق ۲:

```
python src/main.py -p BotVsBot -af MinimaxAgent -df 2 -as MinimaxAgent -ds 2
```

گزارش ۲: حال عمق را به ۴ افزایش دهید مشاهده میکنید که عامل حرکات خود را به کندی انجام میدهد به طور مختصر دلیل این مورد را ذکر کنید. (در گزارش مدت زمان حرکت اول را نیز اعلام کنید)

۲) هرس آلفا-بتا :

در این سوال باید با اضافه کردن هرس آلفا-بتا، پیمایش درخت مینیماکس را ارتقا دهید. برای این کار کلاس `AlphaBetaAgent` را تکمیل کنید.در نتیجه ی این ارتقا باید شاهد افزایش سرعت الگوریتم باشید.

گزارش ۳: چندین بازی را با شرایط زیر اجرا کنید و برد و باخت و سرعت عمل هر عامل را در آن حالت بررسی کنید.

- عامل انسانی با عامل مینیماکس با هرس آلفا-بتا :

```
python src/main.py -p PlayerVsBot -af AlphaBetaAgent -df 2
```

- عامل مینیماکس با عمق ۲ با عامل مینیماکس با هرس آلفا-بتا با عمق ۲ :

```
python src/main.py -p BotVsBot -af AlphaBetaAgent -df 2 -as MinimaxAgent -ds 2
```

- عامل مینیماکس با هرس آلفا-بتا با عمق 4 با عامل مینیماکس با هرس آلفا-بتا با عمق ۲ :

```
python src/main.py -p BotVsBot -af AlphaBetaAgent -df 4 -as AlphaBetaAgent -ds 2
```

گزارش ۴: اگر عامل فعلی با عمق ۴ اجرا کنید متوجه میشوید که اختلاف زمانی بسیاری با عامل مینیماکس بدون هرس بوده است. این اتفاق را تحلیل کنید.

۳) مینیماکس احتمالی :

در مینیماکس و هرس آلفا-بتا فرض میشود حریف بهینه ترین انتخاب ها را انجام میدهد در حالیکه در واقعیت اینگونه نیست و مدلسازی احتمالی عاملی که انتخابهای غیربهینه دارد، ممکن است با نتیجه ی بهتری برای ما همراه باشد. عوامل تصادفی حریف نیز انتخابهای بهینه ندارند و بدین ترتیب مدلسازی آنها با جستجوی مینیماکس، ممکن است نتیجه ی بهینه ی نداشته باشد. روش مینیماکس احتمالی به جای در نظر گرفتن کوچکترین حرکات حریف، مدلی از احتمال حرکات را در نظر میگیرد پس برای ساده سازی مدل احتمالی، فرض کنید عامل حریف حرکات خود را از بین حرکات مجازشان به صورت یکنواخت و تصادفی انتخاب میکنند. در این سوال باید تغییرات لازم را در کلاس `ExpectimaxAgent` اعمال کنید.

گزارش ۵: چندین بازی را با شرایط زیر اجرا کنید و برد و باخت و سرعت عمل هر عامل را در آن حالت بررسی کنید.

- عامل مینیماکس با عمق ۲ با عامل مینیماکس احتمالی با عمق ۲:

```
python src/main.py -p BotVsBot -af ExpectimaxAgent -df 2 -as MinimaxAgent -ds 2
```

- عامل مینیماکس احتمالی با عمق ۲ با هرس آلفا-بتا با عمق ۲:

```
python src/main.py -p BotVsBot -af ExpectimaxAgent -df 2 -as AlphaBetaAgent -ds 2
```

گزارش ۶: چه تفاوتی در عملکرد عامل خود نسبت به عامل `AlphaBetaAgent` احساس میکنید؟ درباره آن توضیح دهید.

توضیحات تکمیلی

- پاسخ به سوالات باید به صورت فردی انجام شود. در صورت مشاهده تقلب، برای همه ی افراد نمره صفر لحاظ خواهد شد.
- پاسخ گزارشات را در قالب یک فایل PDF به همراه پروژه ی تکمیل شده در سامانه کورسز آپلود کنید.
- فرمت نامگذاری تمرین باید مانند AI_P2_40031082 باشد.
- در صورت هر گونه سوال یا ابهام از طریق ایمیل ؟ با تدریس یاران در ارتباط باشید. همچنین خواهشمند است در متن ایمیل به شماره دانشجویی خود اشاره کنید.
- همچنین می توانید از طریق تلگرام نیز با آیدی های زیر در تماس باشید و سوالاتتان را مطرح کنید:
- فاز ۰ : @Arriann و @AParsa404
- فاز ۱: @Mohammad_H915
- ددلاین این تمرین 18 اردیبهشت ۱۴۰۳ ساعت ۲۳:۵۵ است.