به نام خدا


درس هوش مصنوعی و سیستم های خبره

# تمرین امتیازی

طراحان:
محمدعلی آژینی، بهاره کاوسی نژاد، فرناز خوشدوست آزاد

مدرس درس:
دکتر عبدی

## مقدمه

در این تمرین قرار است با استفاده از الگوریتم‌های تدریس شده در بحث جستجو آگاهانه و ناآگاهانه، بازی مشهور water-sort را حل کنید. این بازی که در بیشتر گوشی‌های هوشمند در دسترس است در دسته بندی بازی‌های پازلی قرار دارد. در صورتی که تا به حال این بازی را تجربه نکردید می توانید از این لینک بازی را انجام دهید.
در این بازی شما چندین شیشه شفاف دارید که در هر لوله رنگ‌های مختلفی از آب وجود دارد. هدف این است که تمام آب‌ها را به صورت مرتب و مشابه در یکی از لوله‌ها تخلیه کنید به صورتی که در پایان همانند تصویر زیر، تمام آب‌های یک رنگ در یک لوله قرار بگیرند.

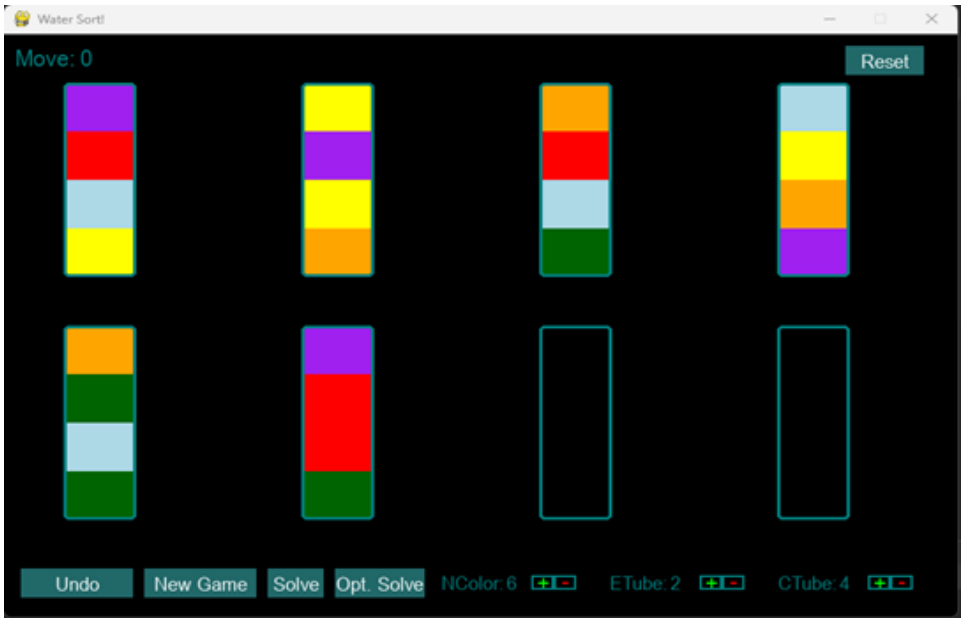
قوانین این بازی به شرح زیر است:

- به خاطر داشته باشید که هر لوله می‌تواند تنها تعداد محدودی از آب‌ها را در خود جای دهد و هیچ گاه نمی‌توانید تعدادی بیشتر از ظرفیت مشخص شده که برای تمامی شیشه‌ها یکسان است، آب بریزید.
- در این بازی شما تنها مجاز هستید که آب‌های رنگی را از شیشه مبدا به شیشه مقصد منتقل کنید که برای اینکار دو شرط زیر باید همزمان برقرار باشند:

  — بالایی ترین آب موجود در شیشه مبدا با بالاترین آب شیشه مقصد، همرنگ باشد.
  — در شیشه مقصد به اندازه کافی، فضای خالی برای جا دادن آب شیشه مبدا داشته باشیم.

## ۱ آماده سازی محیط تمرین

برای اینکه بتوانید در زمان حل تمرین، شهود بیشتری نسبت به بازی داشته باشید محیط گرافیکی ساده ای به زبان پایتون نوشته شده است که پیاده سازی آن در فایل game.py موجود است.




همانطور که در تصویر بالا مشاهده می‌کنید، این بازی از تعدادی شیشه پر و خالی که هرکدام ظرفیت مشخصی دارند تشکیل شده است(ظرفیت برای همه شیشه ها مقداری یکسان است). ظرفیت هر شیشه، تعداد شیشه‌های خالی و همچنین تعداد رنگ‌های مورد استفاده را می‌توانید از قسمت پایین محیط گرافیکی با استفاده از علامت های ‘+’، ‘-’ تغییر دهید. با زدن دکمه New Game بازی جدیدی مطابق با مقادیر مشخص شده ایجاد می‌شود. با پیاده سازی توابع مربوط به حل بازی که در ادامه به آنها اشاره خواهیم کرد، می‌توانید از دکمه های Solve, Opt. Solve استفاده کنید.
در فایل game.py کامنت‌هایی نوشته شده است که توصیه می‌شود با خواندن آنها دید بهتری نسبت به روند بازی داشته باشید.

توجه: امکان تغییر در فایل game.py وجود ندارد.

## ۲ مراحل پیاده سازی

برای اینکه بتوان بازی را با استفاده از دکمه‌های گفته شده حل نمود باید توابع نوشته شده در فایل ai_solution.py را کامل کنید(امکان اضافه کردن تابع و یا متغیرهای مورد نیاز به این فایل را دارید). در تابع solve با گرفتن وضعیت فعلی بازی، شما الگوریتم پایه ای را که در ادامه توضیح

داده می‌شود، پیاده می‌کنید. پس از پیاده سازی تابع اول با استفاده از توابع جستجوی تدریس شده در کلاس و پیاده سازی توابع heuristic مناسب برای این بازی، تابع بعدی به نام optimal_solve را تکمیل کنید که باید نسبت به تابع اول از لحاظ تعداد حرکت مورد نیاز برای حل بازی، عملکرد بهتری داشته باشد:

- پیاده سازی تابع **solve**:

۱. در ابتدا برای اینکه بتوانید بازی را اجرا کنید باید کتابخانه pygame را بر روی نسخه پایتون سیستم تان نصب کنید.

۲. ورودی تابع solve وضعیت بازی در هر مرحله را نشان می دهد. این وضعیت به صورت آرایه ای دو بعدی بوده که هر عضو آن شماره رنگ‌های موجود در هر شیشه را نشان می‌دهد. به عنوان مثال در زیر یک نمونه از مقدار ممکن متغیر گفته شده است که در آن شش شیشه داریم؛ دو تا از آنها خالی و چهار شیشه هرکدام حاوی سه رنگ هستند:

$$[[2, 0, 1], [2, 3, 1], [2, 3, 0], [0, 1, 3], [], []]$$

۳. با استفاده از تابع check_victory بررسی می‌کنیم که آیا بازی به پایان رسیده است یا نه. در صورتی که تمام رنگ‌ها مرتب شده باشند، مقدار متغیر solution_found را برابر با True قرار داده و return می‌کنیم.

۴. در صورتی که بازی به پایان نرسیده باشد، بر روی ورودی تابع پیمایش کرده و دو شیشه ای که امکان جابجایی رنگ میان آنها وجود دارد را انتخاب می‌کنیم. توجه داشته باشید که شیشه مبدا خالی نباشد و شیشه مقصد فضای کافی برای مقدار آب شیشه مبدا داشته باشد.
راهنمایی: برای اینکه تابع پیاده سازی عملکرد بهتری داشته باشد و از مشاهده استیت‌های تکراری جلوگیری شود، در صورتی که در شیشه مبدا بیش از یک واحد از آب‌های با رنگ یکسان روی هم قرار داشتند، همه را با هم در نظر گرفته و تنها بالایی ترین رنگ را انتخاب نکنید.

۵. پس از جابجایی آب‌ها میان دو شیشه مبدا و مقصد، به استیت جدیدی می‌رسیم و در صورتی که در متغیر self.visited_tubes این استیت را نداشته باشیم به عنوان یک حرکت به متغیر self.moves اضافه می‌کنیم و علاوه بر این به عنوان یک وضعیت مشاهده شده به لیست self.visited_tubes اضافه می‌کنیم.

۶. پس از اضافه کردن استیت جدید به متغیر self.visited_tubes و متغیر self.moves، تابع solve را با پاس دادن استیت جدید به صورت بازگشتی صدا می‌زنیم. توجه داشته باشید زمانی که دیگر از استیت به دست آمده نتوانیم حرکتی انجام دهیم، باید به صورت backtracking، حرکت‌های اضافه شده به متغیر self.moves را حذف کنیم. همچنین یکی از حالاتی که ممکن است دیگر حرکتی انجام نشود زمانی است به تمام آب ها به صورت مرتب در شیشه ها قرار گرفته اند و نیاز به حرکت دیگر نیست؛ که در این حالت از تابع، return می‌کنیم.

همانطور که متوجه شده اید مراحل گفته شده پیاده سازی الگوریتم اول عمق (DFS) برای حل این بازی بود. در ادامه می‌خواهیم به کمک تابع اول سطح (BFS) و استفاده از تابع heuristic مناسب، عملکرد الگوریتم پیاده سازی شده در بالا را از لحاظ تعداد حرکت‌های مورد نیاز، بهبود دهیم؛ کاری که در الگوریتم $A^*$ آن را انجام می‌دهیم.
بنابراین در این قسمت به پیاده سازی تابع $A^*$ برای حل بازی پرداخته و از شما انتظار می‌رود که با پیشنهاد توابع هیوریستیک مناسب عملکرد الگوریتم در قسمت قبل را بهبود بخشیده و با تعداد حرکت کمتری بتوان بازی را حل نمود.

- پیاده سازی تابع **optimal_solve**:

۱. در ابتدا برای ذخیره سازی استیت ها و حرکت‌های انجام شده و همچنین مقدار تابع $f = g + h$ نیاز به ایجاد یک صف اولویت یا priority queue داریم. مقدار تابع h برابر با مقدار تقریبی فاصله استیت فعلی با استیت نهایی بازی است و تابع g نشان دهنده فاصله استیت فعلی از استیت ابتدایی بازی است. علاوه بر این برای نگهداری مقادیر توابع h و g برای استیت‌های مختلف، نیاز به دو داده ساختار دیکشنری داریم.
۲. استیت ابتدایی بازی را درون صف ساخته شده ریخته و مقدار تابع g معادل با این استیت را برابر با صفر قرار می‌دهیم. مقدار h را با استفاده از تابع heuristic مناسب که باید پیاده سازی آن را انجام دهید، به دست می آید.
۳. در ادامه با استفاده از حلقه while تا زمانی که صف خالی نشده است، مراحل زیر را انجام می‌دهیم:
   (آ) در ابتدای حلقه، استیت و حرکت‌های انجام شده از صف را خارج کرده و درون متغیرهایی ذخیره می‌کنیم.
   (ب) در صورتی که بازی به پایان رسیده باشد، مقدار متغیر self.solution_found را برابر با True قرار داده و حرکت‌های ذخیره شده در صف را درون متغیر self.moves می‌ریزیم.
   (ج) در صورتی که بازی ادامه داشته باشد همانند قسمت قبل، بر روی شیشه‌های موجود پیمایش کرده و دو شیشه با شروط گفته شده برای جابجایی آب میان آنها را، انتخاب می‌کنیم.
   (د) پس از انتخاب دو شیشه و جابجایی آب میان آنها به استیت جدیدی رسیده و مقدار تابع g را برای آن، یک واحد اضافه می‌کنیم. با صدا زدن تابع heuristic برای استیت به دست آمده، مقدار تابع h را هم محاسبه می‌کنیم.
   (ه) پس از محاسبه مقدار توابع heuristic و g، مقدار تابع f متناظر با این استیت را محاسبه می‌کنیم.
   (و) در نهایت استیت به دست آمده به همراه مقدار تابع f و آرایه ای که در آن حرکت‌های بین شیشه‌ها ذخیره شده است را به صف اضافه می‌کنیم.