به نام خدا

تمرين عملي درس هوش مصنوعي – ماشين بردار پشتيبان بهاره كاوسي نژاد – 99431217

**بخش AI Agent:**

get\_best\_move(game, max\_depth): این تابع با گرفتن وضعیت فعلی بازی و حداکثر عمق جستجو به عنوان ورودی، با استفاده از الگوریتم آلفا-بتا برای تعیین بهترین حرکت برای بازیکن هوش مصنوعی استفاده می‌شود. این تابع تابع alphabeta را فراخوانی می‌کند و بهترین حرکت را که پیدا شده است، برمی‌گرداند.

alphabeta(game, max\_depth, maximizing\_player, alpha, beta): این تابع الگوریتم آلفا-بتا برای انتخاب بهترین حرکت را پیاده‌سازی می‌کند. آن وضعیت فعلی بازی، حداکثر عمق جستجو، یک پرچم بولین برای نشان دادن نوبت بازیکن بزرگنمایی و مقادیر آلفا و بتا برای قطعه‌بندی را دریافت می‌کند.

ابتدا بررسی می‌شود که آیا عمق جستجوی حداکثر رسیده است یا بازی به پایان رسیده است. اگر چنین باشد، ارزش ارزیابی وضعیت فعلی بازی و None به عنوان حرکت برگشت داده می‌شوند.

اگر نوبت بازیکن بزرگنمایی باشد، ارزش ارزیابی بیشینه و بهترین حرکت به ترتیب به float("-inf") و None مقداردهی اولیه می‌شوند.

برای هر حرکت مجاز، یک نمونه بازی جدید ایجاد می‌شود. به این منظور وضعیت جدید بازی کپی می‌شود و سپس حرکت انتخابی اعمال می‌شود.

برای وضعیت جدید، تابع alphabeta به صورت بازگشتی با وضعیت جدید، عمق کاهش یافته و نوبت بازیکن مقابل فراخوانی می‌شود.

ارزش ارزیابی بیشینه و بهترین حرکت به‌روزرسانی می‌شوند اگر یک حرکت بهتر پیدا شود.

مقدار آلفا به روزرسانی می‌شود و اگر مقدار بتا کمتر یا مساوی مقدار آلفا باشد، قطعه‌بندی انجام می‌شود.

اگر نوبت بازیکن کوچکنمایی باشد، فرایند مشابهی با بازیکن بزرگنمایی انجام می‌شود اما با استفاده از ارزش‌های ارزیابی و حرکت‌های کمینهتابع evaluate\_game\_state(game): این تابع وضعیت فعلی بازی را برای بازیکن هوش مصنوعی ارزیابی می‌کند. بر اساس عوامل مختلف، یک امتیاز مطلوبیت محاسبه می‌شود.

coin\_parity: این مقدار تفاوت تعداد دیسک‌ها بین بازیکن هوش مصنوعی و حریف را محاسبه می‌کند.

mobility: این مقدار تفاوت تعداد حرکت‌های مجاز بین بازیکن هوش مصنوعی و حریف را محاسبه می‌کند.

corner\_occupancy: تعداد دیسک‌های بازیکن هوش مصنوعی در گوشه‌های تخته را محاسبه می‌کند.

stability: این تابع تابع calculate\_stability را فراخوانی می‌کند تا پایداری دیسک‌های بازیکن هوش مصنوعی را محاسبه کند.

edge\_occupancy: تعداد دیسک‌های بازیکن هوش مصنوعی در لبه‌های تخته را محاسبه می‌کند.

در نهایت، این عوامل را با وزن‌های مربوطه ترکیب کرده و مقدار ارزیابی نهایی را بدست می‌آورد.

calculate\_stability(game): این تابع پایداری دیسک‌های بازیکن هوش مصنوعی روی تخته را محاسبه می‌کند.

neighbors(row, col): این تابع مختصات همسایه‌های یک خانه را برای یک خانه مشخص محاسبه می‌کند.

این تابع سه منطقه را تعریف می‌کند: گوشه‌ها، لبه‌ها و منطقه داخلی.

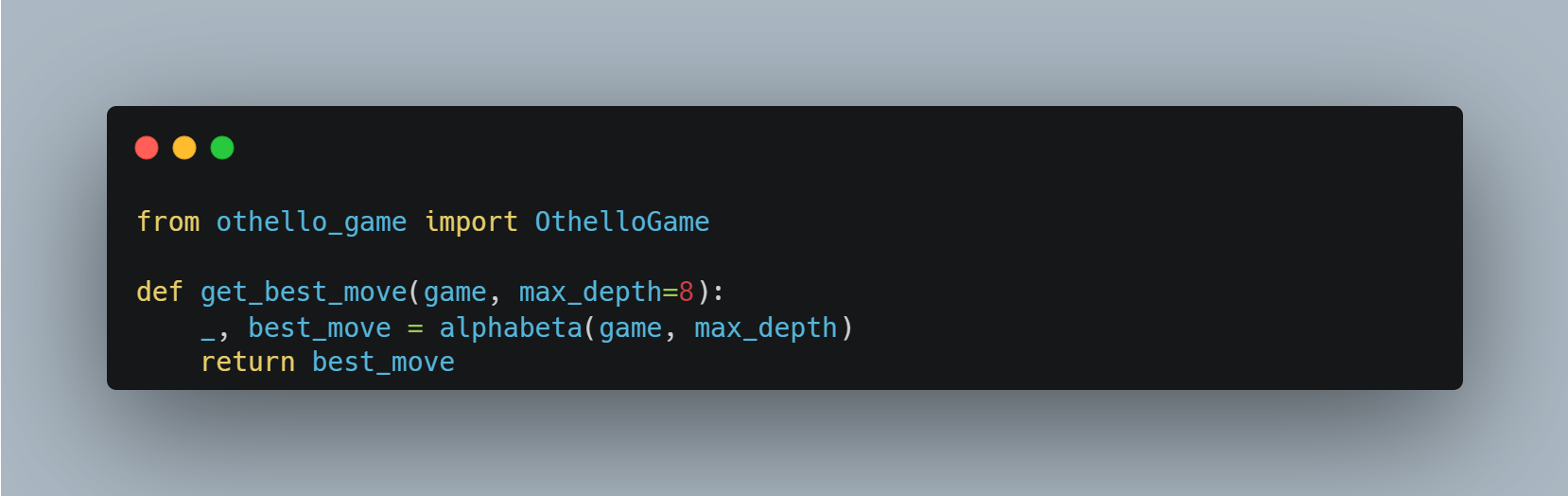
یک متغیر stable\_count برای ردیابی تعداد دیسک‌های پایدار تعریف می‌شود.

یک تابع کمکی به نام is\_stable\_disk تعریف می‌شود تا بررسی کند آیا یک دیسک پایدار است یا خیر.

برای هر منطقه، بررسی می‌شود که آیا هر دیسک در آن منطقه متعلق به بازیکن هوش مصنوعی است و پایدار است یا خیر. اگر چنین باشد، stable\_count افزایش می‌یابد.

در نهایت، stable\_count برگشت داده می‌شود.

این توابع با همکاری هم، وضعیت بازی را ارزیابی می‌کنند و با استفاده از الگوریتم آلفا-بتا بهترین حرکت را برای بازیکن هوش مصنوعی پیدا می‌کنند.



**بخش Othello Game:**

این کلاس `OthelloGame` یک بازی اتلو (Othello) را پیاده سازی می‌کند. اتلو یک بازی استراتژیک است که دو بازیکن در آن سعی می‌کنند بیشترین تعداد دیسک را بر روی تخته بازی خود قرار دهند.

بیایید به توضیح هر یک از توابع این کلاس بپردازیم:

1. `\_\_init\_\_(self, player\_mode="friend")`: این تابع متداول سازنده کلاس است که یک شیء از کلاس `OthelloGame` ایجاد می‌کند. آرگومان `player\_mode` نوع بازیکن را مشخص می‌کند، که در اینجا به صورت پیش فرض "friend" است. شیء ایجاد شده شامل ویژگی‌هایی مانند تخته بازی (board)، وضعیت فعلی بازیکن (current\_player) و نوع بازیکن (player\_mode) است.

2. `is\_valid\_move(self, row, col)`: این تابع بررسی می‌کند که آیا حرکت داده شده توسط بازیکن معتبر است یا خیر. اگر موقعیت (row، col) در تخته بازی خالی باشد و باعث تغییر دیسک‌های حریف در یک جهت معتبر شود، تابع مقدار `True` را برمی‌گرداند، در غیر این صورت `False` را برمی‌گرداند.

3. `flip\_disks(self, row, col)`: این تابع دیسک‌های حریف را در صورت امکان در جهت‌های معتبر معکوس می‌کند. با استفاده از تابع `is\_valid\_move`، این تابع مشخص می‌کند که در هر جهت چه تعداد دیسک باید معکوس شوند و سپس آن‌ها را معکوس می‌کند.

4. `make\_move(self, row, col)`: این تابع حرکت بازیکن را با استفاده از تابع `is\_valid\_move` و `flip\_disks` انجام می‌دهد. اگر حرکت معتبر باشد، دیسک بازیکن در موقعیت مورد نظر قرار داده می‌شود و دیسک‌های حریف معکوس می‌شوند. همچنین وضعیت فعلی بازیکن را تغییر می‌دهد.

5. `is\_game\_over(self)`: این تابع بررسی می‌کند که آیا بازی به پایان رسیده است یا خیر. اگر هیچ حرکت معتبری برای هیچ یک از بازیکنان نباشد یا تمام خانه‌های تخته بازی پر شده باشند، تابع مقدار `True` را برمی‌گرداند، در غیر این صورت `False` را برمی‌گرداند.

6. `get\_winner(self)`: این تابع برنده بازی را برمی‌گرداند. برای این منظور، تعداد دیسک‌های سیاه و سفید را در تخته بازی محاسبه کرده و بر اساس آن، عدد 1 را برای برنده سیاه، عدد -1 را برای برنده سفید و عدد 0 را برای مساوی برمی‌گرداند.

7. `get\_valid\_moves(self)`: این تابع لیستی از حرکت‌های معتبر را برمی‌گرداند. برای هر خانه از تخته بازی، تابع `is\_valid\_move` را فراخوانی می‌کند و اگر حرکت معتبر باشد، آن را به لیست حرکت‌های معتبر اضافه می‌کند. در نهایت، لیست حرکت‌های معتبر را برمی‌گرداند.

این کلاس `OthelloGame` امکانات و عملکردهای مورد نیاز برای اجرای یک بازی اتلو را فراهم می‌کند. با استفاده از توابع مختلف این کلاس، می‌توانیم حرکت‌های معتبر را بررسی کنیم، حرکت‌ها را انجام دهیم، وضعیت بازی را بررسی کنیم و برنده را تشخیص دهیم.

