مستندات پروژه دوم درس هوش مصنوعی

بازی چکرز

امیررضا پورکاظمی سالکویه - ساجده سپهرنیا

چکرز، یک بازی تخته ای کلاسیک است که در آن دو بازیکن بر روی یک شبکه ۸×۸ با هم بازی می‌کنند. هر بازیکن مجموعا ۱۲ مهره متمایز دارد که می‌توان آنها را با هدف گرفتن تمام مهره‌های حریف یا مسدود کردن حرکات آن‌ها فقط به سمت جلو حرکت داد. بازی روی مربع‌های مشکی اجرا می‌شود و تمام حرکات به‌صورت مورب می‌باشد. با رسیدن به ردیف عقب حریف، می‌توان مهره‌های خودی را «پادشاه» کرد، در آن صورت می‌توان به هر چهار طرف ممکن حرکت کرد. در این پروژه یک برنامه با استفاده از زبان پایتون توسعه دادیم که در آن یک عامل هوش مصنوعی با استفاده از الگوریتم Minimax به بازی می‌پردازد. بازی را می‌توان در یکی از مود های زیر اجرا کرد:

- بازیکن در مقابل بازیکن
- بازیکن در مقابل ربات
- ربات در مقابل ربات

پس از اجرای برنامه (تابع Main) باید یک ورودی که تعداد ربات ها است وارد کنید. پس از اینکه تعداد ربات ها (می‌تواند ۰ یا ۱ یا ۲ باشد) را وارد کردید، باید قدرت ربات ها را به عنوان ورودی به صورت عددی وارد کنید. این قدرت همان عمق بررسی درخت حالات در الگوریتم Minimax است.

در بخش های بعدی در مورد کلاس‌ها، ویژگی آنها و متدهای پیاده سازی شده توضیحاتی ارائه شده است.

Piece.py

کلاسی که ویژگی‌ها و عملکرد قطعه‌های بازی چکرز را تعریف می‌کند.
ویژگی‌ها:

- **id**: شناسه قطعه.
- **color**: رنگ قطعه.
- **is_king**: وضعیت شاه بودن یا عادی بودن قطعه.

- **y** و **x**: موقعیت قطعه در تخته.

توابع:

- **__init__**: متد سازنده که ویژگی‌های قطعه را مشخص می‌کند.
- **crown**: تغییر وضعیت قطعه به شاه.

Color.py

این فایل یک enumeration است که رنگ‌های مختلف صفحه بازی و مهره ها را تعریف می‌کند و دو مقدار RED برای رنگ قرمز و BLUE برای رنگ آبی دارد.

Board.py

این فایل شامل کلاس "Board" برای انجام تنظیمات ابتدایی تخته بازی چکرز میباشد.

ویژگی‌ها:

- **pieces**: لیستی از شیء نوع **Piece** است.
- **matrix**: یک ماتریس 8x8 از مقادیر **None** است.

توابع:

- **__init__**: متد سازنده که ویژگی‌های تخته را مشخص می‌کند.
- **new_board**: برای ساختن ماتریس با اندازه 8x8 و قرار دادن قطعات اولیه بر روی تخته استفاده شده است. این تابع به ترتیب برای قطعات قرمز و آبی، موقعیت‌های مختلف را در تخته مشخص می‌کند و قطعات را به لیست **pieces** اضافه می‌کند.

Moves.py

این کلاس شامل توابعی است که حرکت‌های مختلف برای مهره ها را بررسی و اعتبارسنجی می‌کند.

ویژگی ها:

- **TL (Top Left): Tuple** :از دو عدد (-1, -1) نمایش دهنده‌ی جهت حرکت به سمت بالا و چپ است.
- **TR (Top Right): Tuple** :از دو عدد (-1, 1) نمایش دهنده‌ی جهت حرکت به سمت بالا و راست است.
- **BL (Bottom Left): Tuple** :از دو عدد (1, -1) نمایش دهنده‌ی جهت حرکت به سمت پایین و چپ است.

- **BR (Bottom Right): Tuple** :از دو عدد (1, 1) نمایش دهنده‌ی جهت حرکت به سمت پایین و راست است.

در این کلاس یک Instance از همین نوع ایجاد شده و بعداً از همین نمونه استفاده می‌کنیم.

State.py

کلاس **State** وضعیت فعلی بازی چکرز را نگهداری می‌کند و عملیات مختلفی مانند بررسی حرکات مجاز، حملات، و... را انجام می‌دهد

ویژگی‌ها:

- **Board** :board: تخته بازی که وضعیت فعلی بازی را نشان می‌دهد.
- **Color** :turn: نوبت فعلی: Color.RED یا Color.BLUE.
- **int** :red_pieces: تعداد مهره‌های قرمز باقی‌مانده.
- **int** :blue_pieces: تعداد مهره‌های آبی باقی‌مانده.
- **str** :last_move: آخرین حرکت انجام شده در بازی.
- **str** :last_piece_moved: شناسه آخرین مهره ای که حرکت کرده.

توابع:

- **print:** نمایش وضعیت فعلی تخته بازی با استفاده از کاراکتر های رنگی برای نمایش مهره‌ها.
- **legal_moves :**بازگرداندن یک لیست از حرکت‌های مجاز برای یک مهره داده شده.
- **legal_attacks:** بازگرداندن یک لیست از حرکت‌های حمله مجاز برای یک مهره داده شده.
- **move:** جابجایی یک مهره در تخته بازی.
- **attack :**اجرای حرکت حمله در تخته بازی.
- **is_empty_cell :**بررسی اینکه آیا یک سلول در تخته بازی خالی است یا نه.
- **is_opponent:** بررسی اینکه آیا یک سلول در تخته بازی شامل مهره حریف است یا نه.
- **is_in_board:** بررسی اینکه آیا مختصات در محدوده تخته بازی قرار دارند یا نه.
- **is_move_in_board:** بررسی اینکه آیا یک حرکت در محدوده تخته بازی برای یک مهره قابل انجام است یا نه.
- **next_coordinates:** محاسبه مختصات بعدی برای یک مهره پس از یک حرکت.
- **current_coordinates:** بازگشت مختصات فعلی یک مهره.
- **successor:** تولید یک لیست از وضعیت‌های جانشین ممکن.
- **heuristic:** ارزیابی مقدار هیوریستیکی از وضعیت فعلی برای یک رنگ داده شده

Minimax.py

این فایل شامل الگوریتم Minimax مرتبط بازی است که بر اساس آن حرکت‌های مناسب کامپیوتر برای بهبود عملکرد و انتخاب بهترین حرکت‌ها را تعیین می‌کند. این الگوریتم به صورت بازگشتی و با استفاده از یک تابع ارزیابی (هیوریستیک) و عمق داده شده، بهترین حرکت را انتخاب می‌کند.

- **minimax**: این تابع با استفاده از بازگشت، به صورت بازی حداکثر و حداقل، وضعیت‌های مختلف بازی را بررسی می‌کند.
- **state.successor**: این تابع stateهای جدید ممکن را از state فعلی محاسبه می‌کند. این state ها نتایج ممکن برای حرکت‌های مختلف در بازی هستند.
- **forward_pruning**: این تابع لیستی از stateهای جدید ممکن را از تابع successor دریافت کرده و با توجه به اینکه طول این لیست معمولا عدد بزرگی است (بطور میانگین 6.5) میزان شاخه شاخه شدن درخت را تعیین میکند. ما بطور پیش‌فرض این مقدار را 4 در نظر گرفتیم. می‌توان آن را تغییر داد.
- هیوریستیک (state.heuristic()): این تابع برای ارزیابی هر state مختلف استفاده می‌شود. در هر مرحله از جستجو، بررسی می‌شود که آیا عمق مورد نظر رسیده است یا خیر. سپس، بر اساس معیارهای مشخص، مقدار عددی برای امتیاز هر state ِبعدی تخمین زده می‌شود که به الگوریتم کمک می‌کند در انتخاب بهترین حرکت دقیق‌تر عمل کند.
- پارامترهای تابع **minimax**: عمق فعلی جستجو، وضعیت max کننده، مقادیر آلفا و بتا (برای الگوریتم آلفا-بتا) ، قدرت جستجو (یا عمق حداکثر) و همچنین رنگ بازیکنی که تابع را به نفع آن اجرا می‌کنیم، به عنوان ورودی‌های این تابع ارسال می‌شوند.
- انتخاب بهترین حالت: با استفاده از آلفا و بتا در الگوریتم هرس آلفا-بتا، انتخاب بهترین حالت در هر عمق انجام می‌شود. این کار ادامه یا قطع جستجو را با توجه به اینکه آیا حالت بهتری پیدا شده یا خیر، ادامه می‌دهد.

Main.py

این فایل شامل کد اصلی برنامه است که از سایر فایل‌ها استفاده می‌کند و بازی را شروع می‌کند، شامل قسمت‌هایی مانند شروع بازی، تعامل با کاربر و انتخاب حرکت‌ها.

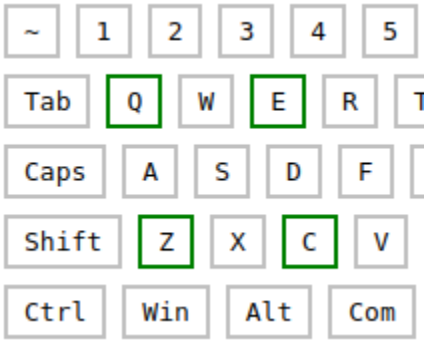
نحوه انجام حرکات توسط بازیکن آفلاین:

برای ورودی دادن به بازیکن باید یک استرینگ شامل دو کاراکتر با فاصله از هم استفاده کنید. ابتدا حرف نشان دهنده بازیکن (که از A شروع شده و تا L ادامه پیدا می‌کند و به بزرگی و کوچکی حروف حساس نیست) و حرف نشان دهنده حرکت به صورت زیر (برای درک بهتر به کیبوردتان نگاه کنید!) :

- Q: بالا چپ
- E: بالا راست

- Z: پایین چپ
- C: پایین راست

استفاده کنید.



ویژگی ها:

- **n_bots:** تعداد ربات ها را در بازی مشخص می‌کند. برای اینکه دو ربات را روبروی یکدیگر قرار دهید این مقدار را باید ۲ وارد کرده و در صورتی که می‌خواهید خودتان با ربات بازی کنید، این مقدار را ۱ وارد کنید. همچنین این قابلیت که بتوانید با دوست‌تان بازی کنید هم در نظر گرفته شده است! بدین منظور باید این عدد را صفر وارد کنید.

توابع:

- **check_end_game:** این متد پس از هر حرکت بررسی می‌کند که آیا بازی به پایان رسیده است یا خیر. این موضوع با استفاده از تعداد حرکات ممکن در عمق بعدی بررسی می‌شود. در صورتی که یک بازیکن حرکت معتبری برای انجام نداشته باشد (مهره هایش تمام شده باشند یا قفل شده باشند) بازی به اتمام می‌رسد.