به نام خدا

پروژه ی امتیازی درس هوش مصنوعی- جستجوی رقابتی

Quoridor

محمد واثق

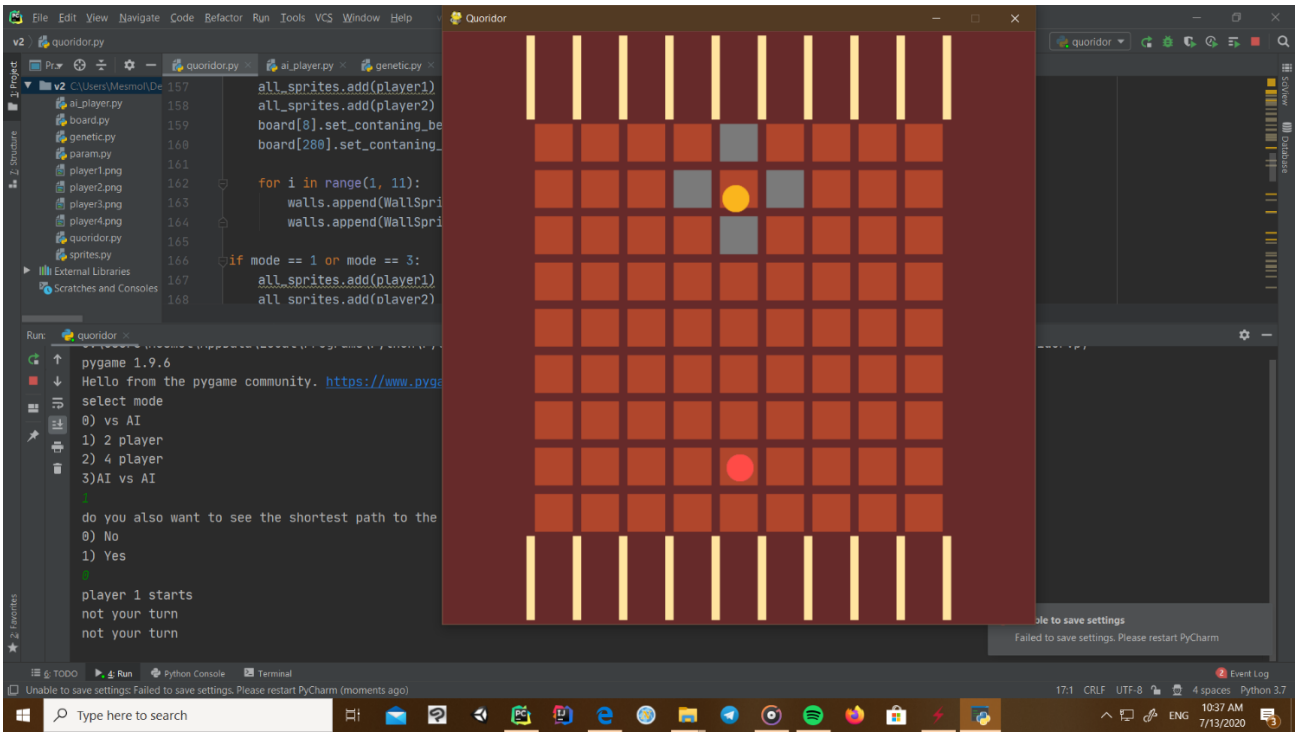
پیاده سازی و منطق بازی

زبان و نحوه ی پیاده سازی

استفاده از کتابخانه ی pygame و توابع موجود در موتور بازیسازی مربوطه برای پیاده سازی راحت تر گرافیکی

بدنه ی کلی و قوانین

در هر دو حالت دو نفره و چهار نفره ، قوانین مرتبط با بازی کوریدور پیاده سازی شده اند، در تصویر زیر خانه های مجاز برای نوبت بازیکن زرد مشخص شده:



پیش از شروع به کد زدن ، حتما قوانین این بازی رو مطالعه کنید ، می تونید از لینک انتهای این فایل استفاده کنید . قوانین بازی پیشاپیش پیاده سازی شدند ، اما ، برای طراحی یک تابع هیوریستیک خوب باید بازی رو خوب بشناسید .
تقریبا تمامی توابعی که نیاز دارید ، با اسامی مرتبط به اونها ، در فایل Game.py هست. کار شما از این بخش ها تشکیل میشه :

1) پیدا کردن یک تابع هیوریستیک خوب برای ارزشیابی گره های درخت مینی مکس ، پیشنهاد میکنم حتما قبل از شروع کار چند بار با همدیگه بازی رو انجام بدید (یه لینک بازی آنلاین هم براتون آخر این صفحات گذاشتم) ، هر چقدر بهتر این بازی رو بشناسید هیوریستیک بهتری میتونید ابداع کنید. ساختن هیوریستیک اینجا دو قسمت داره ، شناختن فاکتور های مهم و تعیین یک رابطه ی خطی بین اینها ، با ضرایب منطقی. کلاس heurostic makerمرتبط با ضرایب و تابع heuristicمربوط به ساختن رابطه ی خطیه .
2) تشکیل یک درخت مینی مکس با هرس مناسب ، طبق صحبت های گفته شده سر کلاس ، اکشن های زیاد و عمق بالا موجب کند شدن مینی مکس میشه ، باید اکشن های "منطقی" رو بررسی کنید ، بخصوص اینکه توی این بازی ، جدای از حرکت مهره ، حرکت دیوار ها هم مهمه. خیلی اهمیت داره چه اکشن هایی رو از اون تعداد بالای حرکت های قابل انجام به مینی مکس پاس بدید ، در نهایت درخت های منطقی تر و عمیق تر (قدرت پیش بینی بالا تر) نمره ی بیشتری میگیرند.
3) روند کار نهایی : هر زمان نوبت مهره ی ایجنت میشه ، استیت رو میگیره ، "بهترین حرکت " رو با استفاده از مینی مکس پیدا میکنه ، حرکتش رو انجام میده و تاثیر حرکت روی زمین مشخص میشه ، همین طور تا دوباره نوبتش بشه یا بازی تمام بشه.
4) تمامی توابع ممکن در دسترس شماست ، هر چقدر که این توابع و کلاس ها ، با توجه به مدیریت شما بهتر کنار همدیگه قرار بگیرند ، نمره ی بالاتری میگیرید . سعی کنید فایل های مرتبط با ایجنت ها و محیط رو با توابعشون خوب از همدیگه جدا کنید.
5) و در نهایت ، قسمت امتیازی این پروژه ی امتیازی 😊 برای پیدا کردن بهترین حالات ممکن برای ضرایب ، میتونید از الگوریتم ژنتیک استفاده کنید ، اگه کسی بخواد اینکار رو کنه ، با توجه به اینکه زمان هم محدوده ، من هم حسابی کمکش میکنم.
6) در نهایت من برای همه ی سوالات شما حاضر هستم ، همه ی ما در شرایط پر فشاری هستیم ، با وجود امتیازی بودن ، اگه کسی خواست پروژه رو انجام بده من هم تا جایی که بتونم به سوالاتش پاسخ میدم.

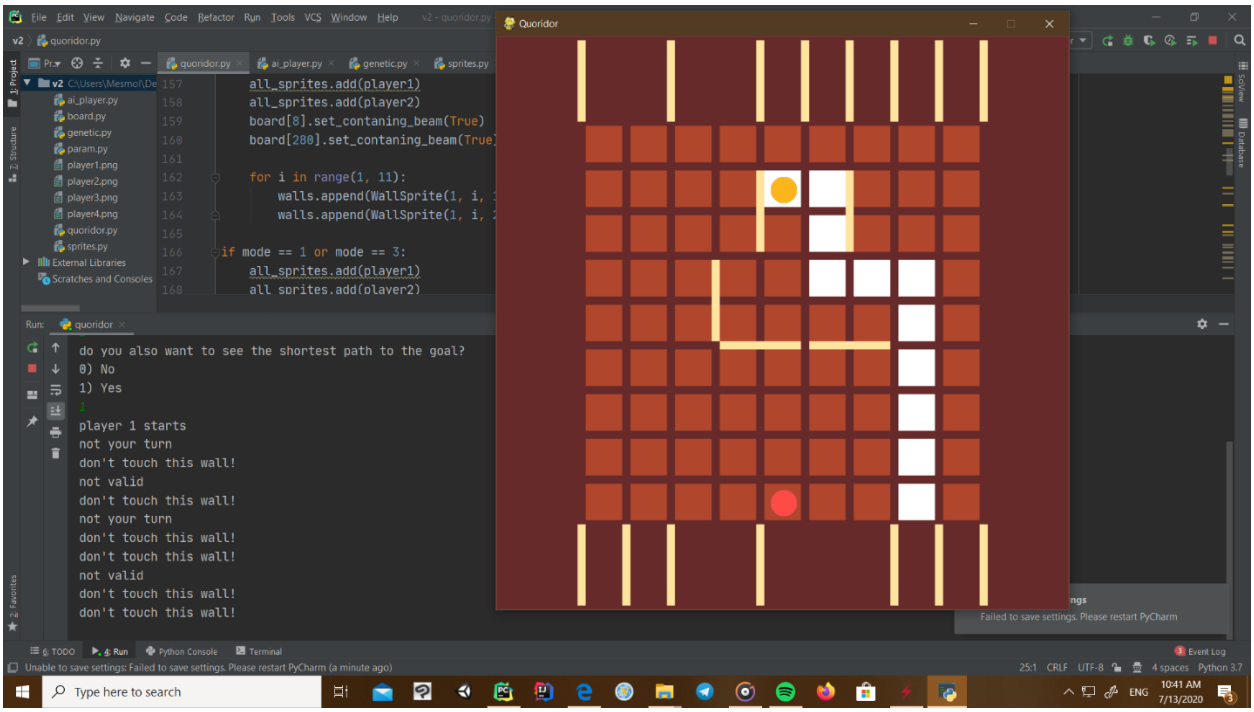
قوانین بلوکه شدن دیوار ها ، حرکت های مورب و مسدود شدن عادی طبق تصاویر بالا درست عمل می کنند، برای بررسی اینکه آیا امکان گیر کردن مهره وجود دارد و اینکه آیا دیواری که قرار میدهیم مسیر را کاملا میبندد یا خیر ، از الگوریتم BFS استفاده میکنیم.

به ازای حرکات مجاز پلیر شاخه ها درخت ایجاد می شوند و بررسی می شود که آیا پلیر به آخر خط خود می رسد یا خیر. در صورتی که دیوار گذاشتن در نقطه ای موجب گیر افتادن مهره شود، امکان قرار دادن دیوار در آن نقطه وجود ندارد.

راه فرار (کوتاه ترین مسیر به خط پایان) و الگوریتم BFS

```
def check_for_not_stulking(player, stateBoard):
    if win(player):
        return True
    exploredSet = set()
    exploredSet.add(player.get_player_number())
    frontier = [player]
    while True:
        if len(frontier) == 0:
            return False
        node = frontier.pop()

        for i in get_valid_moves(stateBoard, node):
            if not exploredSet.__contains__(i):
                exploredSet.add(i)
                temp = Player(player.get_player_number(), i, 2, stateBoard)
                if win(temp):
                    return True
                frontier.append(temp)
```



برد بازی و حرکات

هر خانه و هر دیوار بازی در برد ماتریس بازی دارای یک شماره است :

Number = 17 * row + column

میتوان پی برد که آیا شماره ی خانه ی مورد نظر مربوط به دیوار است یا خانه ای که column و rowبا توجه به فرمول گفته شده و بررسی مهره میتواند به آن برود (مثلا در صورتی که سطر و ستون هر دو عددی زوج باشند عدد مورد نظر قطعا مربوط به یک خانه ی عادی است.) جابجایی مهره ها (بالا ، پایین و چپ و راست رفتن) با کم کردن و زیاد کردن شماره خانه ی مورد نظر قابل انجام است.

moveUp = currentSquare + 2*(numberOfRows = 17)

moveDown = currentSquare - 2*(numberOfRows = 17)

moveLeft = currentSquare - 2

moveRight = currentSquare + 2

* margin همان فاصله ی میان خانه هاست

درخت مینی مکس و هوش مصنوعی

درخت مینی مکس

با توجه به صحبت های گفته شده ، هر اکشن برای بازیکن ها در واقع یک عدد است ، اگر عدد مورد نظر عدد خانه ی عادی باشد ، یعنی مهره باید از حالت فعلی به آن خانه جابجا شود و در صورتی که عدد مورد نظر مربوط به دیوار باشد ، با توجه به عمودی یا افقی بودن دیوار، آن خانه و خانه های اطرافش (در صورت عمودی بودن ، دو خانه ی پایینی و در صورت افقی بودن دو خانه ی سمت چپی) مسدود میشوند.

با استفاده از درخت مینی مکس و هرس آلفا بتا ، حرکات تا عمق دلخواه پیش بینی شده و با استفاده از تابع best Next move عامل متوجه می شود بهترین برای حرکت بعدی برای انجام بازی کدام است.

لینک ها :

https://www.youtube.com/watch?v=6ISruhN0Hc0

https://gorisanson.github.io/quoridor-ai/