Chess

می‌خواهیم تابعی بنویسیم که با استفاده از شیء گرایی یک شطرنج ساده پیاده‌سازی کند.

کریم که چشمانش را خون گرفته از شدت سر رفتن حوصله‌اش در دوران قرنطینه در حالی که قطعی اینترنت همچنان باقی‌ست تصمیم نهایی خود مبنی بر پیاده‌سازی و ساخت برنامه‌ی شطرنج گرفته است.

او از شما می‌خواهد برایش با استفاده از پایتون شطرنج را پیاده‌سازی کنید.

جزئیات

- این بازی در جدولی که از هر طرف نامتناهی‌ست انجام می‌شود. مهره‌ها به دو رنگ سفید و سیاه تقسیم شده‌اند و برای هر رنگ یک مهره‌ی شاه و به تعداد نامتنهای مهره‌ی سرباز که می‌توانند در صفحه حضور داشته باشند و می‌توانند هم حضور نداشته باشند داریم.
- در ابتدای بازی مهره شاه سفید در خانه $(-10, -10)$ و شاه سیاه در خانه $(10, 10)$ قرار دارد. و سربازی درون بازی وجود ندارد.
- نحوه‌ی پیروزی در این بازی به وسیله‌ی مات کردن حریف انجام می‌شود. مات کردن به این معنی‌ست که حداقل یکی از مهره‌های ما(به جز مهره‌ی شاه) در یکی از خانه‌های همسایه‌ی راسی(هشت خانه‌ی دور خانه‌ی مورد نظر) خانه‌ی شاه حریف باشد.
- برای پیاده‌سازی باید از دو کلاس `Piece` و `Board` استفاده کنید که جزئیات آن را مشاهده می‌کنید:

کلاس `Piece`

هر عضو این کلاس دارای سه ویژگی (*Attribute*) است که در تابع `-init-` مقداردهی اولیه می‌شود و به ازای هر شی مقدار آن فرق دارد:

۱. `sort` : که نشان دهنده‌ی نوع مهره‌ی مورد نظر می‌باشد. این مقدار یا `"K"` به معنای شاه و یا `"P"` به معنای سرباز است.

۲. `color` : که نشان دهنده‌ی رنگ مهره‌ی مورد نظر می‌باشد. این مقدار یا `"black"` به معنای رنگ سیاه و یا `"white"` به معنای رنگ سفید است.

۳. `position` : که نشان دهنده‌ی جایگاه مهره‌ی مورد نظر در صفحه می‌باشد به طور مثال این ویژگی برای مهره‌ی شاه سفید رنگ $(-10, -10)$ می‌باشد. نوع این متغیر *tuple* است.

کلاس `Board`

هر عضو این کلاس دارای یک ویژگی (*Attribute*) است که در تابع `-init-` مقداردهی اولیه می‌شود و به ازای هر شی مقدار آن فرق دارد:

- `position` : که به صورت یک دیکشنری می‌باشد که قسمت `Key` در این دیکشنری به تمامی *position* های اشغال شده در صفحه تعلق دارد و قسمت `Value` برای هر *position* یک عنصر از کلاس `Piece` می‌باشد که در آن *position* جای گرفته است. توجه کنید که شما باید در تابع `__init__` این کلاس شاه سفید و سیاه را با شرایطی که در بالا گفته شد به دیکشنری اضافه کنید.

این کلاس دارای چهار متد زیر می‌باشد:

۱. `add` : این متد به عنوان ورودی یک عنصر از کلاس `Piece` را می‌گیرد و آن را به صفحه اضافه می‌کند. توجه کنید که تنها باید یک شاه از هر رنگ در صفحه وجود داشته باشد و در *Position* مهره‌ای که به صفحه اضافه می‌کنیم نباید مهره‌ای وجود داشته باشد. اگر ورودی این متد در تناقض با این توضیحات بود متد باید عبارت `"invalid query"` را **چاپ** کند.

۲. `remove` : این متد به عنوان ورودی یک *position* دریافت می‌کند و اگر در آن *position* مهره‌ای وجود داشت آن مهره را از صفحه پاک می‌کند. توجه کنید که حتما از هر رنگ دقیقا یک شاه باید درون صفحه‌ی بازی قرار داشته باشد. اگر ورودی این متد در تناقض با این توضیحات بود و یا در *position* ورودی مهره‌ای وجود نداشت متد باید عبارت `"invalid query"` را **چاپ** کند.

۳. `move` : این متد به عنوان ورودی یک شی از نوع `Piece` و یک *Position* که نشان‌دهنده جایگاه جدیدی است که این مهره در آن باید قرار بگیرد، دریافت می‌کند. در صورتی که در جایگاه فعلی این مهره در صفحه، همین مهره وجود داشته باشد و جایگاه جدید نیز خالی از مهره باشد، این مهره را به آن جایگاه منتقل می‌کنیم. اگر در جایگاه فعلی این مهره در صفحه همین مهره وجود داشته باشد و جایگاه جدید یکی از مهره‌های سرباز حریف باشد، این مهره به آن جایگاه انتقال می‌یابد و مهره‌ی حریف از صفحه حذف می‌شود(چون حتما باید در هر لحظه از هر رنگ دقیقا یک مهره شاه وجود داشته باشد پس نمی‌تواند شاه حریف را مورد حمله قرار دهد). در غیر این صورت و یا اگر ورودی این متد در تناقض با این توضیحات بود متد باید عبارت `"invalid query"` را چاپ کند.

۴. `is_mate` : این متد به عنوان ورودی یک رنگ (`"white"` و یا `"black"`) را دریافت می‌کند و بررسی می‌کند که آیا مهره‌های رنگ مورد نظر در وضعیت مات قرار دارد یا خیر. اگر قرار دارد مقدار `True` و اگر خیر مقدار `False` را برگرداند.

نکات

- نام فایل پایتون شما باید `solution.py` باشد.
- می‌توانید فایل اولیه‌ی خام و کد تست نمونه را با استفاده از این لینک دانلود کنید.

قسمت آموزشی

در این قسمت راهنمایی‌های سوال به ترتیب در روزهای شنبه، دوشنبه و چهارشنبه ساعت ۱۸ اضافه می‌شود. مشکلات‌تان در راستای حل سوال را می‌توانید از بخش "سوال بپرسید" مطرح کنید.

▼ راهنمایی ۱

مقداردهی در تابع `-init-` به این صورت می‌باشد که برای هر شیء مقادیر مشخص و متفاوتی می‌توانیم تعیین کنیم. به طور مثال نوشتن تابع `-init-` برای کلاس *Piece* به صورت زیر می‌باشد:

```
def __init__(self, sort, color, position):
        self.color = color
        self.sort = sort
        self.position = position
```

و یا برای نوشتن این تابع برای کلاس *Board* در ابتدا قبل از اضافه کردن شاه‌ها به صفحه چون ویژگی `position` یک دیکشنری خالی‌ست باید به صورت زیر باشد:

```
def __init__(self):
        self.position = {}
```

برای متد `add` در کلاس *Board* باید چک کنیم در جایگاهی که می‌خواهیم مهره را *add* کنیم آیا مهره‌ای وجود دارد یا خیر اگر وجود دارد *invalid query* چاپ کنیم وگرنه در ویژگی `position` این *Board* باید این مهره اضافه شود.

برای متد `remove` در کلاس *Board* باید چک کنیم در این جایگاه مهره‌ای وجود دارد؟ و اگر وجود دارد این مهره شاه نباشد در غیر این صورت *invalid query* را چاپ کنیم در غیر این صورت مهره‌ای که در این جایگاه وجود دارد را از دیکشنری کلاس *Board* حذف کنیم.

برای متد `move` در کلاس *Board* باید چک کنیم اگر در *position* جدیدی که مهره باید به آنجا برود مهره‌ی همرنگ وجود داشت یا مهره‌ی شاه وجود داشت *invalid query* چاپ کند وگرنه اگر در جایگاه خالی‌ای می‌رود جایگاه الان مهره از دیکشنری *Board* حذف شود و جایگاه جدید مهره و این مهره در دیکشنری *Board* اضافه شوند و اگر در جایگاهی که می‌رود مهره‌ی حریف وجود داشت هم این مهره و جایگاهش و هم مهره‌ی حریف و جایگاهش از دیکشنری *Board* حذف شوند و مهره‌ی حرکت داده شده و جایگاه جدیدش در دیکشنری *Board* اضافه شوند.

برای متد `is_mate` در کلاس *Board* نیازمند به چک کردن هر هشت خانه‌ی دور شاه داریم تا بفهمیم که آیا حداقل یک مهره از مهره‌های سرباز حریف در این خانه‌ها وجود دارد یا خیر. دقت کنید اگر خانه‌ی شاه مورد نظر خانه‌ی (i,j) هشت خانه‌ی دور آن خانه‌های زیر است:

$$(i+1,j)\ ,(i+1,j+1)\ ,(i+1,j-1)\ ,(i,j+1)\ ,(i,j-1)\ ,(i-1,j+1)\ ,(i-1,j)\ ,(i-1,j-1)$$

▼ راهنمایی ۲

برای متد `add` در کلاس *Board* شبه کد زیر همان ترجمه‌ی راهنمایی یک به زبان پایتون است:

```
if piece.sort=="K" or piece.position in self.position:
    print("invalid query")
else:
    self.position[piece.position]=piece
```

برای متد `remove` در کلاس *Board* شبه کد زیر همان ترجمه‌ی راهنمایی یک به زبان پایتون است:

```
if position not in self.position:
    print("invalid query")
elif self.position[position].sort=="K":
    print("invalid query")
else:
    self.position.pop(position)
```

برای متد `move` در کلاس *Board* در راهنمایی ۳ شبه کد قرار می‌گیرد.

برای متد `is_mate` در کلاس *Board* در راهنمایی ۳ شبه کد قرار می‌گیرد.

▼ راهنمایی ۳

برای متد `move` در کلاس *Board* شبه کد زیر همان ترجمه‌ی راهنمایی یک به زبان پایتون است:

```
def move(self, piece, position2):
        if position2 in self.position:
            if self.position[position2].color == piece.color:
                print("invalid query")
            else:
                if self.position[position2].sort == "K":
                    print("invalid query")
                else:
                    del self.position[piece.position]
                    del self.position[position2]
                    self.position[position2] = piece
                    if piece.sort == "K":
                        if piece.color == "white":
                            global whitekingx, whitekingy, blackkingx, blackkingy
                            whitekingx = position2.x
                            whitekingy = position2.y
                        if piece.color == "black":
                            blackkingx = position2.x
                            blackkingy = position2.y
        else:
            del self.position[piece.position]
            self.position[position2] = piece
```

برای متد `is_mate` در کلاس *Board* شبه کد زیر همان ترجمه‌ی راهنمایی یک به زبان پایتون است:

```
def is_mate(self, color):
        if(color == "white"):
            for i in range(-1, 2):
                for j in range(-1, 2):
                    if i != 0 or j != 0:
                        position = (whitekingx + i, whitekingy + i)
                        if position in self.position:
                            return True
            return False
        else:
            for i in range(-1, 2):
                for j in range(-1, 2):
                    if i != 0 or j != 0:
                        if (blackkingx + i, blackkingy + i) in self.position:
                            return True
            return False
```