توضیحات پروژه

طریقه نحوه اجرا و بازی :

بازی همانطور که گفته شده بود در کنسول ران می شود که یعنی کافیست آن را یا از کنسول فراخوانی کنید و یا در همان محیط ide مثل intellij idea آن را ران کنید .

هر دو حالت بازی با کامپیوتر و بازی ۲ نفره پیاده سازی شده است .

شروع بازی با بازیکن مشکی است طبق قوانین سایت بازی آنلاین که لینکش گذاشته شده بود . در پروژه من player 1 بازیکنی با رنگ سفید است و player 2 بازیکنی با رنگ مشکی است و عدد بعد آن ها ربطی به اینکه کداوم شروع میکند ندارد و player 2 بازی را شروع می کند .

در ابتدا منویی برای شما باز می شود که مشخص می کند کدام حالت بازی را می خواهید ، کافیست در اینجا یک عدد که همان index آن موارد در منو است را وارد کنید . بعد بازی شروع می شود . در هر مرحله کافی است همانطور که در فایل دستور کار پروژه گفته شده بود ، ورودی مااند C l وارد کنید که l عدد و C کاراکتر است.

توجه شود که اگر مهره ای را در جاهای ممکن نگذارید (یعنی موقعیت دلخواه وارد شده محاصره ای انجام ندهد .) آن مهره به طور کلی چاپ نمی شود ، و از آن جایی که شما نوبت خود را از دست داده اید ، نوبت بازیکن بعدی می شود !

روند حل سوال :

- **توضیحات کلاس ها :**

من پروژه را به ۳ قسمت مهم تبدیل کردم . همانطور که در بازی مفاهیم بازیکن ، مهره ها و روندبازی و تغییرات نقشه مهم ترین اجزا هستند ، کلاس ها را به همین ترتیب تقسیم بندی کردم .

کلاس token همان مهره ی ماست که در آن یک عدد برای رنگش وجود دارد (توضیحات چاپ رنگ بعدا داده شده است)، همینطور مکانش که با x و y مشخص می شود و متود های لازمه .
کلاس player برای بازیکن هاست ، هر بازیکن لیستی از مهره هایش دارد و رنگ مخصوص به خود را دارد که یک عدد است .
کلاس playing system مهم ترین کلاس است . در آن اصلی ترین کار های بازی از نظیر چاپ نقشه ، چک کردن اینکه آیا بازی هنوز ادامه دارد یا نه ، گذاشتن یک مهره و آپدیت کردن نقشه و تشخیص دادن جاهایی که می توان مهره گذاشت ، انجام می شود .

- **روند انجام بازی :**

بازی با ساختن یک شی از کلاس playing system شکل می گیرد و player های لازم و token های اولیه گذاشته می شوند . برای سخت player بسته به اینکه رنگ player چی هست ، یک x و y به کانستراکتورش داده می شود تا ۲ token اول از آن رنگ را قرار دهد :

```
if (x == 4 && y == 4) {
    x2 = x + 1;
    y2 = y + 1;
} else {
    x2 = x - 1;
    y2 = y + 1;
}
```

پس از آن لوپ while داخل main که تا وقتی کسی برنده نشده است ران می شود ، شروع می کند و با استفاده از متود playTurn نوبت ها عوض می شوند . در هر نوبت مهره ای گذاشته می شود و با استفاده از متود checkAvailablePlaces جاهای ممکن چک می شود و اگر جای گذاشته شده جزو آن ها بود مهره گذاشته می شود و بعد اگر مهره هایی از رنگ های مخالف محاصره شده اند . برگردانده می شوند که اینها با

استفاده از متود های updateBoard و flip انجام می شوند . (توضیحات نحوه کار متود ها بعدا داده شده است .)

در روند بازی اگر هیچ جای ممکنی نبود و درواقع یعنی هیچ حرکتی برای یک بازیکن باقی نمانده بود ، همانطور که در فایل دستور کار گفته شده بود عبارت pass چاپ می شود و نوبت بازیکن بعدی می شود .

اگر بازیکنی برنده شد از لوپ خارج می شود و تعداد مهره های هرکدام و برنده را چاپ می کند.

چالش های مهم و حل آن ها :

- **چاپ رنگ ها :**

برای چاپ مهره ها با استفاده از یونی کد ها دو مهره یکی توپر به نشانه ی مشکی و یکی تو خالی به نشانه سفید انتخاب کردم . آن هارا در ابتدا در یک رشته ذخیره کردم:

```
String white = "25EF";
String black = "25CF";
```

سپس به یک عدد hexadecimal تبدیل کردم :

```
int whiteValue = Integer.parseInt(white, 16);
int blackValue = Integer.parseInt(black, 16);
```

در نهایت برای چاپ آن ها ، به char کست کردم :

```
(char) token.getColorValue()
```

- **پیدا کردن جاهای ممکن برای گذاشتن مهره :**

متود findAvailablePlaces در کلاس playingSystem این کار را انجام می دهد .

به طور کلی من از در لیست همه ی token ها تک به تک این عملیات را انجام میدادم :

۴ حالت داریم :۱. خط افقی

۲. خط عمود

۳.خط مورب حالت ۱ :

۴.خط مورب حالت ۲ :

متغیر های i و j مشخص می کنند که حرکت ما به چه سمت است ، i حرکت افقی و j حرکت عمودی است .

پس حالات خطوط گفته شده این مقادیر iوj را دارند:

```
//Straight line :
if (k == 0) {
    i = 1;
    j = 0;
}
//perpendicular line :
if (k == 1) {
    i = 0;
    j = 1;
}
//diagonal line from left down to right up :
if (k == 2) {
    i = 1;
    j = 1;
```

```
}
//diagonal line from right down to left up :
if (k == 3) {
    i = -1;
    j = 1;
}
```

هر ۴ حالت گفته شده برای هر token چک می شوند .

در هر بار ما در ابتدا به جلو میرویم و چک می کنیم که آیا به رنگ مخالفی میرسیم یا نه ، اگر رسیدیم flag مربوطه ۱ می شود :

```
while (checkTokenExistence(xForward, yUpper)) {
    if (getTokenColorValue(xForward, yUpper) ==
player1.getColorValue()) {
        //if we have a white token in front of it :
        flag1 = 1;
        break;
    }
    xForward += i;
    yUpper += j;

}
```

بعد برای عقب را چک می کنیم ، اگر به رنگ مخالف رسیدیم ، flag مربوطه ۱ می شود :

```
while (checkTokenExistence(xBackward, yLower)) {
    if (getTokenColorValue(xBackward, yLower) ==
player1.getColorValue()) {
        //if we have a white token in back of it :
        flag2 = 1;
```

```
            break;
        }
        xBackward -= i;
        yLower -= j;

    }
```

از آنجایی که برای گذاشتن مهره یک سمت خط باید رنگ مخالف داشته باشد و سمت دیگر همه از همان رنگ خود token باشد تا بتوانیم آنجا رنگ مخالف بگذاریم وtoken مورد بررسی را محاصره کنیم . پس کافیست یکی از flag ها ۱ باشد و دیگری صفر :

```
if (flag1 == 1 && flag2 == 0) {
    ArrayList<Integer> temp = new ArrayList<>();
    temp.add(xBackward);
    temp.add(yLower);
    availablePlaces.add(temp);
}
//if there is a token of white in back which surrounds them , we put
the white token
//in the other side of line(front) :
if (flag1 == 0 && flag2 == 1) {
    ArrayList<Integer> temp = new ArrayList<>();
    temp.add(xForward);
    temp.add(yUpper);
    availablePlaces.add(temp);
}
```

جاهای ممکن را در یک arraylist دو بعدی ذخیره می کنم که هر عضو آن یک arraylist است که عضو اول آن x است و عضو دوم آن y.

- **گذاشتن مهره و تغییر رنگ مهره های محاصره شده :**

این کار با استفاده از متود updateBoard انجام می شود .

این متود هم ۴ بار برای همان حالات خطوط گفته شده فراخوانی می شود ، و به آن موقعیت مهره ی جدید که گذاتیم را پاس می دهیم .

عملیات انجام شده حدودا مثل متود findAvailablePlaces است :

ابتدا token جدید گذاشته شده را پیدا می کنیم در لیست همه ی token ها . بعد می خواهیم چک کنیم آیا اطراف آن طبق جهت مشخص شده (با i و j) مهره هایی از رنگ مخالف با قابلیت محاصره شدن وجود دارند یا نه . روند چک کردن این است که ابتدا جلو و سپس عقب چک می شوند .

تا وقتی که مهره ای در جلو وجود دارد چک می کند که اگر رنگش از رنگ مخالف است مهره را در یک arraylist دو بعدی مثل قبل که توضیح دادم ذخیره می کند تا وقتی که به رنگ خود میرسد و flag مربوطه را ۱ می کند .

به عنوان مثال عملیات انجام شده برای چک کردن جلو برای بازیکن با رنگ های سفید :

```
while (checkTokenExistence(xForward, yUpper)) {
    //if the token is black:
    if (getTokenColorValue(xForward, yUpper) ==
player2.getColorValue()) {
        //number of black tokens between:
        blackTokensBetween++;
        //puts it in black between to be flipped later :
        ArrayList<Integer> temp = new ArrayList<>();
        temp.add(xForward);
        temp.add(yUpper);
        blackBetween.add(temp);
    } else if (getTokenColorValue(xForward, yUpper) ==
player1.getColorValue()) {
```

```
        //first time checking, the color is white and we want to
find the white token after the
        //first one , so we check if it's not the first time the
token is white :
        if (ctr > 0)
            flag = 1;
    }
    xForward += i;
    yUpper -= j;
    ctr++;
}
```

متغیر ctr برای این است که اولین بار خود مهره چک می شود و رنگی چون هم رنگ خودش است به این معنا نیست که به مهره ای هم رنگ خود در آن خط رسیده است پس من هر بار برای هر مهره چک می کنم که اگر بار اول است ، flag را ۱ نکن .

در نهایت چک می کند که اگر مهره ای محاصره شده است (اگر محاصره شده باشد تعداد آن ذخیره شده است .) و اگر flag ، ۱ باشد ، مهره های محاصره شده را flip می کند :

```
if (flag == 1 && blackTokensBetween > 0) {
    for (ArrayList<Integer> it : blackBetween) {
        flip(it.get(0), it.get(1));
    }
}
```

برای عقب هم دقیقا همین کار انجام می شود و روند همین است .

متود flip که در اینجا صدا زده می شود عملیات برگرداندن مهره که همان عوض کردن رنگ آن هاست را انجام می دهد ، که به راحتی با استفاده از متود changeColor در کلاس token انجام می شود .

- **چاپ نقشه:**

چاپ را با استفاده از متودی در کلاس playingSystem به نام printBoard انجام می دهیم .

روند کلی با استفاده از لوپ های for تو در تو است با این تفاوت که در هر بار چک می کنیم که اگر مهره ای با همان x و y وجود دارد ، مهره را چاپ کن .

توضیحات جزیی دیگر در جاواداک و کامنت های بین کد داده شده است .