بسم الله الرحمن الرحیم

گزارش پروژه هوش مصنوعی بازی اتللو

آرمان حسین‌مردی محمدمهدی پیروی

پروژه هوش مصنوعی اتللو حاضر، متشکل از سه فاز جدا ولی به‌هم‌پیوسته‌ی بازی کاربر با کاربر[1]، عامل هوش مصنوعی با کاربر[2] و عامل هوش مصنوعی با عامل هوش مصنوعی[3] می‌باشد.

ابتدا به بررسی بخش کاربر با کاربر می‌پردازیم:

در بخش کاربر با کاربر می‌بایست ابتدا زمین بازی و قوانین اولیه مربوط به آن را پیاده‌سازی نمود. اولین فایلی که باید برای به‌راه انداختن بازی اجرا نماییم، فایل جاوای GameWindow.java است. در بدنه تابع سازنده، شیئی از کلاس زمین بازی (PlayGround) به نام gp نمونه سازی[4] می‌شود. در این فایل تنظیمات اولیه مربوط به پنجره بازی انجام گردیده و همچنین یک زمین بازی به آن اضافه می-گردد. حال پرسش این است که این زمین بازی چگونه پیاده‌سازی شده و چگونه کار می‌کند؟

کلاس PlayGround دارای ویژگی‌هایی از جمله یک **ماتریس** دوبعدی از جنس عددصحیح[5] و یک متغیر به نام نوبت می‌باشد. علاوه بر آن، ماتریسی دیگر از جنس کلاس PlayGroundTile و دو متغیر برای نگهداری امتیاز طرفین از جنس JLabel و دو متغیر از جنس صحیح برای مجموع امتیازات دو طرف از ویژگی-های این کلاس است.

در این زمین اطلاعات دو بازیکن (چه از جنس کاربر و چه از جنس عامل هوش مصنوعی) درج می‌شود. خوب است ذکر شود، برای Player1 و Player2 که از جنس کلاس والد GamePlayer هستند، می توان دو نوع تابع سازنده از دو کلاس متفاوت AI و UserPlayer (که هردو فرزند کلاس GamePlayer هستند) را فراخواند. با آرگومان های تابع سازنده کلاس AI، شماره بازیکن(Mark)، حداکثر

---

1 User To User
2 AI To User
3 AI To AI
4 Instantiate
5 Int (Integer)

عمق درخت مین‌مکسی که برای تصمیم‌گیری پیمایش می‌کند(Depth) و این‌که آیا این بازیکن بازی را اول شروع می‌کند یا نه؟(FirstPlayer) مشخص می‌شود. تابع کلاس UserPlayer نیز یک آرگومان برای تعیین شماره بازیکن دارد.

علاوه بر این‌ها ما به دو کنترل‌کننده زمان برای طرف های بازی نیاز داریم. لذا از دو متغیر از جنس Time به نام playerHandlerTime استفاده می نماییم.

سپس در تابع سازنده کلاس زمین بازی مقادیر و اندازه‌های گرافیکی مربوط به پنجره زمین بازی مثل تعداد سطرها و ستون‌های زمین بازی، طول و عرض پنجره بازی و رنگ‌ها تنظیم می‌شوند.

تک‌تک سلول‌هایی که از جنس PlayGroundTile بودند، نمونه‌سازی (Instantiate) می‌شوند.

در فرایند Instantiation کنترل‌کننده‌های زمان، هربار با تأخیر ۱۰۰۰ میلی‌ثانیه (یک ثانیه)، سه تابع فراخوانی می‌شود.

```
player1HandlerTimer = new Timer( delay: 1000,(ActionEvent e) -> {
    handleAI(player1);
    player1HandlerTimer.stop();
    manageTurn();
});

player2HandlerTimer = new Timer( delay: 1000,(ActionEvent e) -> {
    handleAI(player2);
    player2HandlerTimer.stop();
    manageTurn();
});
```

تابع handleAI()، حرکات غیرمجاز بازیکن را بررسی کرده و در صورت وجود اخطار می‌دهد. همچنین مکان حرکت بازیکن را چاپ کرده، بورد جدید را بروزرسانی نموده و نوبت فعلی را تغییر می‌دهد.

در تابع دوم، یعنی تابع ()stop از کنترل‌کننده زمان، بازیکن متوقف می‌شود تا طرف دیگر کارش را انجام دهد.

تابع ()manageTurn، دو حالت کلی را بررسی می‌نماید. اگر هیچ بازیکنی، هیچ حرکتی را نتواند در زمین بازی انجام دهد، بازی پایان یافته و علاوه بر چاپ این خبر، اطلاعات و امتیازات، از روی بورد و ماتریس بروزرسانی می‌شود. ضمناً با معرفی برنده به کمک کلاسBoardHelper مجموع امتیازات او یک واحد اضافه می‌شود. اما اگر بازیکنی امکان حرکت و بازی داشته باشد، علاوه بر بروزرسانی اطلاعات، باید وضعیت بازیکنی که نوبتش هست را بررسی کند. اگر جای حرکت داشت و یک عامل هوش مصنوعی بود، دوباره همان نخ[6] که سه تابع داشت شروع می شود ولی اگر یک کاربر معمولی بود باید منتظر کلیک او بمانیم. حال اگر بازیکنی که نوبتش هست جای حرکت نداشت، پس از چاپ *عبارت بازیکن/عامل حرکت مُجازی ندارد!* ، نوبت را تغییر داده و دوباره همین تابع ()manageTurn را صدا می زنیم. یعنی دوباره دو حالت کلی را بررسی می‌نماییم و ...

تاکنون کلیاتی را از اولین بخش پروژه و پایه ساخت بازی اتللو بررسی کردیم که این می تواند یک بازی کاربر با کاربر را تا اینجای کار برای ما فراهم سازد.

اکنون برای تجربه‌کردن بازی با یک عامل هوشمند مصنوعی، نیاز است تا یکی از مقدمات و نیازهای مهم یک عامل را فراهم کنیم . . . درخت مین‌مکس!

[6] Thread

```java
public class Minimax {

    static int nodesExplored = 0;

    //returns max score move
    public static Point solve(int[][] board, int player, int depth, Comparator e){
        nodesExplored = 0;
        int bestScore = Integer.MIN_VALUE;
        Point bestMove = null;
        for(Point move : BoardHelper.getAllPossibleMoves(board,player)){
            //create new node
            int[][] newNode = BoardHelper.getNewBoardAfterMove(board,move,player);
            //recursive call
            int childScore = MMAB(newNode,player, depth: depth-1, max: false,Integer.MIN_VALUE,Inte
                if(childScore > bestScore) {
                bestScore = childScore;
                bestMove = move;
            }
        }
        System.out.println("Nodes Explored : " + nodesExplored);
        return bestMove;
    }
```

پس از پیاده‌سازی درخت مینیمکس و همچنین هرس آلفا و بتا، تصمیم داریم تا از توابع شهودی و هیوریستیک جهت کاهش بار پردازشی عامل هوش مصنوعی استفاده نماییم.

توابع هیوریستیک با تخمین زدن شرایط، امتیاز و وضعیت نسبی کار عامل هوشمند را آسان‌تر می‌نمایند. برای مثال، حرکت بازیکن به خانه‌های گوشه(کنج) از نظر استراتژیک امتیاز و وزن تصمیم‌گیری بیشتری را خواهد داشت.

ما در تابع شهودی و هیوریستیک، ۶ مؤلفه و فاکتور مهم در ارزیابی تقریبی را مورد بررسی قرار دادیم که اندازه‌گیری آن‌ها سخت نیست ولی با این حال بسیار راهگشا خواهد بود.

فاکتورهای شهودی:

۱. پویایی
۲. میزان مرز
۳. مهره‌ها
۴. جای‌گیری‌ها

۵. ثبات
۶. گوشه‌ها و گرفتن آن

اندازه‌گیری این ویژگی‌ها باعث سرعت و سهولت بیشتر در تصمیم گیری می‌شود.

امتیاز کلی کاربر به میزان هر یک از فاکتورهای بالا وابسته است، اما هر کدام از فاکتورهای بالا ضریبی متفاوت دارند چون نقش و وزن آن ها یکسان نیست.

پویایی[۷]**:** عامل پویایی به میزان نسبت حرکت های قابل انجام فرد به رقیب بستگی دارد.

میزان مرز[۸]**:** هر چه‌قدر مربع‌های خالی اطراف مهره‌ها بیشتر باشد، مقدار این فاکتور بیشتر می‌شود.

مهره‌ها[۹]**:** نسبت تعداد مهره‌های فرد به حریف یکی از عوامل مؤثر در تصمیم‌گیری است.

جای‌گیری[۱۰]**:** بر طبق تجربه‌های طولانی‌مدتِ به‌دست‌آمده، به هر یک از ۶۴ خانه زمین بازی، یک عدد نسبت داده‌خواهدشد که اگر فرد روی آنها حرکت کند، احتمال موفقیت بیشتر خواهد داشت. این عامل نیز با وزن مشخصی در تصمیم‌گیری برای حرکت بعدی تأثیر خواهدداشت.

ثبات[۱۱]**:** چهار گوشه زمین بازی، ثابت(stable) حساب می‌شوند، زیرا وقتی به رنگ سیاه یا به رنگ سفید درآیند، تغییر رنگ پیدا نمی‌کنند. هرچقدر به این نقاط ثابت نزدیک‌تر شویم، مقدار این عامل بیشتر می‌شود.

---

7 Mobility
8 frontier
9 pieces
10 placement

گرفتن گوشه‌ها[12]**:** اگر فرد، موفق به گرفتن یکی از گوشه‌ها شود امتیاز خوبی را دریافت خواهد کرد، لذا این یکی از فاکتور های تاثیرگذار می‌باشد.

```
public int eval(int[][] board , int player){
    int score = 0;

    int[] weights = weightSetForDiscCount[BoardHelper.getTotalStoneCount(board)];

    if(weights[0] != 0) {
        score += weights[0] * mobility(board,player);
    }
    if(weights[1] != 0) {
        score += weights[1] * frontier(board,player);
    }
    if(weights[2] != 0) {
        score += weights[2] * pieces(board,player);
    }
    if(weights[3] != 0) {
        score += weights[3] * placement(board,player);
    }
    if(weights[4] != 0) {
        score += weights[4] * stability(board,player);
    }
    if(weights[5] != 0) {
        score += weights[5] * cornerGrab(board,player);
    }

    return score;
}
```

کتاب Opening ها

[12] cornerGrab

Opening Book ها مجموعه‌ای از حرکت‌هایی به عنوان حرکت‌های خوب هستند که اگر هر کدام از آن‌ها اجرا شود بازی به نفع آن فرد خواهد شد.

```
public class OpeningBook {

    private Random rnd = new Random();
    String[] openings = {
            "C4e3F6e6F5g6E7c5",
            "C4e3F6e6F5g6",
            "C4e3F6e6F5c5F4g6F7g5",
            "C4e3F6e6F5c5F4g6F7d3",
            "C4e3F6e6F5c5F4g6F7",
            "C4e3F6e6F5c5F4g5G4f3C6d3D6b3C3b4E2b6",
            "C4e3F6e6F5c5F4g5G4f3C6d3D6",
            "C4e3F6e6F5c5D6",
            "C4e3F6e6F5c5D3",
            "C4e3F6e6F5c5C3g5",
            "C4e3F6e6F5c5C3c6D6",
            "C4e3F6e6F5c5C3c6D3d2E2b3C1c2B4a3A5b5A6a4A2",
            "C4e3F6e6F5c5C3c6",
            "C4e3F6e6F5c5C3b4D6c6B5a6B6c7",
            "C4e3F6e6F5c5C3b4",
            "C4e3F6e6F5c5C3",
            "C4e3F6e6F5",
            "C4e3F6b4",
            "C4e3F5e6F4c5D6c6F7g5G6",
            "C4e3F5e6F4c5D6c6F7f3",
            "C4e3F5e6F4",
            "C4e3F5e6D3",
            "C4e3F5b4F3f4E2e6G5f6D6c6",
            "C4e3F5b4F3",
            "C4e3F5b4",
            "C4e3F4c5E6",
            "C4e3F4c5D6f3E6c6",
            "C4e3F4c5D6f3E6c3D3e2D2",
```

در تابع getMoveFromOpeningBook() حرکاتی که بازیکن تابه‌کنون داشته، با رشته‌های openingBook مقایسه شده و اگر برابر نبود، متغیر isMatch غیرفعال است.

اگر مَچ بودند، حرکت موردنظر به مجموعه حرکات موجود اضافه می‌شود.

در آخر نیز اگر حرکت موجودی بود، یکی از آن‌ها را به‌صورت تصادفی و رندوم به عنوان حرکت انتخابی برمی‌گردانیم.