گزارش پروژه نهایی هوش مصنوعی

امین زینالی
دانشکده ریاضی،آمار و علوم کامپیوتر
دانشگاه تهران

۱ معرفی بازی

در این پروژه می‌خواهیم برای بازی connect 4 یک Agent طراحی کنیم.این بازی یک بردگیم دو نفره است که در آن بازیکنان به نوبت دیسک های خود را در یک جدول $7 * 6$ به صورت عمودی می‌اندازند.اولین کسی که به یک خط افقی یا عمودی یا مورب از چهار دیسک برسد برنده می‌شود.

۲ پیاده سازی

ابتدا از کاربر می‌خواهیم سایز جدول بازی را تعیین کند کافیست تعداد سطر ها و تعداد ستون ها را با یک فاصله به الگوریتم ورودی دهید

درگام دوم از کاربر می‌خواهیم که یکی از حالت های زیر را انتخاب کند:

- Player vs Player
- AI vs Player
- AI vs AI

در حالت بازیکن در مقابل بازیکن صرفا از بازیکن ها ورودی می‌گیریم و به جدول اضافه می‌کنیم و نیازی به انجام کار دیگری نداریم.

در حالت بازیکن مقابل هوش مصنوعی به نوبت یک بار از بازیکن و بار دیگر از الگوریتم Minimax حرکت ها را دریافت کرده و در جدول بازی اعمال می‌کنیم تا اینکه بازی به پایان برسد.

در حالت بعدی دو الگوریتم باهم دیگر بازی می‌کنند و هر کدام به نوبت و با توجه به وضعیت جدول بازی و خروجی الگوریتم Minimax حرکت های خود را انجام می‌دهند تا اینکه بازی به پایان برسد.

۱.۲ تابع **Heuristic**

محاسبه تابع Heuristic از دو بخش تشکیل شده است:

- محاسبه مقدار تابع برای ستون وسط
- محاسبه مقدار برای بقیه خانه های جدول

دلیل این کار این است که ستون وسط در برد این بازی نقش مهمی دارد بنابراین ما هم وزن این ستون را در نتیجه نهایی بیشتر کردیم.

در فایل کد آپلود شده تابعی به نام evaluate_state داریم که یک آرایه چهارتایی را به عنوان ورودی می‌گیرد همان طور که قبلا گفتیم برای برنده شدن در این بازی لازم است چهار مهره بازیکن پشت سرهم در یک خط عمودی یا افقی و یا مورب قرار بگیرد.

تابع ذکر شده به این صورت عمل می‌کند:

```
if count of PLAYER_PIECE in array equals 4 then
    return 100
else if count of PLAYER_PIECE in array is 3 and other element is EMPTY then
    return 60
else if count of PLAYER_PIECE in array is 2 and other elements are EMPTY then
    return 20
if count of OPPONENT_PIECE in array is 3 and other element is EMPTY then
    return -50
```

با این توضیحات هرچقدر Agent به پیروزی نزدیک می‌شود امتیاز بیشتری می‌گیرد ولی تابع بالا برای یک چهارتایی پاسخ می‌دهد نه یک جدول بنابراین باید جدول بخ چهارتایی های افقی و عمودی و مورب تقسیم بندی میکنیم و مقدار تابع ذکر شده را برای هر کدام محاسبه و مجموع آن ها را به عنوان نتیجه برمیگردانیم.

۲.۲ الگوریتم **Minimax**

این الگوریتم را به صورت بازگشتی در تابع minimax پیاده می‌کنیم. این تابع در ورودی صفحه بازی و عمقی که باید برود و همچنین مقادیر آلفا و بتا را می‌گیرد ورودی آخر مشخص می‌کند در این عمق باید کمینه کند یا بیشینه. شبه کد الگوریتم به این صورت است:

```
// exit condition
if board.is_terminal is True then
   check winners
if depth == 0 then
    heuristic(board)

if maximize then
    find empty column
    add a piece to that column
    calculate new score of board by recursion
    update alpha
    handle Beta pruning

else then
    find empty column
    add a piece to that column
    calculate new score of board by recursion
    update beta
    handle alpha pruning
```

ابتدای الگوریتم شرط خروج را بررسی می‌کنیم اگر بازی به اتمام رسیده باشد و یا عمق داده شده صفر باشد لازم نسیت کار را ادامه دهیم و برمی‌گردیم

۳.۲ برنامه کلی

حالا قطعه های لازم برنامه را جداگانه نوشتیم در این جا می‌خواهیم برای کامل شدن برنامه آن ها را به همدیگر وصل کنیم. در حالت کلی برنامه به صورت زیر است:

```
init pygame attributes
game_over = false
while not game_over:
 if MOUSEBUTTONDOWN happens then:
    if PLAYER.is_turn():
        apply player move
 if AI.is_turn()
    apply AI move based on Minimax
```

draw board with pygame