Minimax Search:

برای پیاده‌سازی این قسمت، ابتدا حرکت‌های امکان پذیر را به صورت بازگشتی طی میکنم و یک درخت میسازم (به عمق h) سپس از عمق h شروع میکنم اگر بازیکن سیاه باشم تعداد حرکات باقیمانده بازیکن سفید را حساب میکنم و آن را در منفی یک ضرب میکنم (برای جلوگیری از استفاده از Boolean اضافی) و هرکدام که به صفر نزدیکتر بود را به عنوان حرکت برتر آن شاخه برمیدارم. برای سفید دقیقا برعکس است (یعنی در منفی یک ضرب نمیکنم) تعداد حرکاتی که سیاه برایش مانده است را محاسبه میکنم و باز هم آنهایی که به صفر نزدیکتر است را به عنوان حرکت برتر برمیگردونم. => اگر در آخرین عمق ما سیاه باشیم و حریفمان سفید باشد و حرکتهای سفید در یک شاخه برابر با منفی ۵ و منفی ۱۰ و منفی ۱۵ باشد، در اینجا حرکت منفی ۵ برگردانده میشود زیرا در این حالت مهره سفید کمترین تعداد حرکات باقی مانده را دارد => احتمال باختش بیشتر میشود.

Alpha-Beta Search:

برای پیاده‌سازی این قسمت، دو پارامتر alpha و beta را به تابع alphaBeta (که مشابه تابع minimax در الگوریتم Minimax است) در کلاس AlphaBetaPlayer اضافه کردم که آلفا نشان‌دهنده‌ی بیشترین مقدار utilityپیدا شده برای بازیکن سیاه و بتا کمترین مقدار utility پیدا شده برای بازیکن سفید است. حال اگر در پیمایش یک نود، (برای مثال برای آلفا) مقدار یوتیلیتی یکی از برگ‌های آن نود از آلفا کمتر باشد، می‌توانیم از پیمایش سایر برگ های آن نود صرف نظر کنیم چرا که در این حالت نود مینیموم مقدار برگ‌های خود را به عنوان یوتیلیتی انتخاب می‌کند و این مقدار عددی کمتر از بهترین یوتیلیتی پیدا شده برای آن نود (آلفا) می‌باشد؛ پس بنابراین پیمایش سایر برگ‌های آن نود تاثیر مثبتی نخواهد داشت.

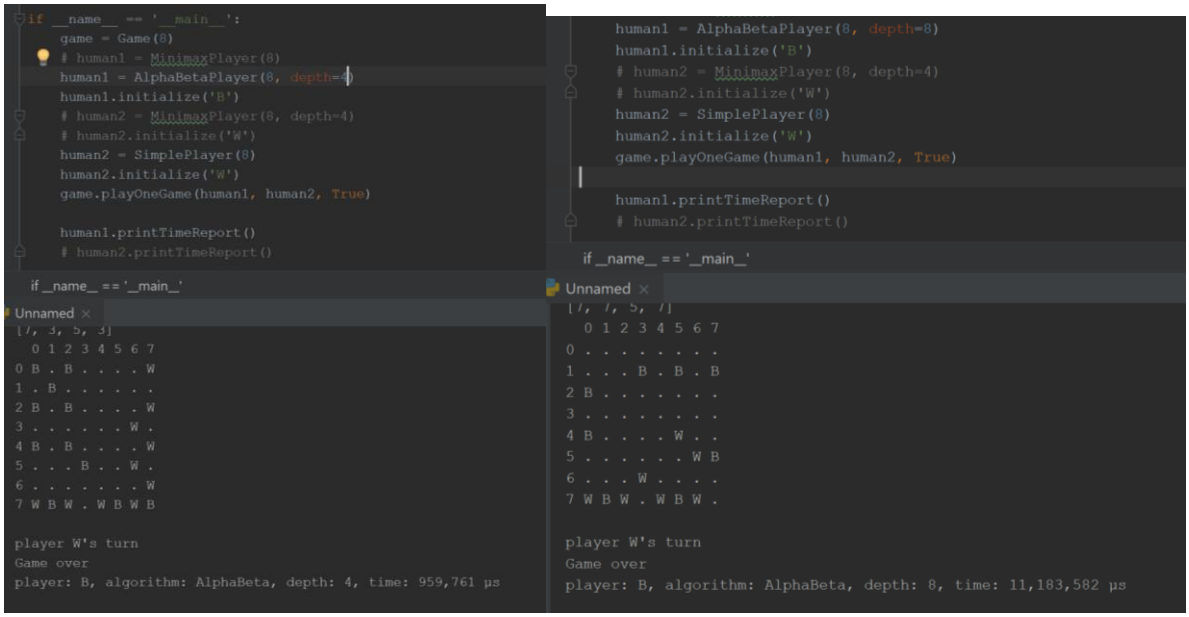
تابع evaluation function :

در هر شاخه کمترین تعداد حرکاتی که برای رقیب باقیمانده است را حساب میکنم. یعنی اگر الان نوبت مهره سیاه باشد و حرکاتش باعث شود به ترتیب مهره سفید ۳ و ۵ و ۶ حرکت برایش باقی مانده باشد، من حرکتی را انتخاب میکنم که ۳ حرکت برای مهره سفید باقی بماند.

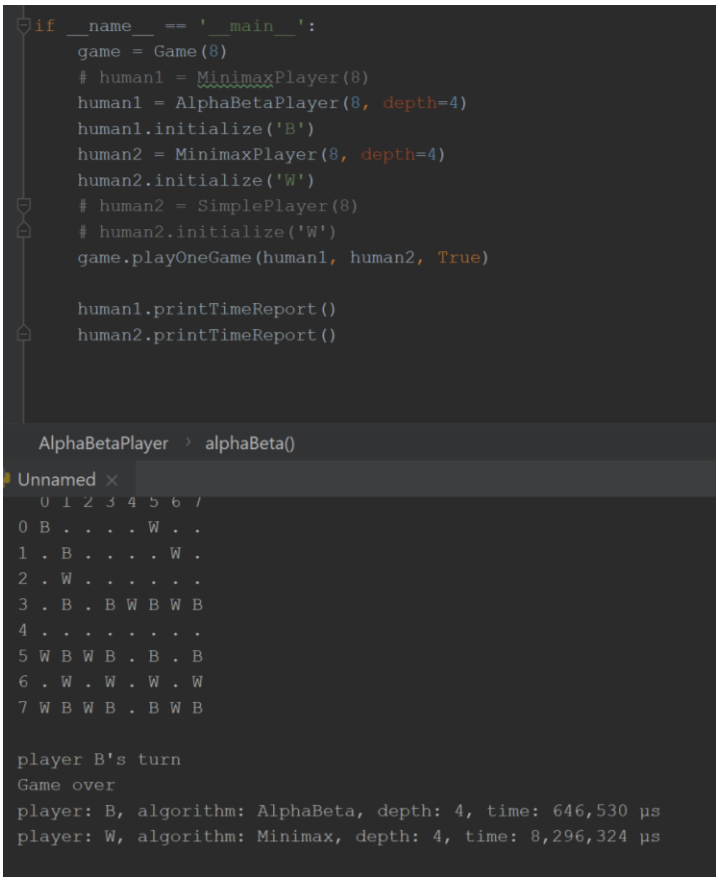
۲) خیر- اگر عمق آنها با هم یکی باشد حرکات آن‌ها با هم فرقی ندارد. زیرا تنها کاری که در الگوریتم alpha-beta انجام میدهیم این است که حرکاتی که به اندازه حرکات اصلی خوب نیستند را حذف میکنیم، نه اینکه حرکات درست تا آن عمق را حذف کنیم.

۳) قبل از پیاده‌سازی این روش، در صورتی که من با عمق ۴ می‌خواستم جستجو کنم، زمانی حدود ۴ ثانیه طول می‌کشید اما زمانی که این بهبود را انجام دادم، زمان برای هر قدم در عمق ۴ به زیر ۱ ثانیه کاهش یافت و در عمق ۸، تازه به بیشینه زمان ۴ ثانیه نزدیک می‌شد.

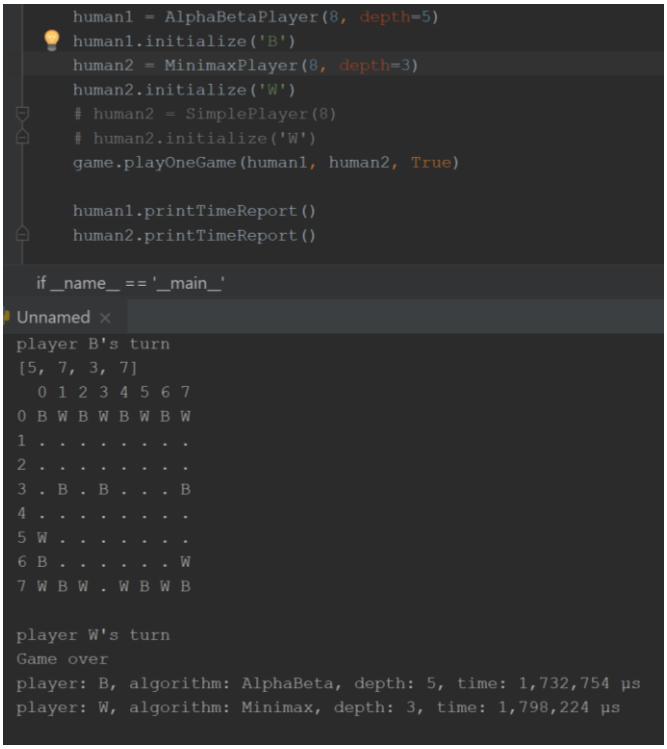
بازیکنی که الگوریتم alpha-beta را تا عمق ۸ اجرا میکند در صورت بازی کردن با یک بازیکن ساده ۱۱ ثانیه طول میکشد و با عمق ۴ حدود ۱ ثانیه.



در صورتی که یک بازیکن با الگوریتم آلفا بتا تا عمق ۴ و یک بازیکن با الگوریتم مین مکس بازی کند: برای بازیکن مین مکس =



درصورتی که عمق‌هایی که جستجو میکنند فرق کنند بازیکن آلفا بتا برنده میشود (در صورتی که عمق بیشتر را جستجو کند)


human1 = AlphaBetaPlayer(8, depth=5)
human1.initialize('B')
human2 = MinimaxPlayer(8, depth=3)
human2.initialize('W')
# human2 = SimplePlayer(8)
# human2.initialize('W')
game.playOneGame(human1, human2, True)
human1.printTimeReport()
human2.printTimeReport()
if __name__ == '__main__'
Unnamed
player B's turn
[5, 7, 3, 7]
0 1 2 3 4 5 6 7
0 B W B W B W B W
1 . . . .
2 . . . .
3 . B . B . . . B
4 . . . .
5 W . . . .
6 B . . . . . . W
7 W B W . W B W B
player W's turn
Game over
player: B, algorithm: AlphaBeta, depth: 5, time: 1,732,754 µs
player: W, algorithm: Minimax, depth: 3, time: 1,798,224 µs