گزارش پروژه دوم هوش مصنوعی
آیلین جمالی - ۸۱۰۸۹۵۰۲۶

در این پروژه ابتدا کلاسی به نام MinimaxPlayer ساختم که از کلاس‌های Game و Player ارثبری می‌کرد.

دو تابع inintialize و getMove آن را خودم با توجه به خواسته‌های توی صورت پروژه پیادسازی کردم.
ابتدا در تابع initialize نام agent را Minimax گذاشته و نقش آن را در side مشخص کردم. (سیاه یا سفید)

برای اینکه الگوریتم minimax را پیادسازی کنم، نیازمند سه تابع دیگر بودم:

minValue
maxValue
evalFunc

تابع minValue، نودهای مینیمم‌کننده و تابع maxValue، نودهای ماکسیمم‌کننده را محاسبه می‌کنند. به این صورت که در ابتدا تمام حرکاتی که هر دو طرف بازی می‌توانند انجام دهند را با استفاده از تابع generateMoves بدست آوردم و سایز آن‌ها را گرفتم. سپس الگوریتم بازگشتی minimax را پیادسازی کردم. شرط پایان این الگوریتم این است که یکی از طرفین دیگر حرکتی برای انجام دادن نداشته باشد (یعنی تابع generateMoves آرایه‌ای خالی برگرداند). در این صورت برای نود ماکسیمم‌کننده عدد منفی بی‌نهایت و برای نود مینیمم‌کننده عدد مثبت بی نهایت برمی‌گردانم. همچنین یک آرایه‌ی خالی نیز برمی‌گردانم به عنوان حرکتی که آن طرف بازی می‌تواند انجام دهد.

از آنجایی که زمین بازی ممکن است آنقدر بزرگ شود که نتوان حرکات را تا انتها پیش رفت و بهترین حرکت را انتخاب کرد، تا عمق خاصی از بازی پیش می‌رویم و بر اساس یک evaluation function، در انتهای عمق انتخاب شده پیش‌بینی می‌کنیم که مقدار امتیاز هرکسی اگر این نود را انتخاب کند تا انتهای بازی چند می‌شود. سپس این مقدار را برمی‌گرداند.

تابع evalFunc این کار را انجام می‌دهد. جمع ضریبی از تعداد حرکات هر نود و تعداد مهره‌هایی که برای حرکت دارد منهای همین عدد برای حریفش را برمی‌گرداند.

در صورتی که به عمق مورد نظر نرسیده بودم، در یک حلقه به ازای هر حرکتی که می‌توانم انجام دهم، زمین بازی را مطابق حرکت انتخاب شده تغییر می‌دهم و این زمین فرضی را برای حریف مقابل (نود ماکسیمم‌کننده یا مینیمم‌کننده) می‌فرستم. پارامترهایی که پاس می‌کنم زمین و مقدار عمقی است که در آن قرار دارم.
بعد از اینکه لایه‌های مورد نظر را پایین رفتم و می‌خواستم برگردم، برای اینکه عمق در هر لایه برای حرکت بعدی ست شود نیازمند اضافه کردن به عمق هستم. سپس بزرگترین مقدار برگردانده شده و حرکتی که باعث ایجاد آن مقدار شده را نگه می‌دارم و از تابع max_value و min_value برمی‌گردانم.

برای اینکه agent ما برنده شود، اولین بار تابع max_value را در متود getMove صدا می‌کنیم و حرکتی که این تابع برمی‌گرداند را به متود playOneGame برمی‌گردانیم.

زمانی که alpha-beta pruning را نیز پیادسازی می‌کنیم، سرعت اجرای الگوریتم خیلی بالاتر می‌رود. این الگوریتم به این صورت کار می‌کند که در نود ماکسیمم‌کننده زمانی که به نودی می‌رسیم که مقدار آن کمتر از یک بتایی باشد، دیگر نیاز نیست بقیه‌ی بچه‌ها را چک کنیم و همان را می‌توانیم برگردانیم. چون در نود مینیمم‌کننده قطعا بچه‌های بزرگتر از آن عدد را انتخاب نمی‌کند. و همین الگوریتم را برای نود مینیمم‌کننده با عدد آلفا تکرار می‌کنیم. با این تفاوت که بتا در نود مینیمم‌کننده مقداردهی می‌شود (مینیمم مقداری که ارزش نودها می‌توانند داشته باشند) و آلفا در نود ماکسیمم‌کننده مقداردهی می‌شود (ماکسیمم مقداری که ارزش نودها می‌توانند داشته باشند).

۱. تابع evaluation function در بالا توضیح داده شده است.
۲. بستگی دارد که چه agentهایی با هم بازی می‌کنند. اما اگر شرایط یکسان باشد و مثلا دو minimax agent با یکدیگر بازی کنند که در بازی اول pruning نداشته باشند و در بازی دوم pruning داشته باشند، حرکت‌هایی که انتخاب می‌کنند یکسان خواهد بود.
۳. برای زمین ۸تایی و عمق‌های یکسان 3 الگوریتم minimax خالی 28.55 ثانیه طول می‌کشد و در مقابل، الگوریتم alpha-beta pruning مقدار زمانی که برای تمام شدن نیاز دارد 14.20 ثانیه است.