گزارش پروژه دوم هوش مصنوعی ـ بخش بازی

الگوریتم minimax

در این پروژه از الگوریتم minimax به همراه alpha-beta pruning برای پیاده‌سازی بازی بین دو حریف استفاده شده است. با توجه به زیاد بودن عمق و braching factor در درخت این بازی، از depth-limited search استفاده کردیم تا بازی در زمان معقولی اجرا شود. برای استفاده از این الگوریتم نیاز است که بتوانیم تخمینی از امتیاز بازیکن در یک گره را به دست آوریم تا زمانی که به عمق مورد نظر رسیدیم این مقدار بازگردانده شود. برای این کار سه پارامتر را ارائه می‌دهیم که evaluation function از حاصل جمع وزن دار آن‌ها برای هر دو بازیکن به دست می‌آید:

- تعداد مهره‌های باقی مانده: با توجه به این که بازیکن زمانی برنده می‌شود که مهره های حریف تمام بشود، این پارامتر پارامتر مهمی محسوب می‌شود و بنابراین بیشترین وزن را به آن اختصاص دادیم. ضریب این پارامتر در evaluation function برابر 3 است. همچنین اگر از یکی از رنگ‌ها مهره‌ای باقی نمانده باشد، بسته به رنگ آن مهره، بیشترین یا کمترین امتیاز ممکن بازگردانده می‌شود تا ارزش آن حالت را مشخص کند.
- تعداد مهره های king : داشتن مهره king در صفحه امتیاز مثبتی برای بازیکن ایجاد می‌کند. اما اگر ضریب این پارامتر را بالا در نظر بگیریم، هزینه از دست دادن این مهره برای بازیکن زیاد می‌شود و باعث می‌شود که در موقعیت‌هایی که می‌تواند با به خطر انداختن این مهره شانس برد خود را افزایش دهد این کار را انجام ندهد. ضریب این پارامتر در evaluation function برابر 0.5 است.
- تعداد مهره های محافظت شده: مهره های محافظت شده مهره‌هایی هستند که امکان حذف شدن آن‌ها در آن استیت از بازی وجود ندارد. به عبارت دیگر مهره حریف نمی‌تواند این مهره را تهدید کند. این پارامتر به این دلیل انتخاب شده که به استراتژیک بازی کردن بازیکن‌ها کمک کند. ضریب این پارامتر در evaluation function برابر 1 است.

محدود کردن حرکت تکراری

در این بازی پس از اضافه شدن king به بازی، به حالت هایی می‌رسیم که بازیکن ها چون حرکت بهتری ندارند، دائما یک حرکت را تکرار می‌کنند. برای این که بتوانیم ارزیابی بهتری از عملکرد الگوریتم داشته باشیم، جلوی حرکت های تکراری را می‌گیریم. به این صورت که وضعیت صفحه بازی را در تعداد محدودی از گام های قبلی ذخیره می‌کنیم و اگر حرکتی باعث می‌شد که به وضعیت قبلی صفحه برگردیم جلوی آن را می‌گیریم.

بررسی عملکرد الگوریتم در عمق‌های مختلف

در این قسمت به بررسی عملکرد الگوریتم در عمق‌های مختلف می‌پردازیم و نتیجه را در عمق های کم و زیاد تحلیل می‌کنیم.

در ابتدا برای این که بتوانیم مقایسه بهتری از تعداد حرکت‌های لازم برای برنده شدن داشته باشیم، تعداد حرکت های لازم برای برنده شدن در حالتی که عمق بررسی بازیکن ها در بازه ۱ تا ۳ باشد را محاسبه می‌کنیم:

تست الگوریتم با عمق کم

در این بخش جست‌وجو را در عمق ۱ برای هر دو بازیکن انجام دادیم.

- در این حالت تعداد مهره‌ها به سرعت کاهش می‌یابد چون در عمل بازیکن‌ها عواقب یک حرکت را بررسی نمی‌کنند و صرفا حرکتی که در آن لحظه امتیاز بیشتری دارد را انتخاب می‌کنند که باعث می‌شود هم مهره ها بیشتر در موقعیت حذف شدن قرار بگیرند و هم بازیکن همیشه حذف کردن مهره حریف را در صورت امکان انتخاب کند.
- در این حالت بازی پس از ۶۰ حرکت به نتیجه می‌رسد که نسبتا سریع است.
- سرعت اجرای هر گام به دلیل کم عمق بودن بررسی بالا است.

تست الگوریتم با عمق متفاوت برای دو بازیکن

در این بخش جست‌وجو را در عمق ۲ و ۵ برای دو بازیکن انجام دادیم.

- در این حالت بازیکنی که عمق بیشتری دارد برنده می‌شود.
- در این حالت بازیکن با عمق بیشتر به زمان بیشتری برای تصمیم‌گیری نیاز دارد.
- در این حالت بازی پس از ۵۸ حرکت به نتیجه می‌رسد که نسبتا سریع است. علت این موضوع این است که بازیکن با عمق بیشتر حرکت‌های بهتری انجام می‌دهد و سریعا برنده می‌شود.

تست الگوریتم با عمق بالاتر

در این بخش جست‌وجو را در عمق ۵ برای دو بازیکن انجام دادیم.

- در این حالت زمان بیشتری طول می‌کشد تا هر بازیکن حرکت خود را انجام دهد.
- در این حالت وقتی تعداد مهره‌های زمین کم می‌شود، پیشروی بازی کند می شود و در بعضی حالت‌ها به نتیجه نمی‌رسد.چون هر بازیکن سعی می‌کند جلوی حذف شدن مهره‌هایش را بگیرد و به مهره های حریف نزدیک نمی‌شود، چرا که زمانی که تعداد کمی مهره باقی مانده باشد بازیکنی که اول به حریف نزدیک شود خود را در خطر حذف شدن قرار می‌دهد.