به نام خدا

گزارش پروژه hands-on2 سینا سلیمیان – 810197528

موضوع: بازی و الگوریتم minimax

توضیح پروژه: در این پروژه قصد داریم در بازی مقابل یک روبات که رغیب ما حساب می شود، طوری عمل کنیم که برنده بازی باشیم. به این منظور به پیاده سازی الگوریتم minimax می پردازیم.

برای پیاده سازی الگوریتم minimax به یک تابع بازگشتی نیاز داریم که موفقیت آمیز بودن یک راه از بین انتخاب های موجود را پیش بینی کند. تابع بازگشتی ()next_move را به کلاس blacksin اضافه می کنیم و آن را در تابع ()get_player_input صدا می زنیم.

این تابع دارای ورودی های turn و depth می باشد. که turn = 0 به معنای نوبت بازی برای player و turn = 1 به معنای نوبت بازی برای opponent می باشد. در این تابع باز گشتی در هر سری یک کپی از وضعیت کنونی بازی(deck ,seen cards, player, opponent, ...) می گیریم و آن را در new_game ذخیره می کنیم. به این صورت تغییرات موقت برای پیش بینی بازی را بر روی کپی ای از کلاس blacksin انجام می دهیم و داده های اصلی ما دچار تغییر نمی گردند.

متغیر depth برای این است که تا ارتفاعی مشخص از درخت را پیش بینی کنیم و با افزایش آن احتمال برد بالا می رود ولی زمان بیشتر می شود.

در تابع ()next_move برای هر یک از انتخاب ها (draw, stop, erase self, erase opponent) 3 حالت وجود دارد: برد ما، برد رغیب، مساوی. که به کمک تابع از پیش تعریف شده ی ()check_for_winners این کار را انجام می دهیم.

تابع ()next_move از دو بخش اصلی تشکیل شده است، نوبت ما و نوبت رغیب. در بخش نوبت ما که turn = 0 است، هدف پیدا کردن بزرگ ترین عدد میان خروجی های حالات ممکن است ولی در قسمت رغیب، هدف باختن ما و پیدا کردن کوچکترین عدد بین 1 و0 و1- است.

در نهایت خروجی تابع next_move()، حرکت انتخاب شده و وضعیت نتبجه ادامه بازی در صورت انتخاب حرکت است. که در تابع get_player_input() فقط با حرکت انتخاب شده کار خواهیم داشت که آن را به تابع handle_input() پاس می دهیم.

برای ساخت درخت، از محل فعلی شروع می کنیم و به صورت یکی در میان در صورتی که بازیکنی stop نکرده باشد نوبت ها را تغییر می دهیم و هنگامی که به ارتفاع مشخص یا برگ ها رسیدیم مقدار 1,-1,0را بر اساس وضعیت امتیاز ها return می کنیم و همینطور به بالا می آییم و در هر مرحله بین حرکات و امتیاز هایشام تصمیم می گیریم و مقداری را return می کنیم تا به ریشه درخت فعلی برسیم و حال می توانیم با توجه به پیش بینی هایی که انجام داده ایم انتخاب کنیک که با انجان کدام حرکت بهترین نتیجه را خواهیم گرفت.

برای هرس کردن نیز بررسی می کنیم که اگر در حرکتی به بشترین جواب (یا کمترین turn=1) می رسیم، دیگر سایر حالات را بررسی نمی کنیم که این کار به شدت به زمان ما کمک می کند.

سوالات:

سوال 1) اگر از ترتیب کارت ها در دسته کارت ها اطلاع نداشتید (نمیدانستید کارت بعدی که می کشید، چه کارتی است) می توانستید از minimax استفاده کنید؟ آیا الگوریتمی میشناسید که در این حالت بتوان از آن استفاده کرد؟

خیر از minimax نمی توان استفاده کرد چون ترتیب کارت های بعدی را نمی دانیم.

به جای آن با توجه به کارت هایی که در دست داریم و تعداد کل کارت ها که 21 است، می توانیم تخمینی از کارت هایی که هنوز برنداشته ایم بزنیم و میانگین اعداد روی آن ها را در هر سری از نوبت مان محاسبه کنیم. اگر جمع کارت هایی که در دست داریم با میانگین محاسبه شده کمتر از 41 بود، آن کارت را برمی داریم و در غیر اینصورت به سراغ بررسی حالات دیگر می رویم. به طور مثال محاسبه می کنیم که در صورت حذف کارت از دست خودمان، جمع میانگین و کارت هایمان چقدر می شود و در صورت مناسب بودن شرایط آن را حذف می کنیم. بدین ترتیب با بیشترین احتمال برد، بازی را انجام می دهیم.

سوال 2) ابتدا بدون هرس کردن نتایج را در چند عمق بررسی کنید(هم از نظر زمانی و هم عملکرد) و سپس از روش های هرس کردن استفاده کنید.

Depth / algorithm	3	5	7
Minimax (no lopping)	14 seconds - 81%	105 seconds-92%	620 seconds – 97%
Minimax (with lopping)	6 seconds - 82%	20 seconds - 92%	69 seconds - 96%

سوال 3) وقتی از روش های هرس کردن استفاده می کنی، برای هر گره درخت، فرزندانش را به چه ترتیبی اضافه کردید؟ آیا این ترتیب اهمیتی دارد؟ چرا آن را انتخاب کردید؟

es-4 eo-3 s-2 draw – 1

احتمال draw کردن بیشتر از سایر حرکت ها است و محدودیت روی آن به احتمال کمتری رخ می دهد زیرا کم پیش می آید که کارت ها تمام شوند و ما بخواهیم کارتی برداریم. پس در این حالت کمی سریع تر به پاسخ می رسیم. و به دلیل مشابه es را در آخرین حرکت قرار می دهیم.