به نام خدا

گزارش پروژه پایان ترم درس هوش مصنوعی و سیستم های خبره (بازی کوریدور)

گروه شماره ۱۲ – غزل کلهری ، عرفان مشیری

دانشگاه شهید بهشتی - نیم سال تحصیلی ۹۹-۹۸

پروژه مورد نظر، شبیه ساز بازی فکری-استراتژی*کوریدور* (Quoridor) می‌باشد که به زبان برنامه نویسی جاوا و با محیط گرافیکی Javafx پیاده سازی شده است.

برنامه نوشته شده، از سه بخش مختلف بازی دونفره: ۱. انسان در مقابل انسان، ۲. انسان در مقابل هوش مصنوعی و ۳. هوش مصنوعی در مقابل هوش مصنوعی تشکیل شده است. در ادامه به اختصار به توضیح هریک از بخش های گفته شده می‌پردازیم.

بطور کل دراین برنامه، بخش گرافیکی بوسیله تابع main کنترل میشود و از طریق توابع کمکی مخصوص هر بخش به کنترل جریان بازی میپردازد.

اولین نوع بازی (انسان در مقابل انسان) همانند بازی حضوری، تابع قوانین تعریف شده است که در صورت پروژه ذکر شده. روند بازی بدین صورت است که کلاس playWithFriend فراخوانده شده؛ اطلاعات بازی را مانند حالت، تعداد دیوارها، مکان هر بازیکن و... را نگه میدارد.

شروع کننده بازی بصورت تصادفی با استفاده از تابع rand انتخاب شده و میتواند یکی از دو اقدام حرکت با استفاده از تابع movePlayer یا گذاشتن دیوار با استفاده از تابع puttingWall را انجام دهد. در صورت اقدام به حرکت، بازیکن برروی یکی از سلول های رابط کاربری گرافیکی کلیک کرده و در صورت معتبر بودن حرکت که توسط تابع movePlayer تعیین میشود، مهره بازیکن جابجا شده و اطلاعات جدید صفحه بازی و بازیکن آپدیت میشود.

در صورت اقدام به گذاشتن دیوار نیز، مشابه همین روند و در تابع puttingWall تکرار میشود. در هریک از حالات گفته شده اگر اقدام انجام شده معتبر نبود، به بازیکن پیامی نمایش داده شده و تغییری در بازی صورت نمیگیرد تا اقدامی معتبر صورت بگیرد.

بعد از هر اقدامی بررسی میشود که به goalState رسیده ایم یا خیر(وضعیتی که یکی از بازیکن ها به لاین نهایی مقابل خود برسد و برنده شود). اگر اینطور بود، بازی تمام میشود.

دومین نوع بازی (انسان در مقابل هوش مصنوعی)، توابع استفاده شده برای انسان، همان توابع بخش قبل میباشد. ولی برای هوش مصنوعی روند متفاوت است. بدین صورت که هرگاه نوبت هوش مصنوعی است، حالت فعلی بازی در قالب یک نود که خود یک شی شامل اطلاعات بازی میباشد، به تابع miniMax پاس داده میشود. سپس کلیه فرزندان هر یک از نودها را در تابع childGenerator تولید میکنیم. فرزندان هر نود، اجتماع تمام حالات ممکن مجاز برای حرکت مهره در وضعیت فعلی خود است که توسط تابع movePlayer انجام میشود، و همچنین تمام حالات ممکن مجاز برای گذاشتن دیوار در صفحه بازی میباشد که توسط تابع puttingWall صورت میگیرد.این توابع در کلاس PathFinder موجود هستند. یک تابع هیوریستیک برای هر نود و فرزندان آن تعیین میکنیم که در ادامه با جزییات بیشتری بررسی خواهد شد.

برای تسریع در سرعت و کاهش حافظه استفاده شده برروی عملیات تولید فرزندان، ساده سازی هایی صورت میگیرد و یکسری از حالات اضافی حذف میشوند. به دلیل محدودیت زمانی، عملیات تولید فرزندان هر نود، تنها تا عمق ۳

ادامه پیدا میکند. در آخر، از طریق هرس آلفا-بتا بهترین فرزند نود ریشه (نودی که حالت فعلی بازی را نگه میدارد) انتخاب شده و جایگزین حالت فعلی بازی میشود (وضعیت بازی در هر مرحله آپدیت میشود).

تابع هیوریستیک تعیین شده برای بازی شامل ۴ پارامتراست:

۱. تفاضل تعداد دیوارهای دو بازیکن از یکدیگر؛ از آن نظر که بازیکن با تعداد دیوار بیشتر نسبت به رقیبش، شانس بیشتری برای طولانی کردن مسیر حریف و در نتیجه نزدیک تر شدن به برد خود دارد.

۲. تفاضل کوتاه ترین مسیرِ دو بازیکن تا پیروزی از یکدیگر؛ هر بازیکنی که مسیر کوتاهتری تا پیروزی داشته باشد، احتمال برد بیشتری دارد.

از آنجاییکه که کوتاهترین مسیر هر بازیکن، خود یک تابع ارزیابی قوی است که بسیاری از جوانب را میسنجد، ما را از استفاده از توابع بیشتر، بی‌نیاز کرده است.

۳. تفاضل حداکثر حرکات مستقیم دو بازیکن به جلو (تا رسیدن به اولین دیوار)؛ از انجاییکه مسیر مستقیم، سریعترین و بهترین راه رسیدن به مقصد است، هر بازیکنی که مسیر مستقیم بازِ طولانی تری داشته باشد، شانس بردن بیشتری دارد.

۴. تفاضل پارامترعمودی موقعیت دو بازیکن از یکدیگر؛ بازیکنی که بیشتر جلو آماده باشد، به خط پایان نزدیکتر است.

وزن هریک از پارامترهای ذکر شده، برای اولین بار بصورت تجربی تعیین شده و بعد از آن، از طریق یادگیری که جلوتر به آن میپردازیم، تغییر کرده و نسبت به دفعات قبل بهتر شده اند.

سومین نوع بازی (هوش مصنوعی در مقابل هوش مصنوعی)، با هدف بهتر شدن بازی هوش مصنوعی در مقابل انسان، از طریق یادگیری میباشد. فرآیند یادگیری بدین صورت میباشد که ابتدا دو ژن تصادفی که خودمون مقداردهی کردیم را انتخاب میکنیم. سپس با استفاده از بازترکیبی و الگوریتم تولید میانگین وزن دار از والدین، ژن ها را در کلاس initiateGeneration تکثیر کرده تا یک نسل ۹ تایی از ژن ها را تولید نماییم. این جمعیت برای نسل های آتی نیز همواره ثابت است. برای ایجاد هر نسل از روش بازنمایی خطی (آرایه ای) استفاده میکنیم، که برای بهینه سازی توابع روش مناسبی است.

در هرمرحله، نسل فعلی را در فایلی ذخیره میکنیم تا بعدا از آن استفاده کنیم، همچنین مجموعه تمام نسل های تولید شده تا کنون (من جمله نسل های از بین رفته) را برای مشاهه روند پیشرفت نسل ها در فایلی جدا ذخیره سازی میکنیم.

برای یادگیری ابتدا ژن های نسل فعلی که در اختیار داریم، سه به سه (دلخواه خودمان) به رقابت با یکدیگر میپردازند یعنی هر ژن آرایه ای ۴ تایی شامل وزن هایی است که همان ضرایب تابع هیوریستیک ما میباشد و ژن ها دو به دو باهم بازی کرده، به ژن برنده امتیاز بُرد (۱ امتیاز برای هر بازی) تعلق میگیرد. در پایان بازی های انجام شده بین ژن ها، آرایه ما که همان نسل فعلی است، بر اساس امتیاز برد هر یک از ژن های موجود در آن، بصورت نزولی مرتب میشود (تابع play).

روند انتخابی ما برای انتخاب بازماندگان نسل فعلی که نسل بعدی را تشکیل میدهند، بدین ترتیب است که ابتدا سه ژن برتر نسل فعلی انتخاب میشوند، سپس برای سه فرزند بعدی از طریق الگوریتم تولید میانگین وزن دار بین والدین تصادفی، سه ژن جدید تولید میشوند. سه فرزند آخر نیز با استفاده از بازترکیبی (بازنمایی خطی تک نقطه ای) بین والدین منتخب تصادفی (۳ مرحله، هر مرحله دو والد) تولید میشوند (تابع generateNextGeneration).

بدین ترتیب نسلی جدید تولید میشود، آن را جایگزین نسل قبلی میکنیم و آن را به مجموعه کل نسل های تولیده شده تا کنون اضافه میکنیم. این روند تولید نسل میتواند بارها ادامه داشته باشد و ما به دلخواه خود با اجرای مکرر بخش یادگیری، نسل هایی بهتر و قوی تر از نسل های قبلی تولید میکنیم(تابع writeGeneration).

در نهایت ژن برتر هر نسل که بنابر سیستم مرتب سازی نزولی ما، اولین ژن آن نسل است، به عنوان ژن برتر انتخاب شده و برای بازی با انسان در بخش دوم استفاده میشود. بنابراین با پیشروی نسل ها و بهبود آنها، هربار هوش مصنوعی قوی تری برای بازی با انسان جایگزین میشود.