گزارش پروژه Tron

امیررضا محسنی – نوید دادخواه – نگار هنرورصدیقیان

نام گروه : GAME OF TRONS

در این گزارش به طور خلاصه به توضیح پیاده‌سازی‌های انجام شده در راستای هوشمندسازی عامل در بازی tron میپردازیم، در ادامه به توضیحی اجمالی درباره کد خواهیم پرداخت و در پایان نتیجه بازی این عامل با حریفان را قرار داده ایم.

الگوریتم بازی

الگوریتمی که برای این بازی در نظر گرفتیم ترکیبی از الگوریتم ژنتیک و مین ماکس به این صورت است که الگوریتم مین ماکس به عنوان تابع سنجش شایستگی (fitness) برای الگوریتم ژنتیک ما در نظر گرفته می‌شود. با توجه به این هر سیکل ۸ ثانیه طول میکشد ما درختی به عمق ۶ در هر سیکل ساخته و مورد بررسی قرار می‌دهیم زیرا در عمق‌های بیشتر به زمانی بیشتر نیازمندیم. در هردرخت عمق های زوج متعلق به حرکت‌های عامل ما و عمق‌های فرد مربوط به حرکت‌های عامل حریف می‌باشد. با وجود real time بودن بازی ما برای بکارگیری minimax لازم بود آن را turned based در نظر بگیریم و لذا در نظر گرفتیم که در آغاز ما یک حرکت انجام میدهیم (عمق صفر درخت مین ماکس) و حریف پس از ما اقدام به تصمیم‌گیری برای حرکت بعدی خود می‌کند و به این ترتیب در هرمرحله نسبت به ما برتری دارد. به کمک این درخت ما میتوانیم سه حرکت بعدی عامل خودمان و عامل حریف را پیش‌بینی کنیم (توجه داشته باشید که ما در طول بازی نسبت به عامل حریف نابینا هستیم و نمیدانیم بخش پیشرو آن در کدام خانه قرار دارد و تنها میتوانیم درمورد خانه‌هایی که توسط آن ساخته شده آگاهی کسب کنیم.)

در رابطه با پیاده‌سازی الگوریتم ژنتیک یک راهبرد این است که فرزندان شایسته را در هر دوره به مین‌ماکس داده و با توجه به حرکتی که توسط این تابع پیش بینی می‌شود حرکت کنیم.

برای پیاده‌سازی الگوریتم‌ها نیاز است تا به ویژگی‌های world و agent و توابعی که آن‌ها را کنترل می‌کنند دسترسی داشته باشیم و تغییرات آن‌ها را بررسی کنیم بدون اینکه این دستورات تغییر به سرور ارسال شوند؛ پس برای شبیه‌سازی هرچه بهتر محیط بازی برای تابع minimax یک پکیج به نام classes در پوشه pythonClient تعریف می‌کنیم که شامل دو فایل world

و agent است که در هریک این دو کلاس که پیشتر در models و نیز پوشه pythonServer تعریف شده بودند با توجه به کارکردی که برای آن‌ها در نظر داریم مجددا ساخته و تعریف شدند.

>>PythonClient/Classes/Worlds.py

در این بخش با بهره‌گیری از بخش‌های موردنیاز بازی که در ks تعریف شده‌اند اقدام به بازسازی جهان بازی می‌کنیم.

در آغاز تابع get_d_cords را تعریف میکنیم که مختصات عامل را با توجه به اینکه در کدام جهت حرکت کند تغییر می‌دهد.

```
4    def get_d_cords(dir_to_take):
5        if dir_to_take == EDirection.Left:
6            return -1, 0
7        elif dir_to_take == EDirection.Right:
8            return 1, 0
9        elif dir_to_take == EDirection.Up:
10           return 0, -1
11       elif dir_to_take == EDirection.Down:
12           return 0, 1
```

در تابع change_board اقدام به پیاده‌سازی حرکات بازی و محاسبه امتیاز میکنیم.

در صورتی که wall_breaker عامل ما روشن باشد ابتدا به کمک توابعی که در ادامه تعریف شده‌اند (نحوه کارکرد آن‌ها مشابه با نام آن هاست و لذا برای جلوگیری از طولانی شدن گزارش به بررسی آن‌ها نمی‌پردازیم) اولویت با این است که وارد خانه‌هایی که حریف مالک آن‌هاست شویم، سپس اولویت بعدی متعلق به خانه‌های خالی و اولویت سوم متعلق به خانه‌های عامل خودمان است، در صورتی که هیچ گزینه‌ای ممکن نباشد ایجنت ما با area wall برخورد کرده و بازی

خاتمه می‌یابد. اگر wall breaker خاموش باشد اولویت با این است که بررسی کنیم و در صورت امکان وارد خانه‌های خالی شویم، در غیر این صورت بررسی می‌کنیم و در صورت امکان wall breaker را روشن کرده و وارد خانه متعلق به حریف می‌شویم، در غیر این صورت وارد خانه متعلق به خودمان شده و در بدترین حالت با area wall برخورد می‌کنیم. در صورتی که امکان روشن کردن wall breaker نباشد و هیچ خانه خالی ای موجود نباشد ترتیب visit خانه‌ها همچنان به ترتیب فوق است. امیتاز برای شبیه سازی نیز به این صورت محاسبه می‌شود که در صورت ساخت هر دیوار جدید یک امتیاز به ما افزوده شده و در صورتی که دیوار حریف را تخریب کنیم یک امتیاز نیز از حریف کاسته می‌شود.در صورتی که عامل بدون روشن کردن wall breaker اقدام به ورود به خانه‌هایی دارای دیوار کند ۱۵۰ امتیاز به علت از دست دادن یک health از دست می‌دهد؛ علت در نظر گرفتن این مقدار برای از دست دادن health این است که در طول بازی ضریب امتیازی زیادی داشته و ما تنها ۳ تا health در طول بازی داریم که با صفر شدن آن عامل ما باخته و بازی خاتمه

می‌یابد پس با در نظر گرفتن این ضریب سعی داریم به عامل اطلاع دهیم تا حد ممکن اقدام به همچین عملی نکند.

با تغییر عمق درخت لازم است تا اطلاعات عامل‌ها در تابع swap شود تا به پیش‌بینی حرکت بعدی عامل حریف بپردازیم.

>>PythonClient/ai.py

در این فایل اقدام به پیاده‌سازی الگوریتم مینی‌ماکس بازی کرده ایم.

در این رابطه اقدام به تعریف توابع فوق کرده ایم:

get_next_nodes

این تابع به عنوان یک تابع کمکی برای minimax تعریف شده است. در این تابع یک لیست به عنوان بهترین جواب با مقدار اولیه min تابع minimax تعریف شده است . در ادامه این تابع در

هر عمق درخت مینی‌ماکس بسته به اینکه عمق برای حریف بوده یا برای عامل ما از جهان بازی یک deep copy گرفته و سپس درخت مینی‌ماکس‌را مجددا می‌سازد و تابع مینی‌ماکس را فراخوانی می‌کند و مقدار جدیدی برای best answer در نظر گرفته و آن را برمی‌گرداند.

Update_world_and_agent

همان‌طور که پیش‌تر گفتیم لازم است برای پیش‌بینی مینی‌ماکس از بازی محیط آن را شبیه‌سازی کنیم. این تابع اقدام به شبیه‌سازی ویژگی‌های عامل ما ، عامل حریف و جهان بازی با مقداردهی‌های موردنظر برای ویژگی‌های این موارد است.

Minimax

در این تابع به دنبال پیاده‌سازی تابع مینی‌ماکس به گونه‌ای هستیم که میزان اختلاف امتیاز دو عامل بیشترین مقدار ممکن شود. برای اینکار درختی به عمق۶ میسازیم.

در صورتی که به عمق ۶ رسیده بودیم لیستی از حرکات که بیشترین اختلاف امتیاز را به ما می‌دهند را برمیگردانیم.

سپس بررسی می‌کنیم که آیا شروط خاتمه بازی برقرار هستند یا خیر، این شروط عبارت اند از :

برخورد یک عامل با area_wall

پایان یافتن سیکل ها

برخورد دو عامل به یکدیگر

پایان یافتن health یکی از عامل‌ها

سپس مشابه با آنچه در رابطه با اولویت حرکت عامل در هرحالت در کلاس world توضیح دادیم اقدام به هدایت آن می‌کنیم، توجه داریم که در صورتی که تعداد سیکل‌های باقیمانده از wall breaker یک‌باشد برای حفظ health عامل اولویت ما در سیکل بعدی ورود به خانه‌های خالی است.

نتایج بازی با عامل‌های ربات تلگرام