علیرضا رشیدی و علی مقیمی

96243085 - 96243097

پروژه ی نهایی بازی ریسک – هوش مصنوعی

دکتر سلیمی بدر

در این بازی ما 2 نوع بازیکن داریم که شامل هوش مصنوعی، و انسان می شود

در ابتدا یک کلاس player داریم که یک کلاس عام است و 2 کلاس دیگر(انسان و هوش مصنوعی) از آن ارث بری میکنند و شامل تعدادی فیلد می باشد مهم ترین فیلد در این کلاس این است که هر بازیکن یک لیستی از ریجن هایی که در نقشه مالک آن های می باشد دارد و فیلد های دیگر زیاد مهم نیستند.

حال بخواهیم در رابطه با متد های مهم صحبت کنیم متد Battle می باشد که 2 عدد آبجکت از جنس کلاس ریجن می گیرد(جلو تر در رابطه با این کلاس گفته می شود.) که شامل ریجن شخص حمله کننده و شخص مدافع می باشد. و در واقع نبرد طبق قوانین بازی ریسک در آن انجام می گیرد (بعد از یافتن بهترین خانه ای که به آن نیاز است حمله کنیم این را صدا میزنیم در bestAttackMoveSelect)

کلاس انسان صرفا برای مشخص کردن تفاوت بین انسان و عامل هوش مصنوعی می باشد.

حال اگر بخواهیم در رابطه نکات مهم کلاس AIPlayer (کلاس هوش مصنوعی) صحبت کنیم این می باشد که یک متد خیلی مهم برای پیاده کردن بهترین ریجنی که هوش مصنوعی داره برای حمله رو حال یه جورایی ایده از الگوریتم minimax استفاده شده که در ابتدا باید گفته شود که این تابع ریفرنسی از جنس Move برمیگرداند که خودش یک کلاس می باشد که شامل 3 فیلد مهم است

1. moveScore : این فیلد با توجه به احتمالات تعداد نیروهای حمله کننده به تعداد نیرو های دفاع کننده در ریجن خودش مقدار دهی می شود که توسط تابع getProbibilty 1 یا 2 محاسبه میشود (2 درست تر و در پروژه اجرای آن مد نظر است است و در قسمت مربوطه به هیوریستیک و ته پروژه جفت توضیح داده میشوند)

2. bestRegionToAttack: این فیلد بهترین موقعیت بین همسایه های ریجن حمله کننده را در خود ذخیره می کند و جنس آن ریجن است

3. bestRegionToAttack : این فیلد در واقع بهترین ریجن را برای حمله کننده نگه می دارد.

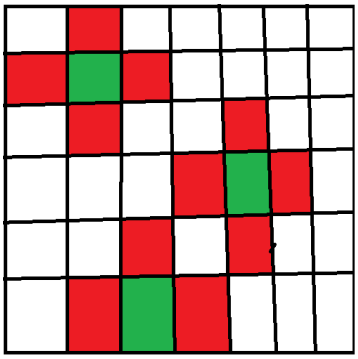
حال الگوریتم متد bestAttackMoveSelect به این شکل می باشد که ابتدا به ازای هر ریجن شخص حمله کننده که در رابطه با حلقه for اول می باشد همه ریجن های مجاور(دشمن) شخص حمله کننده را بررسی میکند که داخل for دوم صورت میگیرد و تابع attackMiniMax فراخوانی می شود که این تابع در واقع احتمال فیلد شماره 1 توضیح داده شده را نسبت به تعداد نیرو های ریجن حمله کننده به نیرو های شخص دفاع کننده محاسبه می کند. و 2 فیلد دیگر را نیز مقدار دهی

می کند حال بعد از اتمام کار این تابع نتیجه در یک ریفرنس Move به نام nextMove ذخیره می شود و حال باید چک شود که آیا بهترین حرکت تا به این جای کار بهتر است یا حرکت جدید (nextMove) که بر این اساس چک می شود که باید احتمال برد ما که همان هیوریستیک حرکت بعدی بود از بهترین حرکت بالا تر باشد تا ما بهترین حرکت را به روز کنیم . باید از 50 درصد نیز بالا تر باشد تا با اطمینان کامل حمله کنیم

2 تا شمارنده در این تابع هستند که که صرفا برای این است که اگر عامل دیگر حرکت خوبی نیافت یک ریفرنس از با فیلد های null برگرداند که با خالی بودن این سنجیده می شود که دیگر در آن نوبت به کسی حمله نکند.

شکلی از روش اجرا شدن کد و توابع

attackMiniMax و bestAttackMoveSelect



خانه های مجاور خانه ای که میخواهیم از آن حمله را شروع کنیم و در واقع از سبز به این خانه ها حمله میکنیم

خانه هایی که از آنها حمله میکنیم

در لووپ اول خانه های سبز پیمایش میشوند و در هر پیمایش خانه های مجاور خانه های سبز چک میشوند و براساس تعداد سرباز های موجود در خانه ی حمله کننده نسبت به تعداد سرباز های خانه های دفاع کننده با فرمولی که در پی دی اف هست محاسبه میشوند و امتیازی میدهند ، اگر امتیاز بهتری یافت شد نودی که بهترین راه و برد را دارد جایگزین نود های قبلی با امتیاز کمتر میشود و در پایان به آن نود است که حمله میکنیم

نحوه کار کردن bestMoveReinforceSelect :(در بخش Reinforce کردن)

ما برای اینکه عامل هوشمند بتواند تشخیص دهد که کدام یک از ریجن های خود را برای افزایش نیرو انتخاب کند از هیوریستیک های NBSR, BSR, BST استفاده کرده ایم که نحوه استفاده آن به طور کامل در مقاله ی Evaluating Heuristics in the Game Risk با از Franz Hahn نوشته شده توجه به مقدار NBSR ما بهترین ریجن برای نیرو اضافه کردن را (برای عامل هوشمند) با الگو گیری از الگوریتم MiniMax استفاده می کنیم.

اگر بخواهیم خلاصه ایی در رابطه با این هیوریستیک ها بگوییم خواهیم داشت:

BST.1 : در این روش برای ریجن i ام می آییم و تعداد نیرو های ***مجاور های دشمن*** را با هم جمع می زنیم تا میزان تهدید ها را در بین کشور ها یا سرزمین های مرزی را محاسبه کنیم فرمول آن به طور خلاصه خواهد بود:

$$BST_x = \sum_{y}^{n} AmountOfUnits_y$$

BSR.2 : حال می توان با تقسیم کردن BSTریجن iام بر تعداد کل نیرو های خودی آن ریجن یک نسبتی به نام Border Security Ratio به دست آورد خواهیم داشت:

$$BSR_x = \frac{BST_x}{AmountOfUnits_x}$$

NBSR.3 : این هیوریستیک که برای پیدا کردن بهترین ریجن برای افزایش نیرو استفاده می شود از حاصل تقسیم BSR ریجن i ام بر جمع کل BSR های ریجن های بازیکن x محاسبه می شود که در واقع یک عدد نرمال شده را به ما می دهد تا بتوان به درستی ریجن های بازیکن حال حاضر را برای افزایش نیرو مورد انتخاب قرار داد. فرمول آن در زیر آمده است:

$$NBSR_x = \frac{BSR_x}{\sum_{z=1}^{n} BSR_z}$$

ما در کد زده شده از تابع CalculateNBSR برای محاسبه این 3 هیوریستیک بهره می بریم و در تابع bestMoveSelect برای هر ریجن بازیکن x را محاسبه می کنیم و در نهایت بهترین ریجن برای افزایش نیرو برای عامل هوشمند مشخص شده و از تابع بازگردانده می شود.

کلاس Region:

در این کلاس چند فیلد مهم هستند که عبارتند از:

1. مختصات های x , y در داخل بورد بازی
2. یک ریفرنس از جنس بازیکن (player) به صاحب ریجن مورد نظر
3. لیستی از ریجن های مجاور ریجن فعلی
4. و تعداد نیرو ها (unit) داخل این خانه
5. ***و تعداد یونیت هایی که برای حمله نیاز هستند*** که برای پیاده سازی این مورد تعداد نیرویی که برای حمله به ریجن مورد نظر نیاز هست مشخص می شود و از تعداد یونیت های داخل آن خانه کم میشود و با آن تعداد حمله

صورت می گیرد و دوباره تعداد باقی مانده بعد از نبرد با نیرو های داخل ریجن حمله کننده جمع می شود که این موارد در خطوط 403 و 43 که شروع تابع و عملیات تابع نبرد می باشد.

یکسری متد در این کلاس موجود هستند که برای ست کردن مجاور های و تشخیص مجاور های دشمن می باشند.

کلاس Risk Game:

این کلاس شامل یک آرایه 2 بعدی از جنس ریجن ها می باشد که با آن بازی انجام می پذیرد.

و یک تابع برای ست کردن player ها روی برد می باشند.

و تابع Game Control که عملیات بازی در انجام می پذیرد. شامل تعیین نوبت حین بازی دریافت تعداد بازیکن های انسان و AI فراخوانی توابع نبرد و reinforce و نمایش بورد بازی و... می باشد.

روش در نظر گرفتن هیوریستیک برای تابع getProbability که در Minimax استفاده شد.

چون برعکس بازی اصلی ریسک ما 1 تاس فقط داریم (حمله 1 تاس و دفاع 1 تاس) براس همین احتمال های برد حمله به این صورت میشود : (Defender Dice , Attacker Dice) > 2و1 | 3و1 ، 3و2 | 4و1 و... 6و5

که حکل حالات برد به این شکل میشوند 5 + 4 + 3 + 2 + 1 = 15

که تقسیم بر کل تعداد 2 تاس میشود 15 / 36 = 0.41

که این احتمال ثابتی است چون 1 تاس داریم ، پس انداختن تاس تاثیری در جواب ما ندارد

یعنی اگر 1 بیاریم احتمال برد 0/36

2 بیاریم 1/36

...

6 بیاریم 5/36

مثل روش بالا

پس اصل مهم در انجا میتواند صرفا چک کردن تفاوت بین تعداد حمله کنندگان به تعداد دفاع کنندگان باشد و نیازی به بدست اوردن آن همراه احتمال نیست

پس یعنی در بین این حالات :

5-2 = 3

5-3 = 2

5-4 = 1

بهترین جواب اولی است چون اختلاف بیشتری دارد

ما این روش را روش اصلی خود گرفتیم و در تابعی به نام getProbablility2 ازان استفاده میکنیم

روش دیگر که در این سوال زیاد قابل استفاده نیست حل و بدست اوردن احتمال با فرمول مارکوو و ... است که جدولی به این صورت میدهد. که این براساس قانون اصلی بازی است که با تعداد

V	U	E	BST	BSR	NBSR
1	1	0	0	0	0
2	7	1	5	0.71	0.18
3	4	2	6	1.50	0.37
4	5	2	9	1.80	0.45
5	5	3	16	3.20	0.38
6	1	1	4	4.00	0.47
7	4	1	5	1.25	0.15

افراد در هر ریجن تعداد تاس پرتابی و درنتیجه احتمال عوض میشود و

عدد ها نزولی مرتب میشوند و میتوان آنهایی را در نظر گرفت که احتمال

بیشتر از 0.5 دارند.

ولی خب ما این روش را بعنوان روش دوم و موقت در getProbablity1

درست کردیم به همراه جدول آن

بقیه ی حالات در کد با کامنت گذاری نوشته شده اند ،

نکات مهم دیگر :

- ما تعداد افرادی که در هر سری به جدول اضافه میشدند را به تعداد خانه های تصرف شده توسط فرد حمله کننده در نظر میگرفتیم.
- جدول ثابت 10 در 10 است به گفته ی سوال
- برای ست رندوم جدول دو روش به کار رفته است که یکی را استفاده میکنیم