گزارش پروژه درس منطق

استاد درس : آقای دکتر فراهانی

موضوع : پیاده سازی مسئله Wampus Hunt

گردآورندگان : مهسا گلمغانی – مجتبی کنعانی

- **مقدمه و مسئله :**

هدف از طرح این مسئله پیدا کردن روشی مبتنی بر منطق های کلاسیک است که با کمک گرفتن از این ابزار بتوانیم در مسئله Wampus Hunt به خانه ای که در آن طلا وجود دارد برسیم و به خانه شروع بازگردیم. اگر به خانه هایی برویم که در آن ها گودال یا Wampus وجود دارد میمیریم و باید تا حد امکان از این خانه ها دوری کنیم.

- **راه حل و ایده :**

به این دلیل که در مسیر پیدا کردن طلا ، گاهی ممکن است به گودال ها یا Wampus هایی برسیم که حضور آن ها را میتوانیم از علائمی که در خانه های مجاورشان ظاهر میشود متوجه شویم ، باید چاره ای برای رد کردن این موانع پیدا کنیم ، در مواجهه با گودال ها کار خاصی نمیتوانیم بکنیم جز این که آن ها را دور بزنیم و امیدوار باشیم که خانه ای که در آن طلا وجود دارد با تعدادی گودال محاصره نشده باشد و راهی برای دسترسی به آن وجود داشته باشد. اما در مواجهه با Wampus ها به کمک اطلاعاتی که از خانه های قبلی که از آن ها گذر کرده ایم ذخیره کرده ایم سعی میکنیم که تشخیص دهیم که Wampus در کدام خانه قرار دارد، اگر موفق به پیدا کردن Wampus شدیم با تیر زدن آن را از سد راه برمیداریم و به مسیرمان ادامه میدهیم تا به مقصد برسیم. برای بدست آوردن این اطلاعات از خانه ابتدایی شروع میکنیم و تابعی را به صورت Recursive صدا میزنیم که وارد هر خانه ی جدیدی که شد ابتدا بررسی کند که آیا این خانه Safe است یا نه؟ خانه ای را Safe در نظر میگیریم که مطمئن باشیم اگر از این خانه به هر کدام از خانه های مجاورش گذر کنیم نمیمیریم. اگر خانه ی جدیدی که وارد آن شدیم Safe بود دوباره تابع را صدا میزنیم تا دوباره این عمل را روی این خانه انجام دهد، اما اگر خانه Safe نبود به خانه ی قبلی باز میگردیم و سعی میکنیم با بقیه حرکت های ممکن در خانه ی قبلی کار را ادامه دهیم در نهایت با این کار ما تمام خانه هایی که از راه های Safe قابل دسترسی بودند را visit کرده ایم و اگر طلا در یکی از این خانه ها باشد ما آن را پیدا میکنیم اما اگر تا اینجای کار نتوانستیم طلا را پیدا کنیم اگر راهی برای رسیدن به طلا وجود داشته باشد باید با کشتن Wampus این راه را باز کنیم پس بررسی

میکنیم که آیا با اطلاعاتی که تا این جا از خانه هایی که بررسی کرده بودیم میتوانیم محل دقیق یک Wampus را تشخیص بدهیم یا خیر ، اگر توانستیم به آن خانه شلیک میکنیم و حال با کشته شدن Wampus مسیر جدیدی باز شده که میتوانیم این مسیر را نیز بررسی بکنیم که ببینیم آیا طلا در این مسیر جدید پیدا میشود یا خیر. در طراحی که ما انجام دادیم هوش مصنوعی ما به هیچ وجه خود را به کشتن نمیدهد و موفق نشدن ما برای پیدا کردن Gold به صورت زیاد شدن بیش از حد تعداد move های Agent می باشد که در این پیاده سازی ماکسیمم move برای پیدا کردن Gold ، ۱۰۰ در نظر گرفته شده است و اگر در صد حرکت موفق به پیدا کردن Gold نشویم برنامه قطع میشود.

- **توضیحات مربوط به کد**

کد ما از سه بخش اصلی تشکیل شده است.

- **World**

در این بخش از کد ما صرفا دنیای بازی را با ورودی های مختلفی میگیریم میسازیم.
Size : نشان دهنده اندازه ماتریس میباشد مثلا اگر ورودی ۴ باشد اندازه ماتریس ۴*۴ خواهد بود.
Gold : نشان دهنده موقعیت طلا در ماتریس ساخته شده میباشد
Wampus_number : تعداد wampus هایی که برای بازی در نظر دارید.
Wampus_coordinate : و در n خط بعدی موقعیت هر wampus
pit_number : تعداد pit هایی که برای بازی در نظر دارید.
pit_coordinate : و در m خط بعدی موقعیت هر pit

با این ورودی ها ماتریس ما ساخته میشود به طوری که هر سلول از این ماتریس شامل یک دیکشنری است که Key های این دیکشنری عبارتند از :
S (Stench) - W (Wampus) - B (Breeze) - P (Pit) - G (Gold) - A (Agent)
در ابتدا تمام Value های مربوط به این مقادیر برابر صفر است.
سپس با توجه به ورودی های دریافت شده Value های مربوطه برابر ۱ میشوند.
Agent در ابتدا در خانه ۰و۰ است
خروجی این تابع ماتریس نهایی خواهد بود.

- **Agent**

در این بخش از کد کلاسی تعریف شده است که مربوط به Agent است و تمام اعمال و ویژگی های Agent در این جا تعریف شده است.

ویژگی ها عبارتند از :

Alive : نشان میدهد که آیا Agent زنده است یا خیر . در ابتدا با True مقداردهی میشود.

World : ماتریسی که با توجه با مقادیر ورودی ساخته شده است و در بخش قبل به آن پرداختیم.

Location : موقعیت Agent در ماتریس World

Moves : تعداد حرکت هایی که تا این جا انجام داده است.

Ok : اگر برابر ۱ باشد یعنی خانه ای که در آن قرار داریم Safe است و اگر ۰ باشد Unsafe است.

Bullet : تعداد گلوله های Agent که برابر ۱ میگذاریم.

Visited : لیستی از تمام خانه هایی که تا این جا دیده است.

Route_full : لیستی از تمام خانه هایی که تا این جا وارد آن ها شده است به ترتیب

Route : نزدیک ترین مسیر از خانه ۰و۰ تا مکان حال حاظر (دور ندارد)

Available_moves : به چه جهت هایی میتواند حرکت کند.

Available_cells : به چه خانه هایی میتواند حرکت کند.

Found_gold : آیا تا این جای کار طلا را پیدا کرده یا خیر.

Wampus_alive : آیا Wampus زنده ای وجود دارد یا خیر.

Cell_data : دیکشنری مربوط به اطلاعات نقطه ای که در آن هستیم.

تمامی این اطلاعات در ابتدا با مقادیر پیش فرض تنظیم شده اند اما بعد از هر حرکت تمامی این مقادیر update میشوند.

توابع مربوط به کلاس Agent عبارتند از:

Find_available_moves : این تابع با توجه به موقعیت ما در ماتریس جهت هایی که میتوانیم به آن ها حرکت کنیم و در داخل ماتریس بمانیم را برمیگرداند و طوری طراحی شده که همیشه جهت بازگشت به خانه قبلی در آخر لیست باشد.

Find_available_cells : با توجه به مقدار Location و Available_moves خانه هایی که میتوانیم به آن ها حرکت کنیم را برمیگرداند.

Which_direction(start,end) : این تابع برای تشخیص جهت حرکت طراحی شده به این شکل که نقطه آدرس نقطه ای که در آن هستیم و آدرس خانه ای میخواهیم به آن برویم را به تابع پاس میدهیم و در خروجی جهتی که باید به آن سمت حرکت کنیم نمایش داده میشود.

Die : بررسی میکند که آیا در خانه ای که هستیم pit یا wampus وجود دارد یا خیر ، اگر وجود داشت میمیرد.

Is_ok : بررسی میکند که ببیند آیا در خانه حال حاضر نشانه ای از وجود Wampus و Pit در خانه های مجاور وجود دارد یا خیر..

Shoot(direction) : در جهت direction تمام خانه ها را بررسی میکند و اگر Wampus در خانه های این جهت وجود داشت Wampus را از روی World پاک میکند و Stench های خانه های مجاور را نیز حذف میکند.

Grab : بررسی میکند که آیا در خانه حال حاضر Gold وجود دارد یا خیر ، اگر وجود داشت Gold را از نقشه حذف میکند و gold_found را ۱ میکند.

Move(direction) : در جهتی که گفته شده Agent را در world جابجا میکند و مقادیر visited و route و route_full را update میکند و پس از جابجا شدن بررسی میکند که آیا میمیرد یا خیر ، آیا طلا را پیدا میکند یا خیر ، آیا خانه جدید Safe است یا خیر ، و در خانه جدید مقادیر available_moves و available_cells و cell_data را نیز update میکند

Show() : این تابع برای نمایش دادن واضح تر World در خروجی نوشته شده است که بتوانیم اتفاقات را راحت تر در آن بررسی کنیم.

- **Logic**

Delete_duplicate(list) : این تابع یک لیست در ورودی میگیرد و مقادیر تکراری را از لیست حذف میکند و برمیگرداند.

Find_wampus(data) : ورودی این تابع اطلاعاتی است که تا به حال از خانه هایی که visit کرده ایم ذخیره کرده ایم و این تابع در صورتی که این اطلاعات کافی باشد آدرس دقیق Wampus را به ما برمیگرداند و این آدرس تنها در صورتی قطعی است که حداقل سه خانه مجاور خانه هدف را چک کرده باشیم و در آن ها Stench وجود داشته باشد. در غیر اینصورت 'not-found_wampus' در خروجی ظاهر میشود.

Kill_wampus(agent,wampus_location_data) : این تابع با توجه به موقعیت Agent نسبت به محل Wampus در نقشه تابع shoot را با جهت مناسب صدا میکند و در صورت کشته شدن wampus خانه های مجاور این خانه را از Stench پاکسازی میکند. این تابع را فقط در صورتی میتوان صدا کرد که خودمان روی یکی از خانه های مجاور wampus وجود دارد باشیم.

Go_kill_wampus(agent,wampus_location_data) : مانند تابع بالا عمل میکند فقط با این فرق که اگر زمانی که از حضور wampus در یک خانه به خصوص مطمئن میشویم خودمان در یکی از خانه های مجاور این wampus نباشیم با استفاده از مسیر های قبلی که طی کردیم به یکی از خانه های مجاور wampus ، move میکنیم و سپس با فراخوانی تابع kill_wampus اقدام به تیر زدن میکنیم.

Go_in(Agent) : این تابع ، تابع اصلی ما است که به صورت DFS شروع به گشتن تمام خانه های قابل دسترسی میکند و تا زمانی این کار را انجام میدهد که یا Gold را پیدا کرده باشیم یا تعداد move های Agent از ۱۰۰ حرکت بیشتر شود در این شرایط از تابع بیرون می آییم و خروجی را نمایش میدهیم که در صورت رسیدن به Gold خروجی ما نزدیک ترین مسیر برای رسیدن به Gold و راه برگشتن به خانه ۰و۰ را نمایش میدهد و در غیر این صورت Nope خروجی ما خواهد بود که به این معنی است که تعداد move های ما از حد تعیین شده بیشتر شده است چون در الگوریتم پیاده سازی شده Agent هیچ گاه به خانه های نا امن سفر نمیکند که این کار باعث مرگش شود.

در این تابع در هر خانه شروع میکنیم و به تمام مسیر های ممکن سفر میکنیم و ترجیحمان این است که به خانه های visit شده نرویم مگر این که خانه visit شده دیگری وجود نداشته باشد و در نهایت اگر از مسیر جهت های ممکن دستاوردی نداشتیم به سمت آخرین جهت حرکت میکنیم که همان جهت خانه ای است که از آن به خانه حال حاضر آمده ایم.

در هر بار صدا کردن تابع دو حالت پیش می آید ، یا خانه جدید Safe است که تابع را دوباره صدا میکنیم یا unsafe است که در این صورت با توجه به دلیل unsafe بودن خانه جدید که سه حالت کلی دارد یا فقط Stench حس میکنم یا فقط Breeze حس میکنیم و یا هردو که با توجه به این که در کدام یکی از این شرایط هستیم لیست های مربوط به possible_wampus و possible_pit را update میکنیم که با استفاده از اطلاعات موجود در این دو لیست و دیتای ذخیره شده از خانه های قبلی که از آن ها گذشته ایم و قوانین مربوط به خود بازی که مثلا اگر در خانه ای wampus باشد باید ۴ خانه مجاور آن دارای S باشند پس با چک کردن تمام گزینه های موجود در possible_wampus فرض میکنیم که این خانه wampus واقعی باشد و با قانون mp به این نتیجه میرسیم که چهار خانه مجاور باید دارای S باشد پس دیتای ذخیره شده را بررسی میکنیم که آیا با اطلاعاتی که از این خانه ها داشتیم مطابقت میکند یا خیر ، اگر مطابقت نکرد میتوانیم این گزینه را از لیست possible_wampus حذف کنیم و هر جا که از حضور wampus در یکی از خانه ها مطمئن شدیم با یکی از توابع kill_wampus یا go_kill_wampus سعی در کشتن wampus میکنیم و در نهایت دوباره این تابع را صدا میکنیم.