Smart Agent Simulation

ماهیت این پروژه را در واقع می توان این­گونه در نظر گرفت که می­خواهیم یک عامل هوشمند پیاده سازی کنیم که تعدادی عناصر از عناصر در مختصات های تصادفی را پیدا کند و آن ها را تا مقاصدی تصادفی – که حفره- نامیده می­شوند- جا به جا کنیم.

ما این پروژه را به دو صورت پیاده کرده ایم:

1. رابط کاربری تحت کنسول (Console App)

در حالت کنسول اپ، ایجنت(های) ما A می باشند که گوی ها را - که با O مشخص می شوند- پیدا کرده و تا حفره ها – که با H مشخص شده اند – می برند.

* عکس

1. رابط کاربری گرافیکی (GUI App)

در این حالت، باب اسفنجی (ها) به دنبال گوی هایی که در ابتدا نمی­داند کجاست، میگردد و هر کدام را که پیدا کرد به داخل حفره هایی که میشناسد منتقل میکند.

* عکس

توضیح کد پروژه

این پروژه کاملا با رعایت اصول شی گرایی نوشته شده است. تمام اشیاء داخل بازی کلاس مخصوص خود را دارند و در هر جا که نیاز بوده است از ارث بری استفاده شده است. کد برنامه تقریبا فقط با استفاده از کتابخانه های استاندارد پایتون نوشته شده است. فقط در رابط گرافیکی از کتابخانه های خارجی tkinter برای طراحی پنجره گرافیکی و ایتم های آن و Pillow برای بارگذاری عکس ها استفاده شده است.

ماژول movement

این ماژول حاوی دو کلاس بسیار پرکابرد است که در همه جای این پروژه استفاده شده­اند. اولین کلاس آن کلاس Coordinates است که همان­طور که از نامش برمی­آید نماینده مختصات اشیا داخل پروژه است. تمام اشیاء اصلی پروژه –گوی، حفره و عامل هوشمند- داری فیلدهایی مانند position هستند که از جنس این کلاس Coordinates اند. این کلاس دارای دو متغیر(یا همان فیلد در مبحث کلاس) اصلی x و y هست که مشخص کننده مکان فعلی یک شی است. همچنین این کلاس دارای توابع(یا همان متد در حوزه کلاس) می­باشد که جهت مقایسه مکان ها و نمایش آن ها استفاده می­شوند.

این کلاس یک تابع Random دارد که که برای تولید مختصات تصادفی در محدوده خواسته شده برای x و y و که با پارامترهای x\_max و y\_max مشخص شده­اند استفده می­شود. تفاوت این تابع با تابع Randomize مهم است. تابع Random یک تابع استاتیک است که یک شی جدید از نوع Coordinates تولید می کند که حاوی مختصات تصادفی ست؛ اما تابع Randomize یک تابع عضو (member method) می باشد که برای تصادفی کردن مختصات شی در حال استفاده می­باشد و شی جدیدی تولید نمی­کند.

همچنین در این ماژول یک Enum داریم؛ که البته در زبان پایتون Enum هم از جنس کلاس می­باشد و برخلاف زبان­های دیگر یک ماهیت جداگانه نیست. این Enum که با نام Direction تعریف شده است در کلاس Agent استفاده می­شود و در خرلحظه جهت عامل هوشمند را مشخص می­کند که با چهار مقدار RIGHT – LEFT – UP –DOWN مشخص میگردد. هم چنین این Enum یک تابع Random دارد که برای تولید جهت تصادفی استفاده می­شود، که در ابتدای بازی برای تولید جهت تصادفی عامل ها استفاده شده است.

کلاس Entity

این کلاس در فایل entity.py تعریف شده است و بصورت مستقیم استفاده نشده است؛ یعنی بعنوان کلاس پدر برای اشیا داخل بازی استفاده شده است. این کلاس شامل مشخصه ها و رفتارهای مشترک گوی ها، حفره­ها و عامل هوشمند می­باشد.

از جمله فیلدهای مشترک اشیا که در این کلاس تعریف شده اند می توان به id و name اشاره کرد که برای تمایز بین اشیا از یک دسته تعریف شده اند. فیلد position که می­توانم گفت تقریبا مهم­ترین مشخصه اشیا است نیز در این کلاس تعریف شده است و از طریق ارث بری در تمامی اشیا هم تعریف می شود و بدین تریت جایگاه همه اشیا در زمین بازی مشخص می گردد. همچنین avatars که لیست عکس های درنظر گرفته شده برای اشیا مختلف است. و از توابع این کلاس می توان به GetNextId اشاره کرد که با دریافت لیستی از اشیاء یک دسته، آیدی عنصر بعدی آن را پیدا می­کند.

همچنین این کلاس یک فیلد(متغیر) بسیار مهم دارد به نام identified. این متغیر در واقع آیدی عامل هوشمندی که آن را شناسایی کرده است را در خود ذخیره می­کند. درصورتی که مقدار آن صفر باشد به این معناست که فعلا توسط هیچ عاملی شناسایی نشده است.

کلاس Orb

این کلاس در فایل orb.py برای شبیه سازی گوی ها تعریف شده است که از کلاس Entity ارث بری می­کند؛ بنابراین علاوه بر فیلدها و توابع مشترک موجود در کلاس پدر یک فیلد اضافی به نام hole دارد. این فیلد یا مقداری ندارد (مقدار None در پایتون) و یا مساوی با ابجکت حفره ای که این گوی را دربردارد می­باشد. درواقع عامل هوشمند از این طریق متوجه می­شود که آیا این گوی خارج از حفره­ها هست یا نه. در صورتی که خارج بود آن را بعنوان یک کاندید احتمالی برای جابه­جایی درنظر می­گیرد؛ البته این درصورتی است که از قبل توسط عامل شناسایی شده باشد.

این شی در حالت کنسول با حرف O و آیدی هر گوی (مانند O1) و در حالت گرافیکی با عکس orb.png که در پوشه resources قرار دارد مشخص شده است. درواقع فیلد avatar این آبجکت از این عکس استفاده میکند.

توجه: فیلد avatars لیستی از اشیاء از نوع کلاس Avatar می باشد. کلاس Avatar در فایل resources/avatar.py تعریف شده است که حاوی آدرس عکس استفاده شده و سایز آن برای نمایش در حالت گرافیکی است. همچنین یک فیلد canvas\_id دارد که وقتی یک شی در پنجره گرافیکی نمایان شد، آیدی المان آن (که مربوط به کتابخانه tkinter است) را ذخیره می کند.

کلاس Hole

این کلاس هم از کلاس Entity ارث می­برد و همانند کلاس orb.py نوشته شده است با کمی تفاوت جزئی. اول اینکه این کلاس در ماژول hole.py تعریف شده است؛ همچنین به جای فیلد hole در کلاس Orb، این کلاس فیلد orbs را داراست که لیستی از گوی­هایی که درون این حفره گنجانده شده­اند را شامل می­شود. این بخش بصورت پویا نوشته شده است و یک فیلد استاتیک به­نام Hole.CAPACITY موجود است که نشان می­دهد هر حفره حداکثر چند گوی را می­تواند نگه دارد. چون در این برنامه این مقدار یک است پس این لیست حداکثر یک عضو دارد.

همچنین این کلاس یک تابع دارد به­نام has\_room. درواقع این تابع چک می­کند آیا داخل این حفره جا برای گوی جدید هست یا نه. این امر با چک کردن تعداد اعضای فیلد orbs میسر می­شود؛ بدین صورت که اگر این تعداد کمتر از CAPACITY – که در پروژه ما یک است- باشد پس جا برای گوی جدید دارد؛ درواقع عامل هوشمند از طریق این تابع می­تواند تشخیص دهد که آیا این حفره قابل استفاده است یا نه.

ماژول agent

این ماژول شامل دو کلاس است. کلاس اول کلاس Candidate می باشد. این کلاس برای جابه­جایی بهینه گوی ها به سمت حفره ها استفاده می­شود. به این صورت که یک عامل هوشمند در هر مرحله بین ایا شناسایی شده اطرافش بررسی می­کند و نزدیک­ترین گوی به خودش را به نزدیک­ترین حفره به آن گوی وصل می­کند. این کار از طریق ساخت یک آبجکت جدید از نوع Candidate انجام می­گیرد که این کلاس دو فیلد hole و orb را برای همین منظور داراست.

این کلاس همچنین تابع distance را دارد که فاصله گوی از حفره را مشخص می­کند؛ و متد drop نیز بررسی می­کند که اگه حفره مورد نظر خالی است گوی انتخابی را در حفره بیاندازد. این کار از طریق مقداردهی orb.hole با hole انتخابی و اضافه کردن orb انتخابی به لیست hole.orbs انجام می­گیرد. این توابع فقط زمانی فراخوانی می­شود که مکان این حفره و گوی دقیقا یکسان شود و یا به نوعی خروجی تابع distance شی Candidate صفر باشد.

کلاس Agent

ماژول مذکور همچنین شامل کلاس Agent است که همان عامل هوشمند می­باشد. این کلاس هم دوباره از کلاس Entity ارث­بری می­کند و فیلدهای مشترک آن­ها را داراست. یکی از تفاوت­های این کلاس در این است که لیست avatars آن 4 عضو دارد. درواقع در این کلاس این فیلد بصورت دیکشنری پایتون (dict) تعریف شده است که به ازای هر Direction یک آواتار دارد و هر شی آواتار یک عکس در جهت مرتبط دارد. سپس یک مشخصه (property) به نام avatar دارد که با توابع جهت کنونی عامل هوشمند آواتار مرتبط را مشخص می­کند. پس این کلاس یک فیلد direction نیز دارد (همان طور که قبلا درباره enum Direction گفته شده بود.) که جهت حرکت عامل هوشمند درهر لحظه را مشخص می­کند. فیلد بسیار مهم دیگر این کلاس moves است که به ازای هر تغییر مکان یک عامل هوشمند یکی به مقدار آن اضافه می­شود.

محدودیت تعداد حرکتی برنامه هم بصورت پویا می­باشد؛ بدین صورت که یک فیلد Game.MAX\_MOVES داریم ]کلاس Game بزودی بررسی می­شود[ و تا زمانی که فیلد moves یک عامل کمتر از این مقدار باشد بازی ادامه دارد. فیلد دیگر candidate می­باشد که از جنس Candidate می­باشد و کاربرد آن بصورت کامل در بخش قبلی بررسی شده است. چند تابع مهم در این کلاس وجود دارد که باید به تفصیل بررسی گردند: