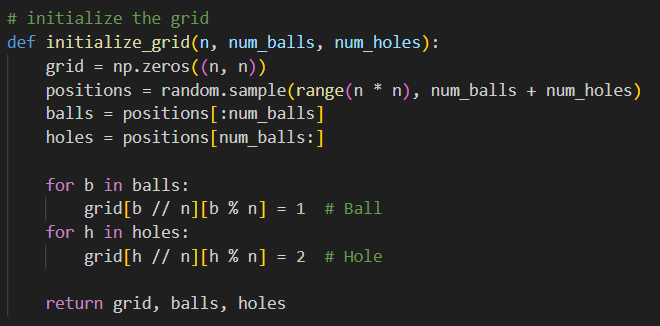
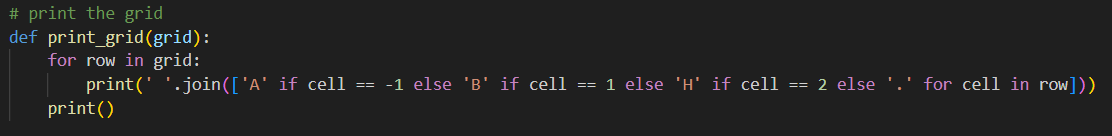
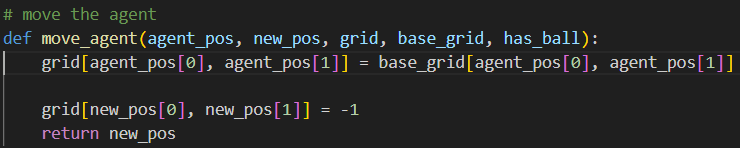
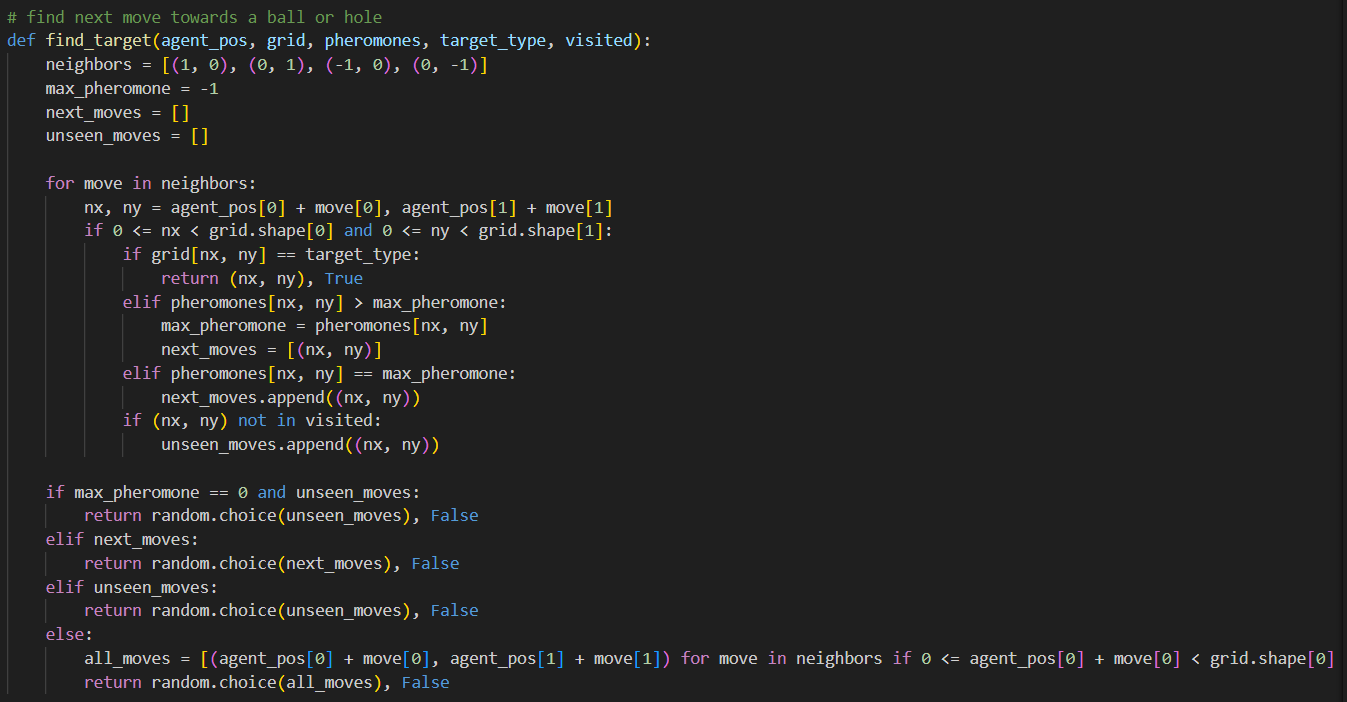
سبا رضی

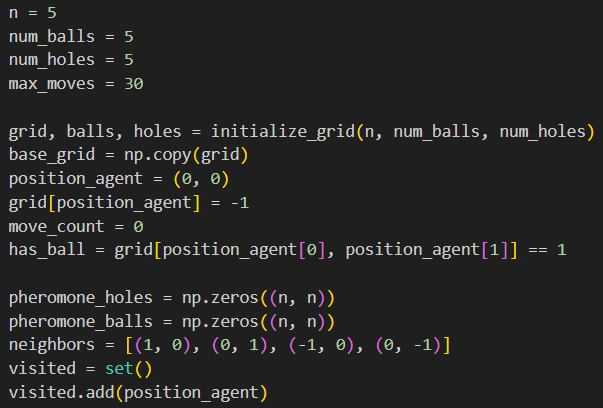
احسان احمدپور

اولین مرحله ، ساخت زمین بازی با استفاده از n گوی و n حفره است.برای انجام این کار، در تابع initialize\_grid() ابتدا یک آرایه ی دو بعدی n در n ساخته، یک لیست positions ایجاد می‌کنیم که شامل موقعیت‌های تصادفی از آزایه است. range(n \* n) تمام موقعیت‌های ممکن در شبکه n × nرا ایجاد می‌کند (از 0 تا n2-1) و در ادامه به صورت تصادفی num\_balls + num\_holes عدد را از این مجموعه انتخاب می‌کند. سپس از لیست تولیدی به تعداد num\_balls از اولین اعداد آن استفاده کرده و مقدار 1 در grid بازی قرار میدهیم و num\_holes از دومین اعداد لیست استفاده کرده و در grid بازی 2 قرار میدهیم.

برای نشان دادن زمین بازی هم از تابع print\_grid() استفاده میکنیم.

در تابع move\_agent() هم حرکت عامل هوشمند در زمین را پیاده سازی میکنیم. این تابع عامل را از موقعیت فعلی به موقعیت جدید حرکت می‌دهد. یعنی در موقعیت جدید عامل -1 که همان A است گذاشته و در موقعیت قبلی همان مقداری که در ابتدا داشتیم را میگذارد.

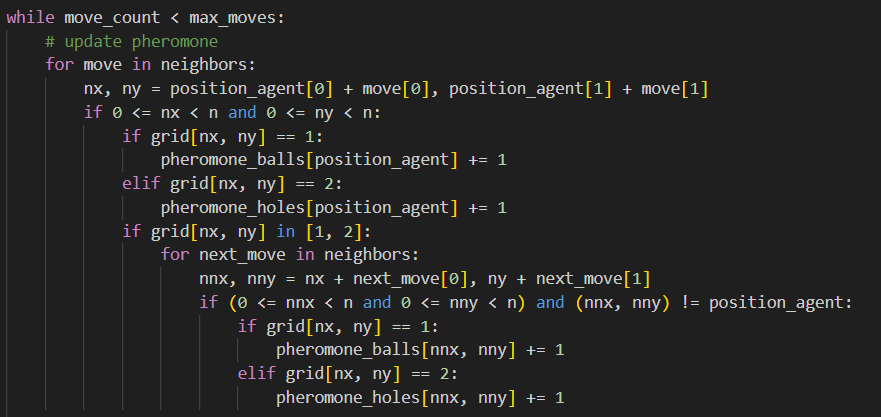
در تابع find\_target() در حقیقت انتخاب میشود که عامل به کدام سمت حرکت کند. ابتدا همسایه‌های نزدیک را بررسی می‌کند و اگر هدف (توپ یا سوراخ) را پیدا کند، آن موقعیت را برمی‌گرداند. در غیر این صورت، همسایه‌هایی را که بیشترین فرومون را دارند انتخاب می‌کند و اگر هیچ‌کدام از همسایه‌ها فرومون نداشتند، یکی از همسایه‌های دیده نشده را به صورت تصادفی انتخاب می‌کند.



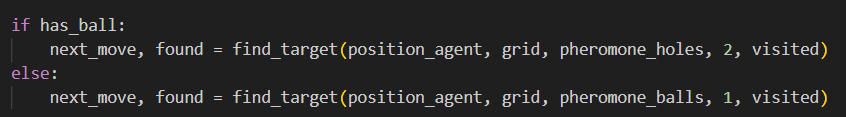
حال تابع main() بازی را توضیح میدهیم . ابتدا پارامتر های اصلی بازی را تعریف میکنیم. زمین بازی را ساخته. و یک کپی از زمین اولیه بازی جهت استفاده در تابع move\_agent() میسازیم. متغیرهای مورد نیاز شامل تعداد حرکت‌ها، اینکه عامل توپ دارد یا نه، فرومون‌ها و همسایه‌ها تنظیم می‌شوند. همچنین مجموعه‌ای از موقعیت‌های بازدید شده ایجاد می‌شود.

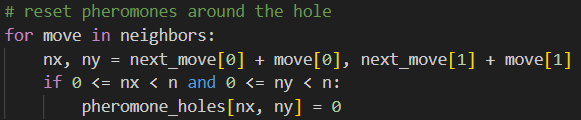
الگوریتم استفاده شده:

در اینجا از الگوریتمی مشابه الگوریتم کلونی مورچگان استفاده میکنیم اما با تغییرات بیشتر. به ازای هر خانه ما دو فرومون نگه داری میکنیم. یکی فرومون های حفره ها و یکی فرومون گوی ها. هر بار که عامل به یک خانه میرود، با توجه به 8 خانه ی اطرافش، این دو مقدار فرومون را آپدیت میکند به این صورت که هر یک حفره یا گویی که در 8 خانه ی اطرافش باشد یکی به مقدار فرومون آن اضافه میکند.البته این کار را برای همسایگان همسایه هایش هم انجام میدهد.

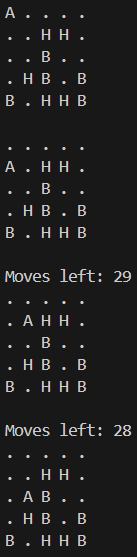
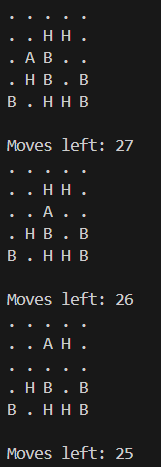
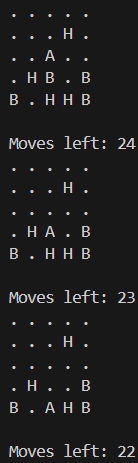
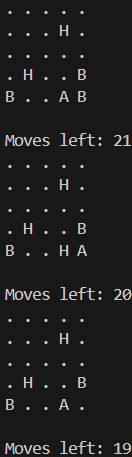
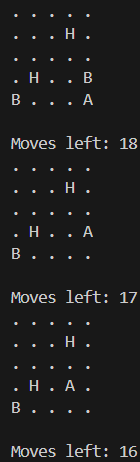
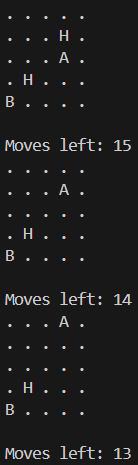
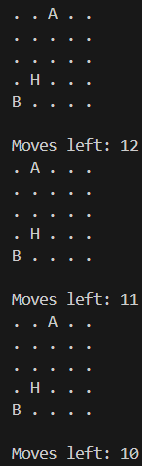
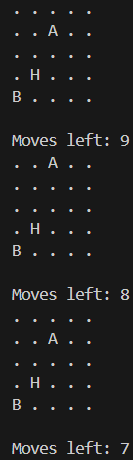
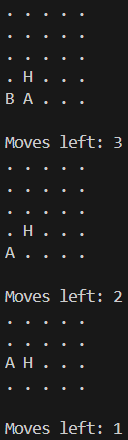
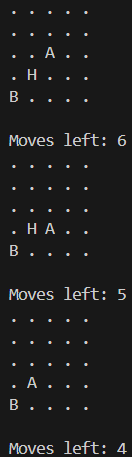
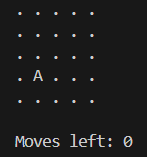


پس از آن، هنگام انتخاب خانه ی بعدی چک میکند اگر در دستش گوی داشته باشد که به دنبال حفره میرود و در غیر این صورت به دنبال گوی میرود. برای اینکار هم چک میکند که اگر 4 خانه ی همسایه حفره یا گوی داشت، به آن رفته و اگر نه به ماکسیسم فرومون اطرافش برود. البته در اینجا چک میکند در صورتی که ماکسیمم فرومون 0 باشد یعنی در قسمتی باشد که قبلا ندیده و یا اطرافش هیچ گوی یا حلقه ای نیست، پس بجای رفتن در خانه های تکراری، به خانه های جدید حرکت کند.



پس از انتخاب حرکت، اگر به گوی یا حفره رسید، چون هر دوتای آن ها دیگر نمیتوانند استفاده شود، مقدار فرومون آن را از 8 خانه ی اطرافشان یکی کم کرده تا دوباره به آن سمت عامل حرکت نکند.

خروجی 5 در 5 با 5 گوی و 5 حفره:



خروجی در زمین 5 در 5 و 10 گوی و 10 حفره:

