prolog

برنامه از الگوریتم جستجوی هزینه یکنواخت برای پیداکردن کوتاه‌ترین مسیر استفاده می‌کند و مسیر را به یک لیست از اقدامات تبدیل می‌کند در کد پرولوگ عملیات زیر انجام می‌شود

بارگذاری و تجزیه نقشه **(Map Loading and Parsing)**

* **read\_map(File)**: این بخش از کد مسئول خواندن فایل نقشه است. فایل نقشه شامل یک شبکه (گرید) است که موقعیت پرنده، خوک‌ها، سنگ‌ها و سلول‌های خالی را مشخص می‌کند. اگر در حین خواندن فایل خطایی رخ دهد، پیام خطا نمایش داده می‌شود.
* **read\_grid(Stream, Row)**: این بخش هر خط از فایل را می‌خواند و آن را به‌عنوان یک‌رشته پردازش می‌کند. اگر خط به پایان فایل نرسیده باشد، خط پردازش می‌شود و شماره سطر افزایش می‌یابد.
* **process\_line(Line, Row, Col)**: این بخش هر کاراکتر از خط را پردازش می‌کند. اگر کاراکتر معتبر باشد (مانند B برای پرنده، P برای خوک، R برای سنگ، و T برای سلول خالی)، موقعیت آن در شبکه ثبت می‌شود. اگر کاراکتر ناشناخته باشد، پیام خطا نمایش داده می‌شود.

**۲. پردازش سلول‌ها (Cell Processing)**

* **process\_cell** : این بخش کاراکترهای مختلف را پردازش می‌کند و موقعیت آن‌ها را در شبکه ثبت می‌کند:
  + **B** : موقعیت پرنده را تنظیم می‌کند.
  + **P** : موقعیت خوک را اضافه می‌کند.
  + **R** : موقعیت سنگ را اضافه می‌کند.
  + **T** : سلول خالی را نشان می‌دهد و هیچ عملی انجام نمی‌دهد.
  + **کاراکتر ناشناخته**: پیام خطا نمایش داده می‌شود.

**۳. مدیریت وضعیت بازی (State Management)**

* **: dynamic bird\_pos/2.**, **: dynamic pig\_pos/2.**, **: dynamic rock\_pos/2.** این بخش از کد، موقعیت‌های پرنده، خوک‌ها و سنگ‌ها را به‌صورت پویا تعریف می‌کند تا در طول اجرای برنامه تغییر کنند.
* **set\_bird :**موقعیت پرنده را تنظیم می‌کند و موقعیت قبلی را پاک می‌کند.
* **add\_pig :**  موقعیت خوک را اضافه می‌کند، اما فقط اگر قبلاً در آن موقعیت خوکی وجود نداشته باشد.
* **add\_rock :** موقعیت سنگ را اضافه می‌کند، اما فقط اگر قبلاً در آن موقعیت سنگی وجود نداشته باشد.

**۴. تعریف حرکات (Movement Definition)**

* **Move :** این بخش حرکات ممکن در گرید را تعریف می‌کند:
  + **up**: حرکت به بالا
  + **Down** : حرکت به پایین
  + **Left** : حرکت به چپ
  + **Right** : حرکت به راست
* **valid\_pos :** این بخش بررسی می‌کند که آیا یک موقعیت در گرید معتبر است یا نه. یک موقعیت معتبر باید در گرید 8 در 8 باشد و روی سنگ قرار نگیرد.

**۵. جستجوی مسیر بهینه (Pathfinding)**

* **find\_path :** این بخش مسیر بهینه را برای رسیدن پرنده به تمام خوک‌ها پیدا می‌کند. ابتدا موقعیت پرنده و خوک‌ها را پیدا می‌کند، سپس مسیر بهینه را برای هر خوک محاسبه می‌کند.
* **find\_optimal\_path:** این بخش مسیر بهینه را برای رسیدن به هر خوک محاسبه می‌کند. از الگوریتم جستجوی هزینه یکنواخت (Uniform Cost Search) استفاده می‌کند.
* **Ucs :** این بخش الگوریتم جستجوی هزینه یکنواخت را پیاده‌سازی می‌کند. این الگوریتم کوتاه‌ترین مسیر را از موقعیت فعلی پرنده به موقعیت خوک پیدا می‌کند.
* **ucs\_search :**این بخش جستجوی هزینه یکنواخت را انجام می‌دهد و مسیر بهینه را برمی‌گرداند.

**۶. تبدیل مسیر به اقدامات (Convert Path to Actions)**

* **convert\_path\_to\_actions :** این بخش مسیر پیدا شده را به یک لیست از اقدامات (حرکات) تبدیل می‌کند. هر حرکت با یک عدد مشخص می‌شود:
  + **0** : حرکت به بالا.
  + **1** : حرکت به پایین.
  + **2** : حرکت به چپ.
  + **3** : حرکت به راست.
* **get\_direction\_number :** این بخش جهت حرکت بین دو سلول را مشخص می‌کند و آن را به یک عدد تبدیل می‌کند.

python

کد آی پایتون از کتابخانه‌های pyswip  و pygame استفاده می‌کند. در این گزارش، هر بخش از کد به طور کامل توضیح داده شده است.

**بارگذاری و اجرای کد پرولوگ (Prolog Integration)**

* **prolog = Prolog()**: یک شیء از کلاس Prolog  ایجاد می‌شود که امکان اجرای کد پرولوگ در پایتون را فراهم می‌کند.
* **prolog.consult("x.pl")**: فایل پرولوگ x.pl که حاوی منطق بازی و الگوریتم‌های مسیریابی است، بارگذاری می‌شود.
* **absolute\_path = str(Path(f'Env/templates/ {template}.txt').resolve()).replace('\\', '/')**: مسیر فایل نقشه (که موقعیت پرنده، خوک‌ها و سنگ‌ها را مشخص می‌کند) به‌صورت مطلق و با فرمت مناسب برای پرولوگ آماده می‌شود.
* **list(prolog.query(f"read\_map(' {absolute\_path} ') "))**: نقشه از فایل خوانده می‌شود و موقعیت‌ها در پرولوگ ثبت می‌شوند.

**دریافت موقعیت‌ها از پرولوگ (Querying Prolog for Positions)**

* **print(list(prolog.query('pig\_pos(X, Y)')))**: موقعیت تمام خوک‌ها از پرولوگ دریافت و چاپ می‌شود.
* **bird\_pos = env.get\_bird\_position()**: موقعیت پرنده از محیط بازی دریافت می‌شود.
* **print(list(prolog.query('bird\_pos(X, Y)')))**: موقعیت پرنده از پرولوگ دریافت و چاپ می‌شود.
* **print(list(prolog.query('findall((PX, PY), pig\_pos(PX, PY), Pigs)')))**: موقعیت تمام خوک‌ها از پرولوگ دریافت و چاپ می‌شود.

**یافتن مسیر بهینه (Finding the Optimal Path)**

* **solutions = list(prolog.query(f"find\_path(Actions)"))**: مسیر بهینه برای رسیدن پرنده به تمام خوک‌ها از پرولوگ دریافت می‌شود.
* **lengths = [len(d ['Actions']) for d in solutions]**: طول هر مسیر محاسبه می‌شود.
* **min\_length = min(lengths)**: کوتاه‌ترین طول مسیر پیدا می‌شود.
* **optimal\_solutions = [d for d in solutions if len(d ['Actions']) == min\_length]**: مسیرهای بهینه (با کمترین طول) انتخاب می‌شوند.
* **actions = optimal\_solutions [0] ['Actions']**: اولین مسیر بهینه انتخاب می‌شود.
* **print(actions)**: اقدامات (حرکات) مربوط به مسیر بهینه چاپ می‌شوند.
* **print(lengths)**: طول‌های تمام مسیرها چاپ می‌شوند.
* **print(solutions)**: تمام مسیرهای پیدا شده چاپ می‌شوند.