Zadanie 2: Instrukcja

Opis

Zadanie polega na zaimplementowaniu symulacji, w której wilk łapie owce.

Symulacja obejmuje dwa rodzaje zwierząt: pojedynczego wilka oraz pewną grupę owiec. Zwierzęta te poruszają się po nieskończonej łące, na której nie występują żadne przeszkody terenowe: wilk goni owce, próbując je złapać, a następnie pożreć, z kolei owce próbują (dość nieporadnie) uciekać przed wilkiem. Wobec powyższych założeń łąka reprezentowana jest jako nieograniczona przestrzeń dwuwymiarowa o środku położonym w punkcie (0,0; 0,0), a zatem za pomocą kartezjańskiego układu współrzędnych, natomiast pozycję każdego zwierzęcia na łące określa para liczb zmiennoprzecinkowych (współrzędne mogą być zarówno dodatnie, jak i ujemne).

Na początku symulacji następuje określenie pozycji początkowych wszystkich zwierząt. Pozycja początkowa każdej owcy ustalana jest w sposób losowy, gdzie każda współrzędna losowana jest z zakresu [`-init_pos_limit`, `init_pos_limit`] (przy czym losowane wartości mają postać liczb zmiennoprzecinkowych i nie muszą być całkowite), natomiast pozycja początkowa wilka jest to środek łąki, czyli punkt (0,0; 0,0).

Przebieg symulacji ma charakter turowy. Podczas każdej tury najpierw po kolei wykonują ruchy wszystkie żyjące owce, a dopiero na końcu wykonywany jest ruch przez wilka. Na początku swojego ruchu owca wybiera losowo jeden z czterech kierunków: północ (góra), południe (dół), wschód (prawo), zachód (lewo), a następnie porusza się w wylosowanym kierunku o odcinek o długości `sheep_move_dist`. Z kolei wilk w ramach swojego ruchu najpierw określa, która owca znajduje się najbliżej niego, po czym sprawdza, czy nie znalazła się czasem w zasięgu jego ataku, czyli w odległości mniejszej niż `wolf_move_dist`: jeśli tak - to ją pożera, co oznacza, że owca znika, natomiast jeśli nie - to rusza za nią w pogoń, poruszając się w jej kierunku o odcinek o długości `wolf_move_dist` (jeżeli w tej samej najbliższej odległości znajdzie się więcej niż jedna owca, podejmowane czynności dotyczą tylko pierwszej z nich). Symulacja kończy się albo kiedy wszystkie owce zostaną pożarte, albo kiedy minie z góry określona liczba tur.

Wymagania na ocenę dostateczną

1. Zaimplementować symulację zgodnie z powyższym opisem, przyjmując następujące wartości:
    - liczba tur: 50;
    - liczba owiec: 15;
    - `init_pos_limit`: 10.0;
    - `sheep_move_dist`: 0.5;
    - `wolf_move_dist`: 1.0.

    W celu generowania liczb losowych skorzystać z pakietu *random* z biblioteki standardowej. W przypadku konieczności wykonywania bardziej zaawansowanych obliczeń matematycznych skorzystać z pakietu *math* z biblioteki standardowej.

2. Zaimplementować wyświetlanie podstawowych informacji o stanie symulacji na zakończenie każdej tury. Informacje te mają obejmować:
    - numer tury;

- pozycję wilka (z dokładnością do trzeciego miejsca po przecinku każdej współrzędnej);
- liczbę żywych owiec;
- jeżeli któraś z owiec została pożarta - informację o tym fakcie wraz z określeniem, która to była owca (jej numer porządkowy).

Wyświetlenie powyższych informacji nie powinno zatrzymywać dalszego przebiegu symulacji - żadna interakcja z użytkownikiem nie jest tu zakładana.

3. Korzystając z pakietu *json* z biblioteki standardowej, zaimplementować zapisywanie pozycji każdego zwierzęcia podczas każdej tury do pliku `pos.json`. Zawartość pliku ma stanowić lista, której elementami będą słowniki, gdzie każdy słownik odpowiadać będzie pojedynczej turze symulacji i zawierać będzie następujące elementy:
   - `'round_no'` - numer tury (liczba całkowita);
   - `'wolf_pos'` - pozycja wilka (para liczb zmiennoprzecinkowych);
   - `'sheep_pos'` - pozycje wszystkich owiec (lista zawierająca pary liczb zmiennoprzecinkowych w przypadku żywych owiec albo wartość `None`/`null` w przypadku owiec, które zostały pożarte).

   Byłoby przy tym pożądane, aby zawartość pliku miała postać sformatowaną, to znaczy była zapisana w kolejnych liniach o odpowiednich wcięciach. Jeśli plik `pos.json` już istnieje, powinien zostać nadpisany.

4. Korzystając z pakietu *csv* z biblioteki standardowej, zaimplementować zapisywanie liczby żywych owiec podczas każdej tury do pliku `alive.csv`. Plik ten ma się składać z dwóch kolumn przechowujących następujące wartości:
   1. numer tury (liczba całkowita);
   2. liczba żywych owiec (liczba całkowita).

   Każdej turze ma w tym pliku odpowiadać jeden wiersz (rekord). Jeśli plik `alive.csv` już istnieje, powinien zostać nadpisany.

Wymagania na ocenę dobrą

1. Zrealizować wszystkie wymagania na ocenę dostateczną.

2. Korzystając z pakietu *argparse* z biblioteki standardowej, zaimplementować przetwarzanie następujących argumentów wywołania programu przekazywanych z linii poleceń:
    - `-c/--config FILE` - dodatkowy plik konfiguracyjny, gdzie `FILE` - nazwa pliku;
    - `-d/--dir DIR` - podkatalog, w którym mają zostać zapisane pliki `pos.json`, `alive.csv` oraz - opcjonalnie - `chase.log`, gdzie `DIR` - nazwa podkatalogu;
    - `-h/--help` - pomoc;
    - `-l/--log LEVEL` - zapis zdarzeń do dziennika, gdzie `LEVEL` - poziom zdarzeń (*DEBUG*, *INFO*, *WARNING*, *ERROR* lub *CRITICAL*);
    - `-r/--rounds NUM` - liczba tur, gdzie `NUM` - liczba całkowita;
    - `-s/--sheep NUM` - liczba owiec, gdzie `NUM` - liczba całkowita;
    - `-w/--wait` - oczekiwanie na naciśnięcie klawisza po wyświetlaniu podstawowych informacji o stanie symulacji na zakończenie każdej tury.

    Wszystkie argumenty przekazywane do programu mają mieć charakter opcjonalny i mają być uwzględniane według poniższych reguł.

    - Opcja `-c/--config`: Określa plik konfiguracyjny, w którym zapisane są wartości dla `init_pos_limit`, `sheep_move_dist` i `wolf_move_dist`. Format pliku konfiguracyjnego opisany jest poniżej. Jeżeli opcja ta nie zostanie podana, wartości dla `init_pos_limit`, `sheep_move_dist` i `wolf_move_dist` pozostają domyślne (zgodnie z wymaganiami na ocenę dostateczną).
    - Opcja `-d/--dir`: Określa podkatalog katalogu bieżącego, w którym mają zostać zapisane pliki `pos.json`, `alive.csv` oraz - opcjonalnie - `chase.log`. Pozwala zatem na rejestrowanie przebiegu różnych symulacji w różnych podkatalogach. Jeśli podany podkatalog nie istnieje, należy go utworzyć. Jeżeli opcja ta nie zostanie podana, pliki powinny zostać zapisane w katalogu bieżącym.
    - Opcja `-h/--help`: Wywołuje wyświetlenie pomocy programu, która obejmuje krótką informację o programie i jego parametrach. Podanie tej opcji powoduje zakończenie działania programu po wyświetleniu owej pomocy - symulacja nie jest w tej sytuacji przeprowadzana.
    - Opcja `-l/--log`: Określa poziom zdarzeń, które mają być zapisywane w dzienniku. Podanie tej opcji powoduje, że w katalogu bieżącym lub - jeśli została podana opcja `-d/--dir` - w odpowiednim podkatalogu katalogu bieżącego tworzony jest

plik dziennika o nazwie `chase.log`, w którym rejestrowane są zdarzenia z działania programu. Sposób rejestrowania zdarzeń opisany jest poniżej. Jeśli plik `chase.log` w danej lokalizacji już istnieje, powinien zostać nadpisany. Jeżeli opcja ta nie zostanie podana, plik `chase.log` nie jest tworzony i żadne zdarzenia nie są rejestrowane.

- Opcja `-r/--rounds`: Określa liczbę tur. Jeżeli opcja ta nie zostanie podana, liczba tur odpowiada wartości domyślnej (zgodnie z wymaganiami na ocenę dostateczną).
- Opcja `-s/--sheep`: Określa liczbę owiec. Jeżeli opcja ta nie zostanie podana, liczba owiec odpowiada wartości domyślnej (zgodnie z wymaganiami na ocenę dostateczną).
- Opcja `-w/--wait`: Określa, że po wyświetlaniu podstawowych informacji o stanie symulacji na zakończenie każdej tury dalszy przebieg symulacji powinien zostać zatrzymany aż do naciśnięcia przez użytkownika jakiegoś klawisza. Jeżeli opcja ta nie zostanie podana, po wyświetleniu tychże informacji przebieg symulacji nie powinien być zatrzymywany (tak jak w przypadku wymagań na ocenę dostateczną).

Należy sprawdzić, czy każda z podanych wartości argumentów spełnia odpowiednie ograniczenia (np. liczba tur jest liczbą całkowitą większą od 0), a jeśli nie - poinformować o błędzie za pomocą właściwego wyjątku.

3. Korzystając z pakietu *configparser* z biblioteki standardowej, zaimplementować wczytywanie wartości dla `init_pos_limit`, `sheep_move_dist` i `wolf_move_dist` z pliku konfiguracyjnego o nazwie podanej w ramach argumentu wywołania `-c/--config`. Plik konfiguracyjny ma mieć postać pliku INI o następującym formacie (podane wartości odpowiadają wartościom domyślnym, zgodnie z wymaganiami na ocenę dostateczną):

```
[Terrain]
InitPosLimit = 10.0

[Movement]
SheepMoveDist = 0.5
```

```
WolfMoveDist = 1.0
```

Należy sprawdzić, czy każda z podanych wartości parametrów spełnia odpowiednie ograniczenia (np. długość odcinka pokonywanego w trakcie jednej tury przez wilka jest liczbą dodatnią), a jeśli nie - poinformować o błędzie za pomocą właściwego wyjątku.

9. Korzystając z pakietu *logging* z biblioteki standardowej, zaimplementować rejestrowanie zdarzeń z działania programu do dziennika zapisywanego w pliku o nazwie `chase.log`. Rejestracji mają podlegać wszystkie zdarzenia o poziomie równym podanemu w ramach argumentu wywołania `-l/--log` lub od niego wyższym:
    - w przypadku poziomu *DEBUG* (10) rejestrowane mają być wszelkie wywołania funkcji wraz z argumentami oraz zwracane przez funkcje wartości (chyba że funkcja takowych nie zwraca);
    - w przypadku poziomu *INFO* (20) rejestrowane mają być informacje o działaniach podejmowanych przez program wraz z ewentualnymi szczegółami (np.: ustalenie pozycji początkowej zwierzęcia - wraz z podaniem tejże pozycji, wykonanie ruchu przez zwierzę - wraz z podaniem, z której pozycji na którą zostaje ono przemieszczone, pożarcie owcy przez wilka - wraz z podaniem, której owcy itp.);
    - w przypadku poziomów *WARNING* (30), *ERROR* (40) i *CRITICAL* (50) rejestrowane mają być odpowiednie zdarzenia zgodne z tymi poziomami.

Wymagania na ocenę bardzo dobrą

1. Zrealizować wszystkie wymagania na ocenę dobrą.
2. Korzystając z pakietu *distutils* z biblioteki standardowej, na bazie napisanego kodu stworzyć pakiet *chase*, który będzie można zainstalować w środowisku wirtualnym. Kod w ramach pakietu powinien być tak zorganizowany, aby można go było wykorzystać jako implementację logiki symulacji, którą dałoby się na przykład obudować graficznym interfejsem użytkownika (interfejs graficzny wywoływałby wówczas odpowiednie funkcje udostępniane przez pakiet w celu przeprowadzenia symulacji). Część kodu odpowiedzialna za przeprowadzenie symulacji w trybie konsolowym (zgodnie z wymaganiami na ocenę dobrą) powinna znaleźć się w osobnym module `__main__.py` w ramach pakietu: w tej sytuacji po zainstalowaniu pakietu uruchomienie symulacji w trybie konsolowym odbywać się będzie poprzez polecenie `python -m chase [ARG]`, gdzie `ARG` - opcjonalne argumenty wywołania zgodnie z wymaganiami na ocenę dobrą.

3. Podczas odpowiedzi zademonstrować tworzenie środowiska wirtualnego za pomocą pakietu *venv* z biblioteki standardowej oraz instalowanie stworzonego pakietu *chase* w tym środowisku, jak również uruchamianie symulacji w trybie konsolowym po zainstalowaniu pakietu.