GARBAGE TRUCK SIMULATOR

Projekt ze Sztucznej Inteligencji

Grupa nr 6

Członkowie projektu:

Dawid Wietrzych, Jacek Krakowski, Michał Romaszkin, Aleksander Chandrykowski, Jakub Kwiatkowski

1. TWORZENIE MAPY DYSKRETNEJ

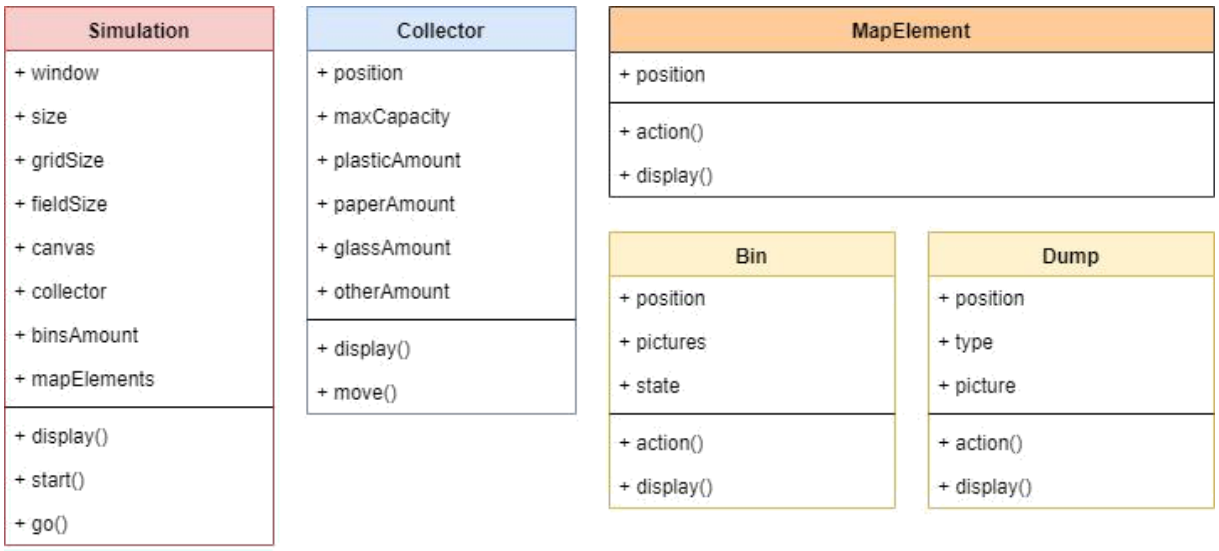
1.1 Informacje ogólne

Zgodnie z założeniami tematu projektu jakim jest stworzenie inteligentnej śmieciarki, która będzie się poruszać pomiędzy poszczególnymi domami i opróżniać zalegające w nich śmieci, nasza grupa obrała za pierwszy cel możliwość wygenerowania mapy dwuwymiarowej, na której znalazłyby się obiekty, takie jak:

- Drogi
- Śmietniki
- Trawa

1.2 Podjęte działania i środowisko programistyczne

Do zrealizowania naszego planu posłużyliśmy się językiem programowania Python. Zaczęliśmy od stworzenia kilku klas, które odpowiadają obiektom na mapie, takie jak: śmieciarka, śmietniki, droga, itd. Następnie zainicjowaliśmy w konstruktorach tych obiektów ich pozycję oraz odpowiednie obrazki (została ona pobrana w całości z Google Grafika). Dodaliśmy klasę symulacji, w której zawarliśmy obszar samej aplikacji okienkowej, a także całą logikę, która generuje nam poszczególne obiekty na mapie. Poza tym dodany został podstawowy ruch agenta – śmieciarki. Poniżej znajduje się grafika ilustrująca początkową (tj. może ulec zmianie) strukturę projektu



2. UKŁAD PROJEKTU

2.1 Składowe

A. Scripts

Folder zawierający wszystkie pliki źródłowe symulacji. Odpowiedzialne za poprawne działanie samej aplikacji okienkowej, generowania obrazów, czy w końcu logiki związanej z działaniami jakie podejmuję śmieciarką. Należą do nich klasy z konstruktorami poszczególnych obiektów:

- ***Bin.py***
- ***Collector.py***
- ***Dump.py***
- ***Grass.py***
- ***MapElement.py***
- ***Road.py***

A także plik z zawartością logiki do symulacji o nazwie ***Simulation.py***
Klasa ta posiada konstruktor oraz podstawowe metody kontrolujące symulację. Dzięki niej generujemy losowo (w dużej części) pozycję wszystkich domków, zieleni czy w końcu dróg po których porusza się nasza śmieciarka.
Ostatnim plikiem jest ***main.py*** jako środowisko uruchamiające cała symulacje gdzie też możemy ustalić ile domków (czyli potencjalnych miejsc gdzie śmieciarka ma się udać) ma się wygenerować na mapie.

B. Images

Tu znajdują się wszystkie potrzebne grafiki, które generowane są na mapie (domki, śmietniki, trawa, droga).

C. Data

Nic innego jak potrzebne dane do uczenia maszynowego, jak np. pozycje śmieciarki z wygenerowania kilkudziesięciu map.

D. Documentation

Plik zawierający dokumentację i raport z projektu.

2.2 Zasady działania agenta oraz samej symulacji

1. Śmieciarka:

- jest tworzona na podstawie strategii niepoinformowanej, czyli nie wie gdzie rozmieszczone są domki,
- nie może poruszać się po takich obiektach jak: trawa, domki, śmietniki,
- może poruszać się tylko po wygenerowanych drogach,
- śmieciarka startuje zawsze z określonego punktu (lewy górny róg mapy).

2. Drogi:
    - są generowane losowo w pozycji wertykalnej (pionowe), z kolei horyzontalne (poziome) są zawsze tworzone stale w równych odstępach,
    - drogi horyzontalne sięgają tzw. „Końca mapy” czyli granicy naszej siatki,
    - drogi wertykalne zawsze ograniczone są przez inne obiekty.
3. Kontenery:
    - generowane zawsze w lewym górnym rogu mapy.
4. Śmietniki:
    - tworzone są w zdefiniowanej ilości pomiędzy drogami i kontenerami.
5. Trawa:
    - jest wypełnieniem pozostałych wolnych miejsc na mapie.

3. ALGORYTM DFS (DEPTH-FIRST SEARCH)

3.1 Zdefiniowanie problemu

Celem zadania jest umożliwienie naszemu agentowi (śmieciarce) poruszania się, używając algorytmu przeszukiwania w głąb. Strategią tego algorytmu jest przechodzenie z wierzchołka początkowego, w naszym przypadku jest to pole w którym agent zaczyna swoją trasę, do pierwszych nieodwiedzonych pól, aż do momentu, kiedy nie ma możliwości dalszego poruszania się. Wtedy cofamy się do pola, z którego ostatnio przyszliśmy i przechodzimy do następnego nieodwiedzonego pola (o ile istnieje). Czynności te powtarzamy do momentu odwiedzenia wszystkich pól, co w naszym przypadku oznacza, że wyznaczyliśmy wszystkie drogi do danego punktu.

3.2 Rozwiązanie problemu

W celu uzyskania wyników, używamy tzw. *wrappera*, który uruchamia funkcję *dfs* odpowiedzialną za wyszukanie ścieżki. Funkcja *dfs* posiada w sobie stos, w którym zapisane ma współrzędne przebytych punktów. Na podstawie ostatniego elementu na stosie, sprawdzane są warunki możliwego przejścia do innych punktów. Jeżeli można iść dalej, to współrzędna zostaje wrzucona na stos. Cykl powtarza się. Za każdym razem gdy funkcja *dfs* zakończy swoje działanie, *wrapper* sprawdza czy warunki końcowe są spełnione. Jeżeli tak, to kończy swoje działanie.

3. ALGORYTM BFS (BREADTH-FIRST SEARCH)

2.1 Zdefiniowanie problemu

Celem zadania jest umożliwienie naszemu agentowi poruszania się, używając algorytmu przeszukiwania wszerz. Strategią tego algorytmu jest wyznaczenie wszystkich wierzchołków sąsiadujących z wierzchołkiem startowym, a następnie ich sprawdzanie. Cykl ten powtarzamy, aż do wyznaczenia wszystkich wierzchołków. Ważne jest, aby zapamiętać czy dany wierzchołek był już wcześniej odwiedzony, żeby uniknąć zapętlenia.

2.2

W celu uzyskania wyników, używamy *wrappera*, który uruchamia funkcję *bfs* odpowiedzialną za wyszukanie ścieżki. Funkcja *bfs* posiada w sobie kolejkę, w której zapisane ma współrzędne przebytych punktów. Na pierwszego elementu na liście, sprawdzane są warunki możliwego przejścia do innych punktów. Jeżeli można iść dalej, to współrzędna zostaje dołączona do kolejki. Cykl powtarza się do momentu sprawdzenia wszystkich możliwych punktów. Gdy funkcja *dfs* kończy swoje działanie, *wrapper* sprawdza czy warunki końcowe są spełnione. Jeśli tak jest, to kończy swoje działanie.