**Raport 12.06.2019**

**Planowanie ruchu oraz prezentacja końcowa**

**Grupa 11**

**Planowanie ruchu – algorytm DFS (w strategii niepoinformowanej).**

Algorytm został zaimplementowany przy użyciu rekurencji.

**dfs\_move**(grid, position, visited\_houses, counter, solution, count, temp)

* grid – przekazuje siatkę aby śmieciarka była w stanie sprawdzić 4 pola wokół siebie
* position – aktualna pozycja śmieciarki
* visited\_houses – lista odwiedzonych domów
* counter – aktualna ilość ruchów śmieciarki (max 130)
* solution – lista ruchów
* count – liczba domów na mapie
* temp – poprzedni ruch

metoda „dfs\_move” na początku sprawdza czy nie osiągnęła już maksymalnej długości, jeśli tak kończy działanie danej rekurencji. Działanie kończy się również, gdy śmieciarka znalazła już wszystkie domy, wtedy lista „solution” jest dopisywana do listy list „solutions”. Następnie w przypadku, gdy nie zakończyła działania dopisuje poprzedni ruch „temp” do „solution”. W tym momencie następuje sprawdzenie czy wokół śmieciarki nie znajduje się jakiś nieodwiedzony dom, jeśli tak śmieciarka zabiera śmieci, dekrementuje zmienną „count”, dodaje dom do „visited\_houses” oraz wywołuje „dfs\_move”. Po sprawdzeniu domów śmieciarka szuka dróg pomijając ostatnio wykonany ruch. Po znalezieniu takiej drogi zmienna „temp” przyjmuje jaki ruch należy wykonać oraz wywołany zostaje „dfs\_move” ze zmienioną pozycją odpowiadającą wykonanemu ruchowi.

Po wywołaniu metody „dfs\_move” zostaje wybrana najkrótsza lista z listy list „solutions”, czyli najoptymalniejsze przejście jakie znalazł algorytm DFS.

**Planowanie ruchu – algorytm BFS (w strategii niepoinformowanej).**

Algorytm został zaimplementowany przy pomocy iteracji.

**bfs\_move**(grid, position, count)

* grid – przekazuje siatkę aby śmieciarka była w stanie sprawdzić 4 pola wokół siebie
* position – aktualna pozycja śmieciarki
* count – liczba domów na mapie

Metoda „bfs\_move” działa w pętli for, która odpowiada za ilość obecnie rozpatrywanych „zagłębień”. Na początku sprawdza, czy znajdują się wokół niej nieodwiedzone domy, jeśli tak to je odwiedza, zabiera śmieci, dekrementuje zmienną „count” i dodaje dom do „visited\_houses”. W przypadku, gdy zostały odwiedzone wszystkie domy metoda zwraca „solution” – listę kroków i kończy działanie. Jeśli program nie zakończył działania to sprawdza ile ma dostępnych dróg wokół siebie (nie będących poprzednim ruchem). Przy więcej niż jednej opcji metoda rozpoczyna szukanie większej ilości „zagłębień”. Do każdego z nich dodaje odpowiednie ruchy oraz zmienia pozycję.

Ważną metodą dla strategii niepoinformowanej jest „find\_houses(grid)”**.** Służy ona do zwrócenia ilości domów na wybranej mapie.

**Machina Learning**

Jest to dziedzina zajmująca się problematyka SI. Polega na stworzeniu automatycznego systemu, który sam będzie się doskonalił na podstawie zgromadzonych danych. Algorytmy oparte o ten sposób rozwiazywania problemów, wymagają od programisty podania zestawu danych, na podstawie których algorytm jak i cały program ma wyciągnąć wnioski i „nauczyć się” jak zachowywać się w takiej sytuacji. ML sprawuje się idealnie w sytuacji, kiedy nie możemy określić jasno reguł, które będą dobrze działały i problem jest za duży do rozwiązania go ręcznie. Jednakże w celu dobrego działania ML niezbędny jest odpowiedni zestaw pod względem ilości jak i ilości danych.

**Drzewa decyzyjne**

Drzewo decyzyjne w ogólności to struktura drzewiasta stworzona przez algorytm która jest utworzona na podstawie danych treningowych. Na jej podstawie śmieciarka wybiera najlepsze zachowanie w danej chwili w jakiej się znajduje.

**get\_data\_tree\_from\_file()**

Metoda odpowiada za pobranie wszystkich plików z pliku **dataset.txt**, zwraca ona nam wszystkie dane podzielona na dwa zbiory **choices\_train** który zawiera dane treningowe oraz **choices\_test** który zawiera dane testowe.

**train\_decision\_tree(choices\_train, possibilities\_train)**

Metoda odpowiada za nauczenie naszego drzewa, jako parametr przyjmuje zestaw danych treningowych, w wyniku ta metoda zwraca obiekt DecisionTreeClassifier.

**get\_tree\_decision(clf, possibilites\_test, choices\_test)**

Metoda zwraca ruch jaki podjeło drzewo w obecnej sytuacji.

**ai\_move(grid, position, clf)**

Metoda odpowiada za ruch śmieciarką na podstawie decyzji jaką zwróciła wybrana z metoda uczenia maszynowego (regresja lub drzewa decyzyjne).

**create\_dataset(grid, soultion, position)**

Metoda tworzy dataset który jest wykorzystywany przez drzewa decyzyjne i regresje logistyczną w postaci:

Wartość oczekiwana, otoczenie śmieciarki w obszarze 5x5 (oddzielone przecinkami)

Przykład: 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 5, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

**Legenda:**

0 – trawa  
1 – droga  
2 - dom  
3 – wysypisko śmieci  
4 – odwiedzony dom  
5 – odwiedzona droga

**Regresja logistyczna**

Model regresji logistycznej jest szczególnym przypadkiem uogólnionego modelu liniowego. Znajduje zastosowanie, gdy zmienna zależna jest dychotomiczna, to znaczy przyjmuje tylko dwie wartości takie jak na przykład sukces lub porażka,

**train\_logistic\_regression(X\_train, Y\_train)**

Jako parametry do metody zostają podane dane treningowe dla współrzędnej X i Y, w wyniku działania metody zostaje zwrócony obiekt Liniowej Regresji.

**get\_linear\_regresion\_decision(regr, possible\_choices)**

Metoda jako parametry przyjmuje obiekt, który został utworzony przez wyżej wymienioną metodę, który odpowiada za regresje liniowa i zestaw możliwych decyzji. W wyniku działania metody zostaje zwrócony ruch, który został uznany za najoptymalniejszy.

**Vowpal Wabbit**

**create\_rabbit\_model()**

Metoda tworząca model jednorazowo, za każdym razem, gdy dataset uległ zmianie.

**move\_rabbit(grid, position, count)**

Metoda odpowiada za ruch śmieciarką na podstawie decyzji jaką Vowpal Wabbit uznał za za najoptymalniejsza.

**Create\_dataset\_for\_rabbit(grid, solution, position)**

Metoda tworząca dataset dla Vowpal Wabbita w postaci:

Wartość oczekiwana | opisane otoczenie na kwadracie 5 x 5 wokół śmieciarki

Przykład: 8 | 1x1:.0 1x2:.0 1x3:.0 1x4:.0 1x5:.0 2x1:.0 2x2:.0 2x3:.0 2x4:.0 2x5:.0 3x1:.0 3x2:.0 3x4:.0 3x5:.0 4x1:.0 4x2:.0 4x3:.20 4x4:.0 4x5:.0 5x1:.0 5x2:.0 5x3:.20 5x4:.0 5x5:.0

**Legenda:**

axb – współrzędne pola (z kwadratu 5x5)  
0 - trawa  
20 – droga po której można iść  
35 - dom   
2 – droga odwiedzona  
4 – odwiedzony dom  
1 – wysypisko śmieci