

Wirtualne zespoły robocze - instrukcja do ćwiczenia: „agenty autonomiczne”

Celem ćwiczenia jest poznanie zagadnień związanych z implementacją agentów autonomicznych, tzn. sterowanych za pomocą algorytmów w imieniu użytkownika, takich jak:

- algorytmy sterowania,
- automatyczne negocjacje,
- poszukiwanie optymalnych strategii negocjacyjnych,
- rozwiązania Pareto-optymalne,
- zastosowanie metod uczenia.

Algorytmy sterowania

W najprostszym przypadku autonomiczny obiekt jest wyposażony w stały, niemodyfikowalny algorytm sterowania. W przypadku systemów bardziej zaawansowanych możliwe jest również dodawanie reguł a nawet uczenie agenta, czyli modyfikacja algorytmu w trakcie działania, na podstawie doświadczeń z interakcji agenta ze środowiskiem, a w tym i innymi autonomicznymi agentami. Wybór typu algorytmu jest najczęściej uzależniony od dostępności wiedzy o sterowaniu, a w szczególności od charakteru wiedzy (wiedza sformalizowana – np. zbiór reguł, niesformalizowana – np. zbiór przykładów sterowania, luźne heurystyki), jak również od tego czy agent ma się uczyć i w jakim zakresie. Przykładowe algorytmy wraz z wymaganiem dostępności wiedzy przedstawione są w poniższej tabeli:

algorytmy	dostępność wiedzy	uczenie
algorytm stały (niemodyfikowalny w trakcie działania systemu)	wiedza heurystyczna	brak
system ekspertowy – system regułowy lub regułowy system rozmyty	wiedza heurystyczna	brak
drzewa decyzyjne	zbiór przykładów sterowania	jest
sztuczna sieć neuronowa	zbiór przykładów sterowania	jest
genetyczny system uczący się	tylko ocena jakości działania	jest
uczenie ze wzmocnieniem	tylko ocena jakości działania	jest
rozmyty system uczący się (np. ANFIS)	wiedza heurystyczna oraz zbiór przykładów sterowania	jest

Algorytmy powyższe mogą być ze sobą łączone na różne sposoby np. równoległe i szeregowo (hierarchiczne). Łączenie równoległe pozwala na zastosowanie algorytmu odpowiedniego do sytuacji lub zadania do wykonania. Przykładem zastosowania łączenia szeregowego jest system ogólnych reguł typu:

jeśli mało paliwa to uzupełnij paliwo,

jeśli masz uzupełnić paliwo to skieruj się ku beczce z paliwem lub ogłoś chęć kupna

oraz algorytmów niższego poziomu służących np. do obliczania kąta skrętu kół (lub serwomechanizmu pozwalającego na dotarcie do celu) czy realizacji transakcji kupna, ustalania korzystnej ceny itp. Strategia każdego poziomu może podlegać uczeniu pod warunkiem istnienia gotowych strategii niższego poziomu.

Automatyczne negocjacje

Założmy, że uczestnicy mają możliwość zbierania przedmiotów (pieniądze oraz beczki z paliwem), oraz wymiany ich pomiędzy sobą. Wymiana taka jest możliwa, gdy uczestnicy znajdują się w bliskiej odległości od siebie i mogą dokonywać przelewów pieniężnych oraz przekazania towaru (za pomocą specjalnych ramek). Weźmy pod uwagę sytuację, gdy jednemu z uczestników zaczyna brakować paliwa, gdy znajduje się w dużej odległości od najbliższej beczki. W takiej sytuacji racjonalną decyzją jest kupno paliwa (np. 10 kg beczki) od innego uczestnika z dostawą (beczka jest dowożona do kupującego) lub bez dostawy. Wysokość ceny może być uzależniona od ilości paliwa posiadanego przez kupca, ilości paliwa w posiadaniu sprzedawcy oraz od ilości gotówki. Każdy z agentów może posiadać stałą lub adaptacyjną (podlegającą uczeniu) strategię negocjacji. Może to być strategia mieszana, tzn. wybierająca decyzje z pewnym prawdopodobieństwem. Jednym ze sposobów reprezentowania takiej strategii jest zbiór reguł. Przykładowy zbiór reguł:

jeżeli ilość paliwa $< P1$ **to** należy paliwo kupić z dostawą za cenę $< C1$
jeżeli ilość paliwa $< P2$ **to** należy paliwo kupić za cenę $< C2$
jeżeli ilość paliwa $> P3$ **to** można paliwo sprzedać za cenę $> C3$
jeżeli decyzja kupna paliwa **to** należy wybrać potencjalnego sprzedawcę, który znajduje się blisko i ma dużo paliwa **i** następnie zaproponować mu cenę $C4$
jeżeli cena sprzedawcy $C5 >$ cena akceptowalna **to** zaproponuj nową cenę $C6 = C4 + dC$, gdzie dC zależy od ilości posiadanego paliwa i ilości gotówki
jeżeli cena sprzedawcy akceptowalna **to** przystąp do transakcji
jeżeli po N krokach negocjacji cena sprzedawcy nieakceptowalna **to** przerwij negocjacje z tym sprzedawcą i poszukaj innego

...

W przypadku stałej strategii negocjacyjnej agentów, dany agent może poszukać strategii w stosunku do niej optymalnej metodą uczenia.

Przy dodatkowo zróżnicowanych umiejętnościach (opcja `if_different_skills` w pliku `main.cpp`), korzystne może okazać się zawiązanie stałych koalicji (drużyn) z podziałem na role: np. zbieracze monet, dostawcy paliwa.

Zadanie podstawowe - algorytmy sterowania:

Opracowanie algorytmu sterowania autonomicznym agentem pozwalającego na maksymalizację zysku (ilości gotówki) w ciągu 10 minut. Można to osiągnąć poprzez dobór odpowiedniego kąta skrętu kół `my_vehicle->state.wheel_turn_angle` w zakresie (-45 stopni, 45 stopni), siły pchającej pojazd do przodu `my_vehicle->F` w zakresie $(-F_{max}, F_{max})$ oraz stopnia hamowania `my_vehicle->breaking_degree` w zakresie (0,1). W przypadku skrętów, można użyć zmiennych `my_vehicle->wheel_turn_speed` i

my_vehicle->*if_keep_steer_wheel*. Kod algorytmu należy umieścić w funkcji *AutoPilot::AutoControl()* w module *agents*. Do włączenia lub wyłączenia automatycznego sterowania służy klawisz Y. Do włączania testu automatycznego sterowania służy klawisz F2.

Informacje o stanie środowiska dostępne dla agenta:

- wysokość terenu w punkcie (x,z) można uzyskać za pomocą funkcji *terrain*->*GroundHeight(x,z)*
- informacje o *i*-tym przedmiocie *terrain.p[i]*:
 - położenie *vPos*
 - typ przedmiotu: *typ* ze zbioru $\{ITEM_COIN, ITEM_BARREL, ITEM_TREE\}$
 - ilość paliwa lub nominał w polu *value*
 - czy przedmiot można wziąć – jeśli pole *to_take* == 1
- liczba przedmiotów: *terrain.number_of_items*
- przedmioty z ograniczonego obszaru można uzyskać stosując funkcję *terrain*->*ItemsInRadius(...)*
- informacje o stanie obcych obiektów, które mogą uczestniczyć w rywalizacji lub współpracować w tablicy *network_vehicles[]*

Wektory lokalnego układu współrzędnych obiektu można obliczyć na podstawie jego orientacji:

```
Vector3 vect_local_forward = state.qOrient.rotate_vector(Vector3(1, 0, 0));  
Vector3 vect_local_up = state.qOrient.rotate_vector(Vector3(0, 1, 0));  
Vector3 vect_local_right = state.qOrient.rotate_vector(Vector3(0, 0, 1));
```

Uwaga! W zadaniu podstawowym, na początku pliku *main.cpp* należy ustawić zróżnicowanie umiejętności na *false*:

```
bool if_different_skills = false;
```