Zastosowanie Qlearningu w zachowaniu boidów

Jakub Łęcki, Marek Hering, Maciej Jabłoński

08.06.2020

Streszczenie

Tematem niniejszej pracy jest problem nauczenia boidów (w naszym przypadku są to ryby), aby poprzez prawidłowe poruszanie się maksymalizowały swój czas życia. W tym celu użyliśmy koncepcji qlearningu oraz algorytmu stada.

1 Boidy

1.1 Pochodzenie

Termin **boid** został stworzony przez Craiga Reynoldsa w 1987 roku jako określenie stworzenia wykazującego cechy stadne. Słowo boid wzięło się z uproszczenia terminu 'bird-like' jako odniesienie do ptaków formujących się w gromady.

1.2 Zasady zachowania

Okazuje się, że w świecie rzeczywistym wiele gatunków zwierząt łączących się w grupy wykazuje podobne własności. Patrząc na stada ptaków, ławice ryb, roje pszczół lub stada owiec można zauważyć, że każda z jednostek stosuje się do 3 podstawowych zasad:

1. Rozdzielność - osobnik nie lubi przebywać w tłoku, dlatego zachowuje dystans do swoich sąsiadów
2. Spójność - osobnik nie lubi przebywać w samotności, więc kieruje się ku najbliższym współstadnikom
3. Wyrównanie - osobnik porusza się w kierunku zbliżonym do kierunku otaczających członków stada

Łącząc te 3 proste zasady, boidy tworzą złożone i bardzo zorganizowane skupiska, które obserwujemy jako np. ławice ryb, które pozostają w płynnym, nieustannym ruchu.

2 Qlearning

Jest to jedna z technik szerokiej dziedziny uczenia maszynowego znanej jako "Uczenie ze wzmocnieniem" (ang. Reinforcement Learning). Opiera się na śledzeniu zachowania agentów oraz efektów, które owe akcje powodują. W tym celu używana jest tablica stanów-akcji zwykle nazywana jako **qtable**.

2.1 Qtable

Tablica ma wymiary m $\times$ n, gdzie:

- m - liczba możliwych stanów
- n - liczba akcji możliwych do wykonania

W każdej komórce $Q(s, a)$ znajduje się oczekiwana wartość nagrody jaką agent otrzyma będąc w stanie s i wykonawszy akcję a. Po wykonaniu akcji, agent przechodzi do kolejnego stanu s'. Agent będąc w stanie s będzie wybierać swoją kolejną akcję na podstawie polityki $argmax(Q(s))$, czyli wybranie akcji za którą teoretycznie otrzyma największą nagrodę.

2.2 Środowisko

Jest to zbiór agentów oraz dowolnych innych encji z którymi agent może w jakiś sposób oddziaływywać. Rolą środowiska jest wykonanie akcji wybranej przez agenta i ocenienie jak dobrze akcja została wybrana. W tym celu środowisko nadaje agentowi nagrody (i kary, jeśli nagroda jest ujemna). W kolejnych rundach skutkuje to stopniowym poprawianiem procesu wyboru akcji i agent zbiera coraz wyższe nagrody. Po wykonaniu kroku środowisko przekazuje do algorytmu uczenia zestaw danych:

- stan w którym był agent
- akcja jaką wykonał
- stan w którym znajduje się po wykonaniu akcji
- nagroda jaką otrzymał za przejście do kolejnego stanu

2.3 Proces uczenia

Aby agent wybierał z czasem coraz lepsze decyzje, wartości w Qtable muszą ulegać zmianie. Odbywa się to w oparciu o poniższe równanie:

$$Q'(s,a) \leftarrow Q(s,a) + \alpha \cdot \Big(r + \gamma \cdot \max_{\text{a}} Q(s',a) - Q(s,a)\Big) \tag{1}$$

Gdzie s i a to stan i akcja przed jej wykonaniem, a s' to stan po wykonaniu akcji.

2.3.1 Współczynnik uczenia α

Wartość $\alpha \in< 0,1 >$ reguluje jak bardzo znacząca jest nowa informacja uzyskana w wyniku wykonania akcji w środowisku. Przy wartości 0 agent nie będzie się uczył niczego nowego, natomiast przy $\alpha = 1$ agent zaakceptuje całą zdobytą wiedzę.

2.3.2 Współczynnik dyskontowania γ

Wartość $\gamma \in< 0,1 >$ określa ważność przyszłych nagród zdobywanych przez agenta. Wartość dążąca do zera zwiększy sugerowanie się pamięcią krótkotrwałą, natomiast do 1 pamięcią długotrwałą. Ważnym elementem jest, aby γ rzeczywiście zawierała się w przedziale $< 0,1)$ ponieważ zapewnia to zbieżność wartości przewidywanej nagrody. Jeśli proces uczenia byłby nieskończony i $\gamma \geqslant 1$ (a nawet lekko poniżej) wartości nagród rosłyby nieustannie zaburzając proces.

2.3.3 Efekt

W wyniku przeprowadzenia odpowiedniej ilości kroków wartości nagród zbiegają się do optymalnych, a w tabeli powstają zależności pomiędzy poszczególnymi stanami umożliwiające dotrzeć do nawiększej nagrody, dysponując jedynie obecnym stanem i przewidywaną nagrodą.

2.3.4 Polityka wspomagająca

Gdy podczas uczenia agent będzie słuchał się tylko tabeli qlearningu, może się zdarzyć że zadowoli się częściowo poprawnym rozwiązaniem, zamiast szukać rozwiązania dokładnego. Aby uniknąc takiej sytuacji proces wybierania akcji rozszerza się często o politykę **epsilon greedy**, w której z pewnym prawdopodobieństwem wybierana jest losowa akcja zamiast wskazywanej przez tabelę. Wartość ϵ jest zmniejszana z czasem, aby agent coraz częściej stosował to czego się nauczył.

3 Kontrola ruchu boidów

3.1 Ruch ciągły

Boidy muszą być osadzone w przestrzeni euklidesowej, zatem każdy z boidów posiada trzy dwuwymiarowe wektory położenia, prędkości i przyspieszenia.
Aby agent mógł poruszać się ruchem gładkim, postanowiliśmy kontrolować boidy za pomocą przyspieszenia, a na prędkość nałożyć ograniczenia w postaci minimalnej i maksymalnej wartości.

3.2 Stan agenta

Stanem agenta jest jego pozycja oraz kąt nachylenia prędkości do osi X.

3.3 Dyskretyzacja

Qlearning z definicji działa na wartościach dyskretnych, ponieważ wartości nagród wpisujemy pod konkretne wiersze i kolumny tabeli. Zatem konieczne było zdyskretyzowanie stanu agenta. Wykonaliśmy to poprzez przeniesienie pozycji agenta w przestrzeń o dużo mniejszej rozdzielczości r, a kąt spłaszczamy do N równo oddalonych wartości.

3.4 Akcja agenta

W klasycznym algorytmie qlearningu na wyjściu otrzymujemy N wartości nagród, spośród których indeks maksymalnej wartości symbolizuje akcję do wykonania. W naszym problemie jednak sprowadzenie ruchu do zaledwie kilku kierunków nie spełniłoby założenia o gładkości, dlatego przyjęliśmy nową politykę. Zamiast wyznaczać przyśpieszenie na podstawie jednej akcji, wyznaczamy je według wzoru:

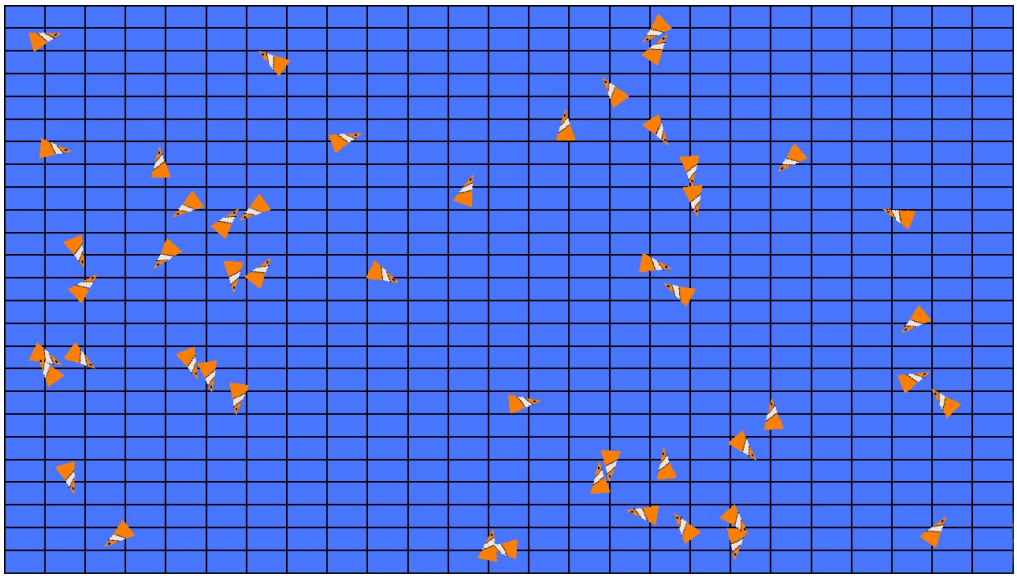
$$a = \sum_{i=0}^{N} a_i \tag{2}$$

gdzie i to indeks akcji, N to ilość możliwych kierunków, natomiast a_i to wektor w postaci biegunowej:

$$a_i = [r_i, i \cdot 360/N] \tag{3}$$

gdzie r_i to wartość nagrody otrzymanej za i-tą akcję.
Użycie wartości nagrody poskutkuje poprawną reakcją boida ponieważ w miejscach których ma unikać nagroda będzie ujemna, a w możliwych do przemieszczenia - dodatnia.

4 Utrzymanie agentów na planszy



Rysunek 1: Stan początkowy planszy z agentami. Siatka symbolizuje komórki qtable zależne od położenia

Na początku zajęliśmy się nauczeniem boidów, aby nie dotykały krawędzi. W tym celu przyjęliśmy rozdzielczość tablicy stanów 25×25 oraz 12 kierunków w których możliwe jest obliczenie przyśpieszenia.
Dodatkowo:

1. ilość ryb = 50
2. długość epoki = 2000 klatek ($\approx$ 1 min)

4.1 Nagradzanie

Boid za dotknięcie granicy otrzymuje nagrodę o wysokości -1000.

4.2 Wynik

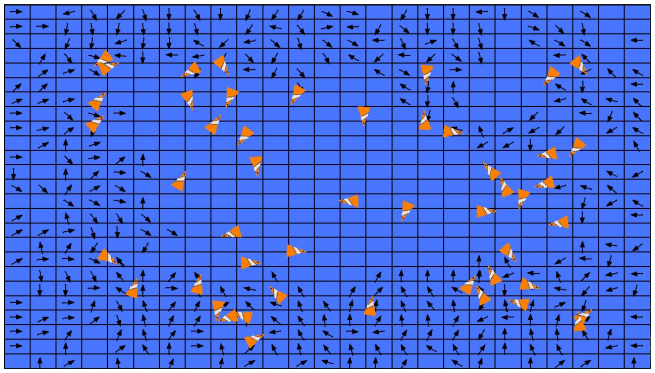
Boidy dość skutecznie utrzymywały się przy życiu, jednak ich ruch bardziej przypominał odbijanie się niż omijanie krawędzi. Wynika to z faktu, że nagroda wyznaczana jest tylko w tej komórce w której agent zginie, nie ma żadnej relacji pomiędzy sąsiednimi stanami, więc reakcja następowała tylko tuż przy granicy.

4.3 Modyfikacja polityki aktualizacji tablicy stanów

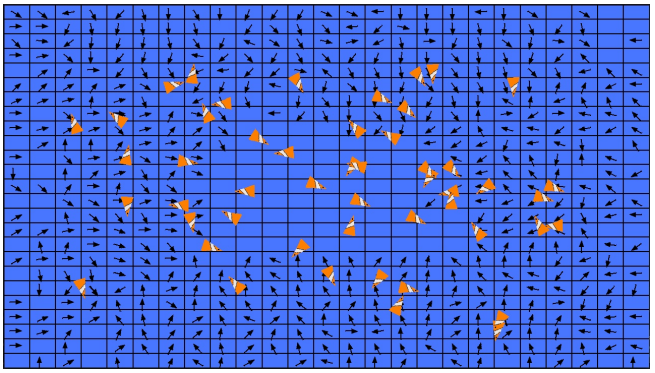
W tradycyjnym algorytmie qlearningu, równanie (1) Bellmana aktualizuje stan przez wybranie maksymalnej nagrody możliwej do uzyskania. W naszym przypadku nie jest to skuteczne, ponieważ wykonanie akcji nie skutkuje natychmiastowym uzyskaniem nagrody. Aby umożliwić agentowi ocenienie czy warto ruszyć się w danym kierunku zrezygnowaliśmy z wybierania wartości maksymalnej. Zamiast tego bierzemy wszystkie możliwe nagrody do uzyskania po wyjściu z obecnego stanu i liczymy średnią arytmetyczną z minimalnej i maksymalnej nagrody. Dzięki temu w tabeli stanów powstaje swego rodzaju dyskretne pole siłowe, które nadaje boidom kierunek w którym powinny się udać.

4.4 Stan tabeli z nową polityką nagród

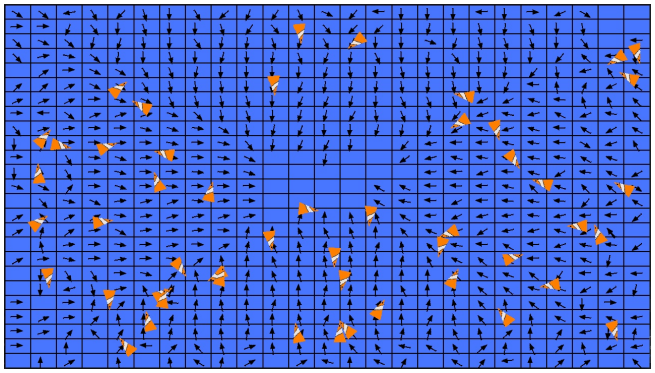
Po powyższej zmianie wyniki mają się następująco. Strzałki oznaczają kierunek wypadkowego przyśpieszenia.



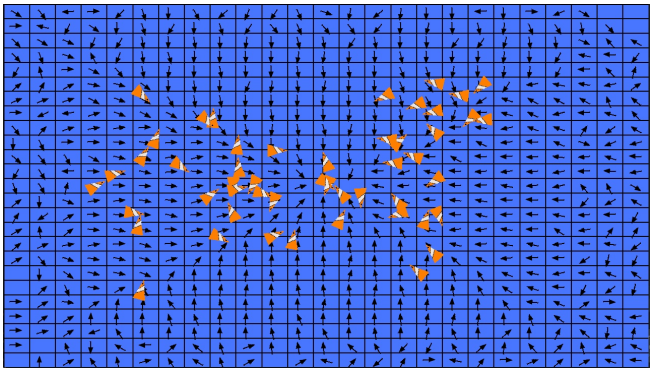
Rysunek 2: Stan tabeli po dwóch epokach



Rysunek 3: Stan tabeli po 3 epokach

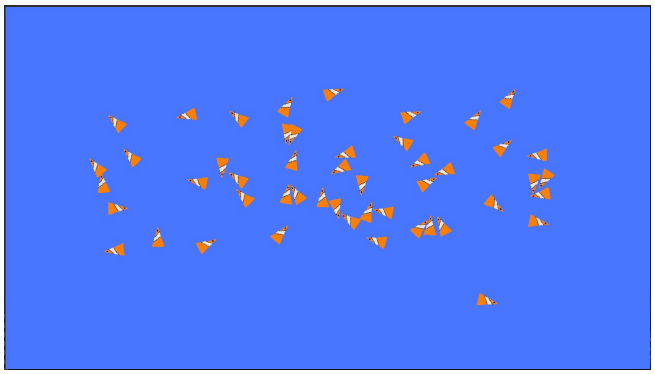


Rysunek 4: Stan tabeli po 5 epokach

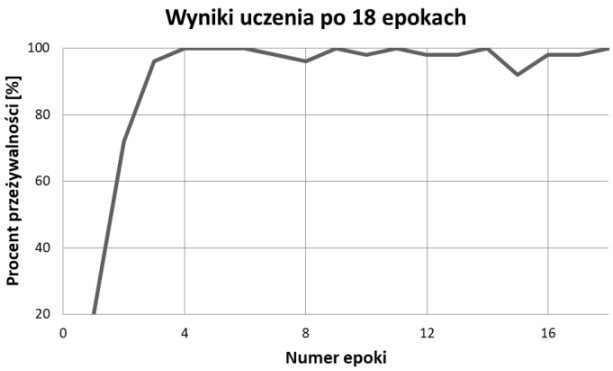


Rysunek 5: Stan tabeli po 10 epokach

Jak widać już po 10 epokach agenci nauczyli się skutecznie unikać krawędzi. Patrząc na kierunki poszczególnych strzałek widzimy, że w tabeli powstało pole, które zbiega się do środka tworząc optymalne ścieżki nawet w nietrywialnych miejscach takich jak rogi planszy. Jest to bardzo korzystne, ponieważ takie miejsca sprawiają zawsze problem w alogorytmicznym wykrywaniu kolizji.



Rysunek 6: Ryby utrzymujące się na środku planszy po nauczeniu



Rysunek 7: Wykres przeżywalności ryb w czasie uczenia

5 Polowanie i uciekanie

W kolejnym etapie postanowiliśmy dodać do środowiska boidy drugiego rodzaju, tzn. drapieżniki. Ich zadaniem jest manipulowanie swoim ruchem, aby zjadać ryby. W momencie kolizji drapieżnika z rybą następuje śmierć ryby. Drapieżniki tak samo jak ryby umierają na krawędziach planszy.

5.1 Qtable

Aby umożliwić nauczenie się zarówno uciekania jak i gonienia rozszerzyliśmy tablicę stanów o 2 wymiary. Jeden który indeksuje rodzaj uczącego się boida, drugi natomiast zawiera stany jako dyskretny kąt pod którym agent widzi swój cel. Celem odpowiednio jest najbliższa ryba dla drapieżnika i najbliższy drapieżnik dla ryby.

5.2 Pole reakcji

Odnajdywanie najbliższego celu dla każdego boida na planszy ma istotne wady.

1. Jest wysoce nieoptymalne, ponieważ wymaga przeszukania każdy z każdym więc ma czas $O(n^2)$
2. Sugeruje, że boid posiada wiedzę *a priori* o położeniu dowolnego innego boida, co nie pokrywa się z rzeczywistością, gdzie każdy organizm ma ograniczone pole widzenia i reakcji.