AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

Sztuczna inteligencja na przykładzie symulacji komputerowej
Artificial intelligence in computer simulation

Autor: *Bartłomiej Konieczny*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Mirosław Gajer*

Kraków, 2015

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Uprzejmie dziękuję doktorowi Mirosławowi Gajerowi za możliwość podjęcia interesującej tematyki pracy i za wskazówki przy jej powstawaniu.

Spis treści

1. Wstęp 7

1.1. Cele pracy 7

1.2. Zawartość pracy 8

2. Uczenie maszynowe - rozdział teoretyczny 9

2.1. Podejścia do uczenia maszynowego 9

2.1.1. Uczenie nadzorowane 9

2.1.2. Uczenie nienadzorowane 10

2.1.3. Uczenie ze wzmocnieniem 12

2.2. Podsumowanie 14

3. Implementacja 15

3.1. Struktura stan-akcje $Q(S, A)$ 15

3.2. Polityka wyboru akcji 16

3.2.1. Równowaga pomiędzy eksploracją, a eksploatacją 16

3.2.2. ϵ-greedy policy 17

3.2.3. Optimal policy 17

3.3. Symulacja graficzna 17

3.4. Reprezentacja stanu 18

3.4.1. Akcje 20

3.4.2. Wartość nagród 21

3.4.3. Wybór algorytmów 21

3.5. Wykorzystane technologie 24

3.6. Napotkane problemy 24

3.7. Obsługa symulacji 24

3.8. Wynik działania 25

4. Podsumowanie 27

1. Wstęp

W ostatniej dekadzie można zauważyć zwiększone zainteresowanie rozwiązaniami z dziedziny sztucznej inteligencji. Innowacyjne pomysły z użyciem tych algorytmów pozwalają nie tylko na interpretacje ogromnych ilości danych, których człowiek nie jest w stanie przetworzyć ale również, między innymi, na rozwój autonomicznych pojazdów, jeżdżących bez nadzoru kierowcy.
Według [1] sztuczną inteligencją możemy nazwać "badanie i rozwój inteligentnych maszyn, w szczególności programów komputerowych".
Inteligentne zachowanie agenta możemy zdefiniować, gdy agent[2]

- dostosowuje swoje zachowanie do aktualnych warunków i celów,
- ma zdolność zmiany otoczenia i celów,
- uczy się z doświadczenia,
- wykonuje odpowiednie do swoich ograniczeń akcję.

Wykorzystując powyższe definicję, sztuczną inteligencje określamy jako dziedzinę naukową zajmującą się badaniem, rozwojem i implementacją programów i maszyn wykazujących cechy inteligencji, tzn. takie które ucząc się z doświadczenia, będąc zmiennym w stosunku do otaczającego ich otoczenia i warunków dążą do wykonania swoich celów, uwzględniając obowiązujące je ograniczenia.

1.1. Cele pracy

Celem pracy jest opis i implementacja inteligentnego agenta. Agent wykorzystując algorytmy uczenia ze wzmocnieniem wyciąga wnioski z podejmowanych akcji i dostosowuje swoje zachowanie. Aby lepiej zaprezentować wyniki jego działania, stworzone zostanie proste środowisko graficzne, które będzie przedstawiać wyniki przeprowadzanych przez robota wyborów.
W pracy zostanie wyjaśnione pojęcie uczenia maszynowego i podstawowe podejścia do rozwiązania problemów z tej dziedziny. Dzięki temu możliwe będzie podsumowanie różnic między uczeniem ze wzmocnieniem, a pozostałymi podejściami do uczenia maszynowego.

1.2. Zawartość pracy

Pierwszy rozdział stanowi wprowadzenie do pracy, określający jej zakres. W rozdziale drugim przedstawiono wyjaśnienie podstawowych pojęć i podstaw teoretycznych. W rozdziale trzecim zostały wyjaśnione szczegóły implementacji inteligentnego agenta oraz opis wybranego algorytmu. Pracę zamyka rozdział podsumowujący jej treść.

2. Uczenie maszynowe - rozdział teoretyczny

Jednym z najistotniejszych zagadnień z dziedziny sztucznej inteligencji jest uczenie maszynowe. Uczenie maszynowe jest metodą analizy danych, która automatyzuje budowę modelu analitycznego na podstawie nauki z danych. W wielu zastosowaniach użycie metod uczenia maszynowego jest znacznie bardziej efektywne od manualnego programowania, w wyniku czego uczenie maszynowe znalazło szerokie zastosowanie w informatyce i innych dziedzinach. W ostatniej dekadzie można zauważyć zwiększone użycie metod uczenia maszynowego[3].

2.1. Podejścia do uczenia maszynowego

– uczenie nadzorowane,

– uczenie nienadzorowane,

– uczenie ze wzmocnieniem,

2.1.1. Uczenie nadzorowane

Uczenie nadzorowane polega na wnioskowaniu funkcji z określonych danych trenujących. Wykorzystując dostarczone przykłady algorytmy potrafią estymować wartości danych, które mogą nie występować w podanym zbiorze wejściowym. Dzięki generalizowaniu z przykładów, metody uczenia nadzorowanego są w stanie wyznaczać przewidywane wartości na podstawie danych trenujących.

Ważną cechą danych trenujących w uczeniu nadzorowanym jest konieczność ich oznaczenia. Algorytm, aby móc szacować pożądane wartości funkcji, musi posiadać wiedzę o ich cechach.

Przykładem zastosowania algorytmów uczenia nadzorowanego jest system rozpoznawania niechcianych wiadomości w klientach pocztowych. Danymi wejściowymi są w tym przypadku kategoryzowane na pożądane lub niepożądane wiadomości e-mail. System generalizując podane mu przykłady jest w stanie zidentyfikować kolejne wiadomości i wykonać odpowiednią akcję, zależnie od preferencji użytkownika (może to być na przykład usunięcie lub przeniesienie do zdefiniowanego folderu).

Wiele różnych algorytmów uczenia nadzorowanego zostało wykorzystanych by rozwiązać problem klasyfikacji wiadomości e-mail. Użyto między innymi algorytmów k-nearest neighbor[4], Naive Bayes[5][6] czy Random Forest[7], jednak wiąże się to z istotnymi wadami[8]:

– **Wymagane oznaczenie danych testowych**. Metody uczenia nadzorowanego wymagają, aby dane trenujące były oznaczone. W przypadku klasyfikacji wiadomości e-mail, koniecznie jest ich oznaczenie w zależności od tego czy są szkodliwe czy nie. Problem stwarza tutaj wielkość danych. Ilość wiadomości, która jest wymieniana w sieci jest bardzo duża. W związku z czym, żeby klasyfikacja miała sens, wymagane też jest oznaczenie sporej ilości przykładów, co nie zawsze jest możliwe i opłacalne do zrealizowania.

– **Mała liczba danych testowych**. W związku z niewielką (w stosunku do wszystkich możliwych) ilością danych trenujących, algorytm jest mało odporny na modyfikowane dane. Osoby rozsyłające niechciane wiadomości bardzo często będą zmieniać ich treść i strukturę, na taką, która nigdy nie pojawiła się wśród danych trenujących. Może mieć to negatywny wpływ na wynik działania algorytmu.

2.1.2. Uczenie nienadzorowane

Podobnie jak w uczeniu nadzorowanym, algorytmy uczenia nienadzorowanego wyznaczają funkcje na podstawie danych wejściowych, jednak są w stanie odkryć niewidoczne zależności między nimi. Konsekwencją wynikającą z charakterystyki algorytmów uczenia nadzorowanego jest niemożność określenia błędu lub poprawności rozwiązania. Celem działania algorytmu może być na przykład kategoryzowanie informacji (klasteryzacja).

W odróżnieniu od uczenia nadzorowanego, metody uczenia nienadzorowanego są w stanie wykryć ukryty wzorzec w danych wejściowych.

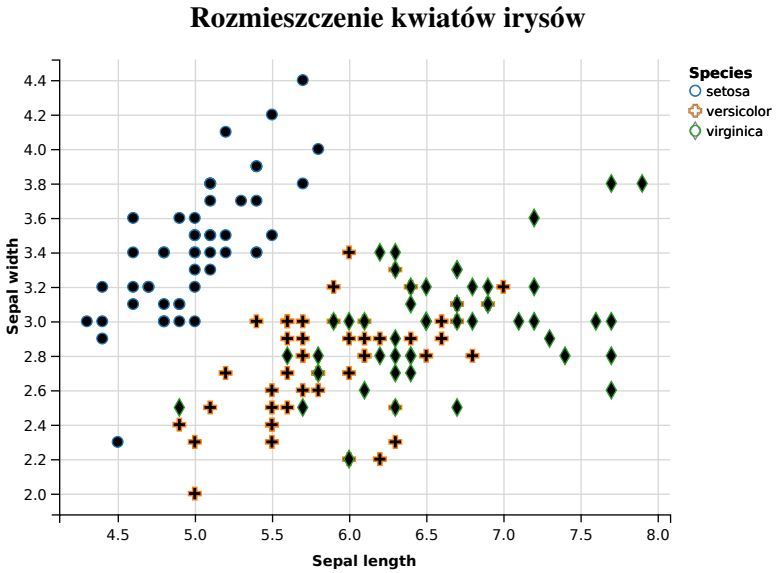
Używając jako wejścia informacji dotyczących kwiatów irysów w postaci przedstawionej na listingu 2.1, algorytmy uczenia nienadzorowanego mogą być zastosowane w celu wywnioskowania gatunku kwiatu (*setosa*, *versicolor*, *virginica*) na podstawie długości i szerokości płatka (*sepal*) i listka kielichu (*petal*).

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
53           6.9         3.1          4.9         1.5 versicolor
54           5.5         2.3          4.0         1.3 versicolor
55           6.5         2.8          4.6         1.5 versicolor
56           5.7         2.8          4.5         1.3 versicolor
101          6.3         3.3          6.0         2.5  virginica
102          5.8         2.7          5.1         1.9  virginica
103          7.1         3.0          5.9         2.1  virginica
104          6.3         2.9          5.6         1.8  virginica
```

Listing 2.1. Przykład danych o kwiatach irysów

Na rys. 2.1, przedstawione zostało rozmieszczenie gatunków kwiatów w zależności od długości i szerokości płatka kwiatu. Wyraźnie widać podział na dwa podstawowe klastry

- gatunek setosa,
- gatunek versicolor i virginica.

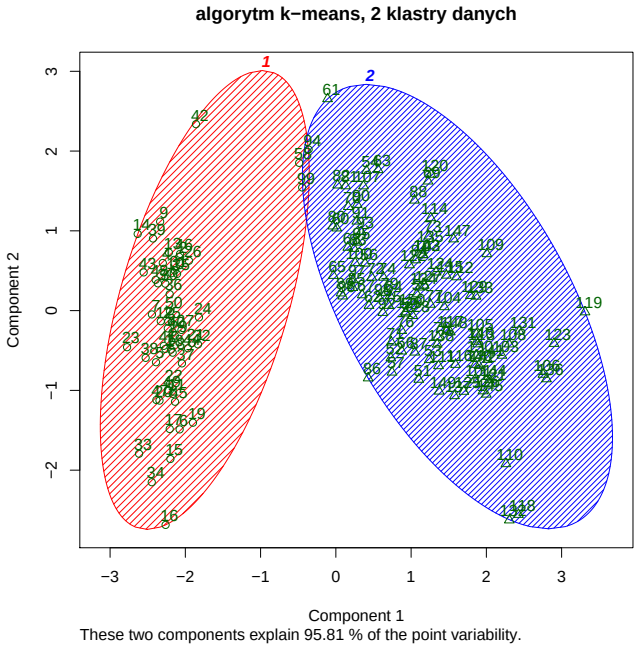


Rys. 2.1. Populacja kwiatów irysów w zależności od szerokości i długości płatka kwiatu

Stosując algorytmy uczenia nienadzorowanego, nie wiemy w jaki sposób klasyfikować dane trenujące. W tym celu możemy wykorzystać algorytm k-means[9], który na podstawie określonych cech grupuje dane w podaną liczbę klastrów. Poddając przedstawione dane działaniu algorytmu k-means, uzyskujemy wynik przedstawiony na rys. 2.2 oraz rys. 2.3.

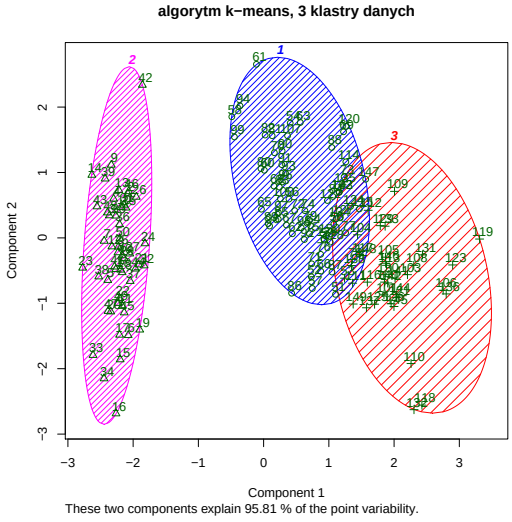
Na rys. 2.2 można zaobserwować, że algorytm zadziała z dużą skutecznością rozróżniając pomiędzy dwoma gatunkami dla dwóch klastrów, tj. pomiędzy gatunkiem setosa, a gatunkami versicolor i virginica, jednak algorytm będzie miał problem rozróżnić pomiędzy gatunkami versicolor i virginica. W tym celu można zbiór podzielić na trzy klastry jak na rys.2.3, ale nie gwarantuje to wystarczającej skuteczności przydziału.
Tym samym można zaobserwować, że gatunek versicolor i virginica nie są rozróżnialne, używając przedstawionych danych wejściowych.
Rozważając sytuacje, w której dodana by została duża liczba dodatkowych wpisów, różnych od aktualnych danych, można by było zaobserwować pojawienie się kolejnego klastra danych, co oznacza pojawienie się nowego gatunku kwiatu.

Wynik działania algorytmu k-means



Rys. 2.2. Wynik działania algorytmu klasteryzacji k-means dla 2 klastrów kwiatów irysów w zależności od szerokości i długości płatka kwiatu

Wynik działania algorytmu k-means

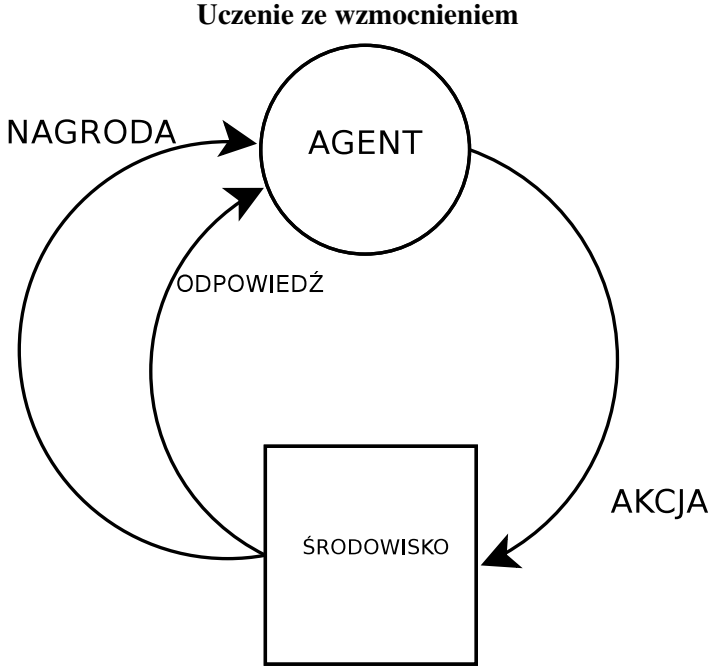


Rys. 2.3. Wynik działania algorytmu klasteryzacji k-means dla 3 klastrów kwiatów irysów w zależności od szerokości i długości płatka kwiatu

2.1.3. Uczenie ze wzmocnieniem

Uczenie ze wzmocnieniem jest obszarem uczenia maszynowego, w którym inteligentny agent wykonując akcję w otoczeniu otrzymuje odpowiednią nagrodę. Robot decyduje jaka akcja zostanie wykonana

na podstawie istniejącej polityki. Algorytmy uczenia ze wzmocnieniem nie wymagają danych trenujących by wykonać swoje zadanie. Na podstawie wiedzy o stanie otoczenia, możliwych do wykonania akcji w danym stanie oraz funkcji nagrody, która agent otrzymuje za wykonanie danych akcji, robot, ucząc się metodą prób i błędów odkrywa sposoby wybierania kolejnych akcji (polityki) tak, aby osiągnąć swój cel.



Rys. 2.4. Schemat uczenia ze wzmocnieniem

Na rys. 2.4 przedstawiono schemat współpracy agenta ze środowiskiem. Agent podejmując akcję w otoczeniu dostaje nagrodę za podjętą decyzję w danym stanie, oraz informacje o nowym, aktualnym stanie środowiska (otoczenie może ulec zmianie po wykonaniu przez robota akcji).

Algorytmy uczenia ze wzmocnieniem dotyczą specyficznych zadań występujących w uczeniu maszynowym. Problem polega na odnalezieniu przez agenta optymalnej akcji do wykonania na podstawie wiedzy o aktualnym jego stanie. W przypadku, gdy to działanie jest powtórzone, możemy mówić o MDP (Procesy decyzyjne Markowa).

Procesem decyzyjnym Markowa nazywamy model środowiska w algorytmach uczenia ze wzmocnieniem, oznaczamy go jako:

$$MDP = (S, A, P(s, s'), R(r, r'), \gamma)$$

gdzie

1. S to zbiór stanów
2. A zbiór możliwych akcji
3. $P(s, s')$ prawdopodobieństwo przejścia do stanu s' ze stanu s po wykonaniu akcji a, w czasie t

4. $R(r, r')$ nagroda po przejściu do stanu s' ze stanu s

5. γ współczynnik regulujący stosunek między nagrodami przewidywanymi w późniejszym czasie i otrzymywanymi aktualnie, $0 < \gamma < 1$

Zadaniem MDP jest uzyskanie takiej polityki robota, która zmaksymalizuje sumę otrzymywanych nagród.

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$$

Podstawowymi algorytmami w metodach uczenia ze wzmocnieniem jest Q-learning[10] oraz SARSA[11].

Algorytmy uczenia ze wzmocnieniem znalazły zastosowanie między innymi w implementacji autonomicznego helikoptera[12], udoskonaleniu działania windy[13], usprawnienia sygnalizacji świetlnej[14] czy budowie inteligentnego robota[15].

2.1.3.1. Q-learning

Celem działania algorytmu Q-learning jest nauczenie się funkcji $Q(s, a)$, która jest przewidywanym wynikiem wykonania danej akcji, w danym stanie.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha * (R_{t+1} + \gamma * maxQt(s_{t+1}, a) - Q_t(s_t, a_t))$$

gdzie

$Q_t(s_t, a_t)$ - poprzednia wartość,

α - stopień uczenia, spełnia zależność $0 < \alpha <= 1$,

γ - stopień dyskontu, spełnia zależność $0 <= \gamma <= 1$,

R_{t+1} - nagroda zaobserwowana po wykonaniu akcji a w stanie t.

2.2. Podsumowanie

Uczenie nienadzorowane posiada tę przewagę nad uczniem nadzorowanym, że nie wymaga oznaczenia danych trenujących oraz potrafi wykryć ukryte zależności pomiędzy cechami przykładów. Jednak w odróżnieniu od obydwu metod, uczenie ze wzmacnianiem jest odmienne w tym, że nie wymaga podawania jakichkolwiek danych przykładowych, a jedynie na podstawie zdefiniowanych nagród i akcji, agent uczy się odpowiedniego działania. By zaimplementować robota uczącego się wykonywania akcji w otoczeniu wybrano implementacje metod uczenia ze wzmacnianiem, ze względu na charakterystykę środowiska odpowiadającą wymaganiom procesów decyzyjnych Markowa.

3. Implementacja

Agent mając wiedzę na temat

- akcji, które może wykonać,
- stanu, w jakim się znajduje,
- zasad, definiujących warunki otrzymania nagrody,
- polityki wybierania akcji

jest w stanie dostosować swoje kolejne działania (wybrane akcje), tak aby uzyskać jak najlepszy wynik w kolejnych iteracjach symulacji. Celem agenta jest dotarcie do punktu końcowego, zdobywając jak największą nagrodę.

Po osiągnięciu celu symulacja jest resetowana, jej wynik jest zapisywany. Robot ucząc się na podstawie poprzednio dokonanych wyborów, buduje strukturę mapującą stan-akcje $Q(S, A)$ do konkretnej wyliczonej wartości. W programie do przedstawienia wartości struktury stan-akcje $Q(S, A)$ wykorzystano wielowymiarową tablicę.

3.1. Struktura stan-akcje $Q(S, A)$

$Q(S, A)$ jest to struktura danych przechowująca wszystkie stany i możliwe do wykonania w nich akcje. Każdej z tych akcji przyporządkowana jest odpowiednia wartość, która jest poddawana modyfikacji przez odpowiedni algorytm uczenia ze wzmocnieniem. Ostatecznym celem działania algorytmu jest uzyskanie takich informacji w strukturze $Q(S, A)$, aby robot korzystając z polityki optymalnej (tzn. takiej, w której agent wybiera zawsze najlepszą według niego akcję) wykonał cel z jak najlepszym wynikiem.

W programie zaimplementowana jest funkcja drukowania wyników symulacji do arkusza kalkulacyjnego, przykładowy wpis zaprezentowany jest na rys. 3.1.

	A	B	C	D	E	F	G	H
1		XEX E0E XEX	XEX O1E XEX	XOX E2E XEX	XEX E3O XEX	XEX E4E XOX	XOX O5E XEX	XEX O6O XEX
2	MOVE_DOWN	9.873307	0	17.81543	0	0	0	0
3	MOVE_UP	1	0	-964.07	0	18.31628	0	0
4	MOVE_LEFT	11.67331	0	17.56035	0	-1	0	0
5	MOVE_RIGHT	10.97538	0	17.63493	0	-1	0	0
6								
7	Goal state achieved with -17880 points and timestep equal to 607							
8								
9								

Rys. 3.1. Przykładowy wynik działania algorytmu Q-learning. Można zauważyć, że do każdego stanu przydzielone są akcje wraz z wartością skumulowanej nagrody wyliczonej za pomocą algorytmu.

Na rys. 3.2 przedstawiono znaczenie symboli w pojedynczym wpisie.

X B X
O 67 O
X O X

Rys. 3.2. Liczba w samym środku reprezentuje numer stanu. Bezpośrednią z nią sąsiadujące są typy obiektów, **B**order - granica, **O**bstacle - przeszkoda, **E**mpty - puste pole, **P**rize - nagroda (stan końcowy)

3.2. Polityka wyboru akcji

Agent mając do dyspozycji jedną z możliwych akcji a ze zbioru wszystkich akcji A, musi podjąć decyzje, która akcję ma wybrać jako następną. W tej implementacji rozważane są dwie polityki

1. ϵ-greedy policy
2. optimal policy

3.2.1. Równowaga pomiędzy eksploracją, a eksploatacją

Od wyboru polityki zależy stopień eksploracji algorytmu. Gdyby agent za każdym razem wybierał najbardziej korzystną według niego akcję, byłby podatny na zakleszczenie w nieoptymalnym rozwiązaniu. Wprowadzając pewną losowość wyboru akcji agent, będzie w stanie odkryć nowe rozwiązania, nawet jeżeli według jego aktualnej wiedzy znalazł to najbardziej optymalne. Powstało dużo prac dokładniej badających to zagadnienie [16], [17].

3.2.2. ϵ-greedy policy

Polityka ϵ-greedy jest strategią wyboru akcji polegającą na wyborze losowej akcji z prawdopodobieństwem ϵ, a w przeciwnym wypadku wybranie najbardziej korzystnej. Agent wybiera akcję, która według niego będzie najbardziej korzystna z prawdopodobieństwem $1 - \epsilon$, a w przeciwnym wypadku wykonuje akcję losową. Parametr ϵ jest regulowany zależnie od potrzeb i musi spełniać zależność $0 < \epsilon < 1$.

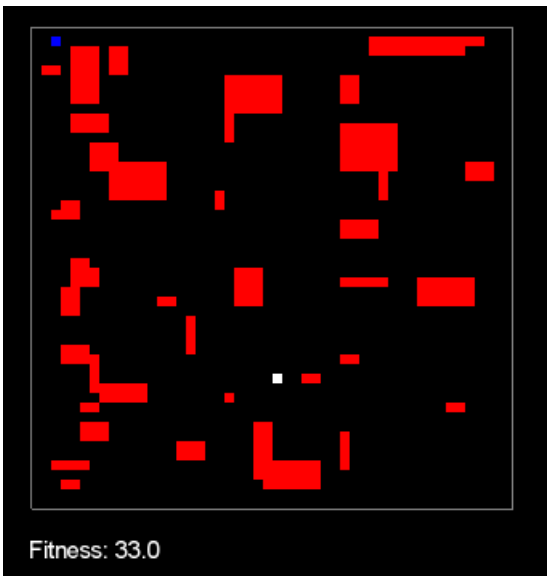
3.2.3. Optimal policy

Polityka optymalna, agent za każdym razem wybiera najkorzystniejszą według jego wiedzy akcję.

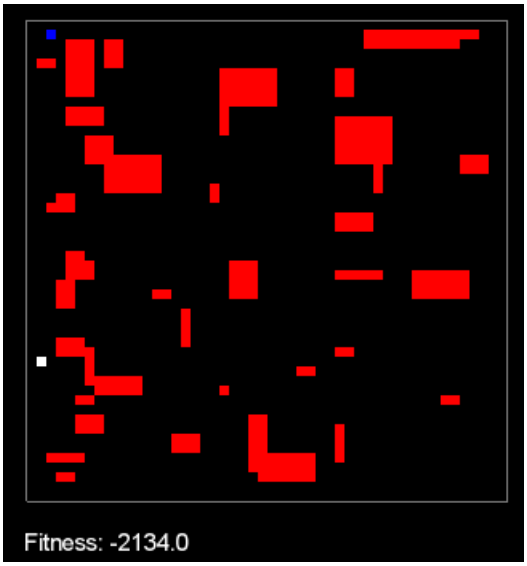
3.3. Symulacja graficzna

Jako symulację graficzna działania algorytmu wykorzystano figury geometryczne reprezentujące: agenta, przeszkody, granice i cel. Agent porusza się na dwuwymiarowej przestrzeni o wymiarach $50 * 50$ pikseli.

Przykładowy stan środowiska w symulacji przedstawiono na rys. 3.3, 3.4.



Rys. 3.3. Przykładowy stan symulacji



Rys. 3.4. Przykładowy stan symulacji

Na rys. 3.3, 3.4, można zauważyć następującą reprezentacje obiektów jako kolor:

- czerwony - przeszkoda,
- biały - agent,
- niebieski - nagroda,
- szary - granica.

3.4. Reprezentacja stanu

Stan w jakim znajduje się aktualnie robot jest przedstawiony w postaci siatki $3 * 3$. Agent jest świadomy obiektów znajdujących się nad nim, pod nim oraz po jego prawej i lewej stronie. Agent może napotkać następujące obiekty

– przeszkoda (O), robot **może** wejść na przeszkodę, jednak traci za to określoną liczbę punktów;

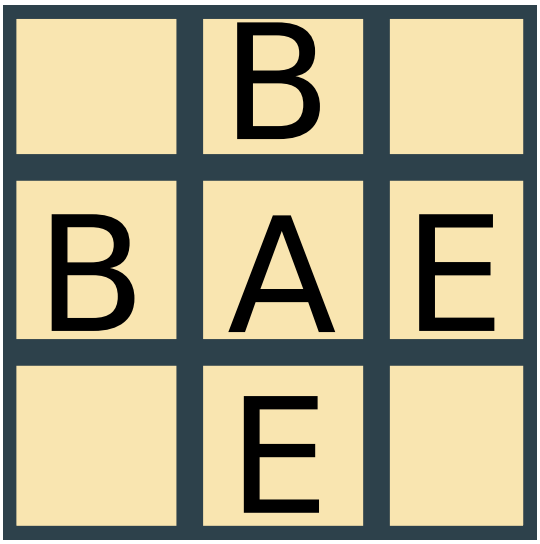
– nagroda (P), stan końcowy. Gdy agent osiąga cel symulacja zapisuje wynik działania i uruchamia się kolejny epizod nauki robota;

– granica (B), granica jest obiektem wyznaczającym pole symulacji, dlatego też nie jest możliwe przemieszczenie się na nią;

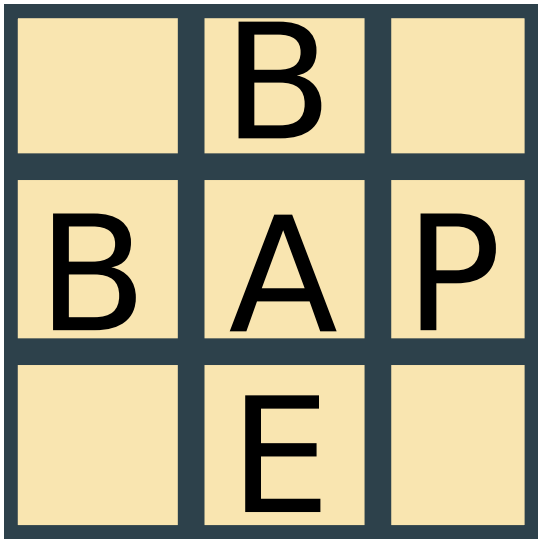
– puste pole (E), pole nie zawierające żadnego z pozostałych obiektów;

Przykładowe stany w jakich może się znaleźć robot przedstawiono na rys. 3.5, 3.6, 3.7 i 3.8.

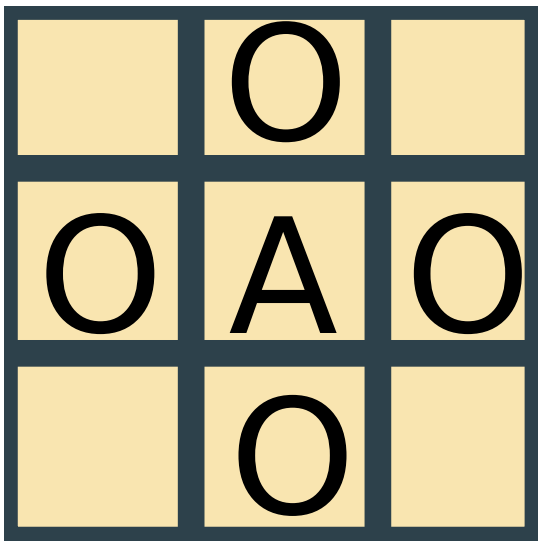
Rys. 3.5. Agent graniczy od górnej strony z przeszkodą, natomiast w pozostałych kierunkach znajdują się puste pola



Rys. 3.6. Agent sąsiaduje od góry i lewej strony z granicą, oznacza to, że robot znajduje się w górnym lewym rogu mapy, a jedynymi akcjami jakie może wykonać to ruch w prawo i ruch w dół



Rys. 3.7. Agent sąsiaduje od góry i lewej strony z granicą, a od prawej z stanem końcowym. Agent wykonując ruch w prawo osiągnie swój cel, powodując tym samym zakończenie i zresetowanie symulacji



Rys. 3.8. Agent graniczy z każdej strony z przeszkodą, jakikolwiek ruch powoduje utratę punktów

3.4.1. Akcje

Robot znajdując się w dowolnym stanie nie graniczącym z obiektem B (granicą), jest w stanie wykonać jedną z czterech akcji:

1. ruch do góry,
2. ruch do dołu,
3. ruch w prawo,

4. ruch w lewo.

Agent **nie może** przemieścić się na granicę, w związku z czym w przypadku sąsiedztwa z granicą, robot posiada ograniczony zakres ruchów.

3.4.2. Wartość nagród

Algorytm, aby wiedzieć czy akcja, którą wykonał jest korzystna czy nie musi otrzymywać nagrodę. W zaimplementowanym środowisku robot dostaje ujemną nagrodę o wartości(-1) po wykonaniu jakiegokolwiek ruchu na puste pole. Dzięki temu zabiegowi agent stara się jak najszybciej dojść do celu, wykonując jak najmniejszą liczbę ruchów. Aby utrudnić dojście do celu agentowi, wprowadzono ujemną nagrodę o wartości -200 po kontakcie z przeszkodą. Robot dąży do odnalezienia takiej drogi, która nie przebiega przez przeszkody. Za dojście do celu uzyskuje pozytywny sygnał o wartości 1000.

Początkowo takie rozwiązanie wydawało się działać prawidłowo, jednak miało jedną poważną wadę. Agent bardzo długi czas spędzał w początkowych etapach symulacji, nie wiedząc w która stronę powinien się udać, aby dojść do celu. Dlatego wprowadzono pozytywną nagrodę o wartości 2, za każdy ruch w stronę nagrody.

3.4.3. Wybór algorytmów

Do implementacji inteligentnego zachowania agenta, wybrano algorytm Q-learning z dziedziny uczenia ze wzmocnieniem. Wartość parametrów:

- discount factor: 0.7
- learning rate: 0.6
- ϵ: 0.2

Na listingu 3.1 przedstawiono implementację algorytmu Q-learning.

```
        State currentState, nextState;
        Action currentAction;

//        lowers discount factor after an arbitrary amount of time
        if(renderTimer % 1000 == 0 && discountFactor > 0){
            System.out.println("lowering discountFactor");
            discountFactor -= 0.1d;
        }

        if(renderTimer % threshold == 0){
```

```
//            start out in an initial state as the current state
            currentState = State.identifyState(agent.getX(), agent.
               getY(),
Environment.getInstance().getEnvironmentState());
            List<Action> currentlyPossibleActions = new ArrayList<>(
               qValues.get(currentState).keySet());

//            select one among all possible actions for the current
   state
//            epsilon-greedy policy takes random action once in a
   while, otherwise chooses the best one
            double d = Math.random();
            if (d > epsilon){

//                get best action for current state
                double Qmax = qValues.get(currentState).get(
                   currentlyPossibleActions.get(0));
                Action bestAction = null;

                for(Map.Entry<Action, Float> actionEntry : qValues.
                   get(currentState).entrySet()){
                    if(actionEntry.getValue() >= Qmax){
                        Qmax = actionEntry.getValue();
                        bestAction = actionEntry.getKey();
                    }
                }
                if(bestAction == null){
                    for(Map.Entry<Action, Float> actionEntry :
                       qValues.get(currentState).entrySet())
                        System.out.println(actionEntry);
                    throw new NullPointerException("Empty action!");
                }
                currentAction = bestAction;
            }
            else {
//                random action
                currentAction = currentlyPossibleActions.get(
                   ThreadLocalRandom.current().nextInt(0,
```

```
currentlyPossibleActions.size()));
            }

//            consider transitioning to state, after executing
   current action
            nextState = State.considerNextState(currentAction, agent.
               getX(), agent.getY());

//            find qMax for nextState
            float Qmax = 0;

            for(Map.Entry<Action, Float> actionEntry : qValues.get(
               nextState).entrySet()){
                if(actionEntry.getValue() > Qmax){
                    Qmax = actionEntry.getValue();
                }
            }

//            get the reward for the (current state, current action,
   next state) tuple
            float reward = agent.returnReward(currentState,
               currentAction, agent.getX(), agent.getY());


//            update Q value
            float currentStateQValue = qValues.get(currentState).get(
               currentAction);
            float qValue = currentStateQValue + learningRate *(reward
                + discountFactor *Qmax - currentStateQValue);
            qValues.get(currentState).put(currentAction, qValue);


//            set current state to next state
            currentState = nextState;

            agent.move(currentAction, Gdx.graphics.getDeltaTime());
            timeStep++;
```

```
return reward;
```

Listing 3.1. Implementacja algorytmu Q-learning

3.5. Wykorzystane technologie

Kod został napisany w języku Java w wersji 8. Do generowania grafiki został wykorzystany framework LibGDX. W celu notowania wyników do plików kalkulacyjnych została wykorzystana biblioteka Apache POI. Aby zmniejszyć ilości kodu użyto biblioteki Project Lombok, pozwalającej na automatyczne generowane getterów, setterów, konstruktorów lub metod hashcode i equals. Do budowania i zarządzania projektem wykorzystano narzędzie Gradle.

3.6. Napotkane problemy

Przy projektowaniu aplikacji, największy problem stanowiło odpowiednie przedstawienie aktualnego stanu środowiska i robota. Nieodpowiednie odwzorowanie stanu, może powodować niepoprawną naukę agenta. Zakładając rozmiar mapy $50 * 50$, można zauważyć, że przy naiwnej reprezentacji stanu, tzn. takiej, w której każda możliwa kombinacja stanu mapy jest reprezentowana osobno, powstaje ogromna ilość możliwych akcji w tablicy wartości $Q(S, A)$ Ostatecznie postanowiono przedstawić stan w postaci siatki $3 * 3$.

Kolejnym problemem, który napotkano przy implementacji działania algorytmów jest ich testowanie. Algorytmy uczenia ze wzmocnieniem są niełatwymi metodami do testowania, niepoprawność działania agenta może wynikać z wielu różnych błędów implementacji, jak również z niepoprawnego modelu środowiska czy reprezentacji stanu. Negatywnie na działanie robota wpływają też nieodpowiednio dobrane parametry implementacji lub polityki.

3.7. Obsługa symulacji

Do obsługi symulacji zostały zaimplementowane następujące funkcje:

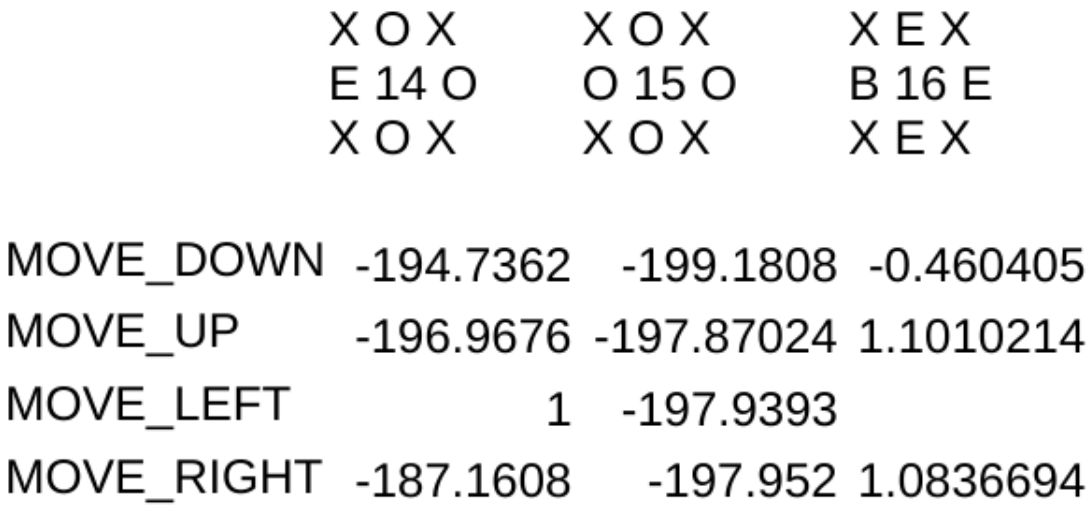
- Spowolnienie symulacji - wciskając przycisk $-$, można spowolnić działanie symulacji
- Przyspieszenie symulacji - wciskając przycisk $+$, można przyspieszyć działanie symulacji
- Zresetowanie symulacji - klawisz $ENTER$ służy do resetowania symulacji
- Zmiana polityki robota - używając przycisku p, można sterować polityką robota wybierając pomiędzy ϵ-greedy, a optimal policy

3.8. Wynik działania

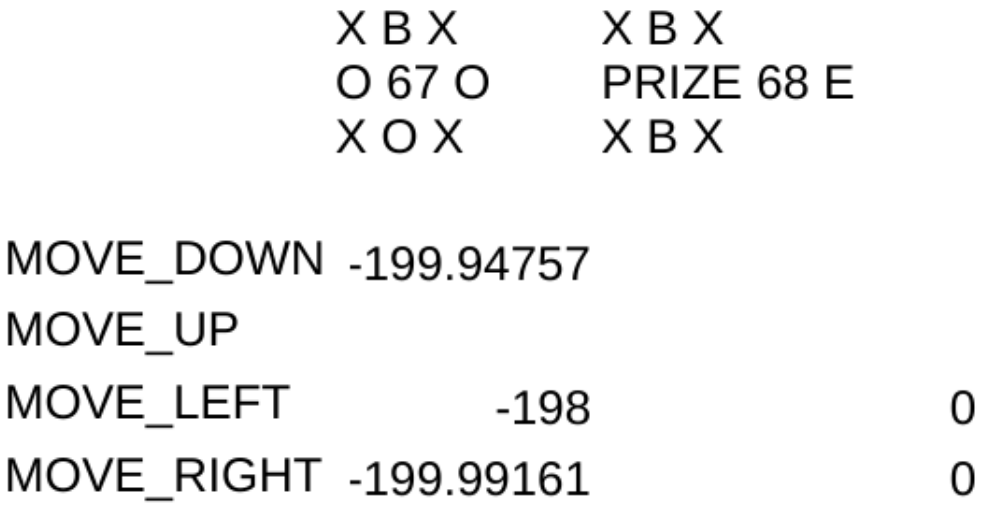
Algorytm z łatwością poradził sobie z wyznaczeniem najkrótszej, najbardziej korzystnej drogi w zaprojektowanym środowisku. Po niedługim procesie nauki, robot generuje skuteczną tablicę wartości $Q(S, A)$. Jako, że zaimplementowany został algorytm off-policy Q-learning, można zaobserwować brak wpływu nauczania na politykę wybierania akcji. Pomimo to, że agent znalazł już satysfakcjonujące rozwiązanie to nadal jest zaangażowany w eksploracje nowych stanów. Gdyby wykorzystano algorytm on-policy SARSA, robot pod wpływem uzyskanych wyników zmienił by strategię wyboru kolejnych akcji, wykorzystując politykę optymalną. Przykładowe wyniki przedstawiono na rys. 3.9, 3.10 i 3.11.

	X E X O 7 E X O X	X O X E 8 O X E X	X O X E 9 E X O X
MOVE_DOWN	-199.672	-0.89987	-197.941
MOVE_UP	1	-197.87	-185.226
MOVE_LEFT	-192.931	1	1
MOVE_RIGHT	-0.99974	-199.181	-0.92027

Rys. 3.9. Przykładowy wynik działania algorytmu. Można zaobserwować poprawnie wyznaczone wartości akcji. Nagroda znajduje się w lewym górnym rogu mapy, dlatego też nie opłaca się agentowi wykonać akcję w przeciwnym kierunku. Agent jeżeli ma taką możliwość wykonuje ruch w takich stan, żeby nie znaleźć się na przeszkodzie



Rys. 3.10. Przykładowy wynik działania algorytmu. Analogicznie jak na obrazku wyżej, agent wykonuje podobne wybory. Można zauważyć, że robot znajdując się wewnątrz przeszkody (stan nr. 15), każdy ruch uważa za niekorzystny ale jednak z delikatną przewagą ruchu do góry lub w lewo (w kierunku nagrody). W przypadku, gdy agent graniczy z lewej strony z przeszkodą jak w stanie nr. 16, niedostępna jest akcja MOVE LEFT



Rys. 3.11. Przykładowy wynik działania algorytmu. Na tym obrazku widać stan nr. 68, w którym agent nigdy się nie znalazł. Jest to logiczna konsekwencja tego, że nagroda (stan końcowy) znajduje się w lewym górnym obszarze mapy, a więc nie może sąsiadować z granicą od dołu

4. Podsumowanie

Zaimplementowany agent poprawnie nauczył się osiągać cel w zaprojektowanym środowisku. Odpowiednio zaimplementowany algorytm Q-learning pozwolił robotowi na efektywne generowanie wyników tablicy $Q(S, A)$, dzięki czemu bardzo szybko odnalazł optymalną drogę do osiągnięcia stanu końcowego. Ze względu na prostotę środowiska agent zaledwie po około siódmej iteracji osiąga najkorzystniejszy wynik. Dla lepszych testów działania algorytmu w przyszłości odpowiednie byłoby wprowadzenie bardziej zaawansowanej struktury środowiska. Analogicznie istotną kwestią byłby poprawny dobór parametrów algorytmu Q-learning oraz usprawniona reprezentacja stanu.

Bibliografia

[1] John McCarthy. „What is artificial intelligence". W: *URL: http://www-formal. stanford. edu/jmc/whatisai. html* (2007), s. 38.

[2] Poole David L. i Mackworth Alan K. *Artificial Intelligence: Foundations of Computational Agents*. New York, NY, USA: Cambridge University Press, 2010. ISBN: 0521519004, 9780521519007.

[3] Pedro Domingos. „A few useful things to know about machine learning". W: *Communications of the ACM* 55.10 (2012), s. 78–87.

[4] Loredana Firte, Camelia Lemnaru i Rodica Potolea. „Spam detection filter using KNN algorithm and resampling". W: *Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing*. IEEE. 2010, s. 27–33.

[5] Muhammad N Marsono, M Watheq El-Kharashi i Fayez Gebali. „Binary LNS-based naïve Bayes inference engine for spam control: noise analysis and FPGA implementation". W: *Computers & Digital Techniques, IET* 2.1 (2008), s. 56–62.

[6] R Deepa Lakshmi i N Radha. „Spam classification using supervised learning techniques". W: *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*. ACM. 2010, s. 66.

[7] Irena Koprinska i in. „Learning to classify e-mail". W: *Information Sciences* 177.10 (2007), s. 2167–2187.

[8] Wenjuan Li i in. „Towards designing an email classification system using multi-view based semi-supervised learning". W: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*. IEEE. 2014, s. 174–181.

[9] Adam Coates i Andrew Y Ng. „Learning feature representations with k-means". W: *Neural Networks: Tricks of the Trade*. Springer, 2012, s. 561–580.

[10] Christopher JCH Watkins i Peter Dayan. „Q-learning". W: *Machine learning* 8.3-4 (1992), s. 279–292.

[11] Gavin A Rummery i Mahesan Niranjan. „On-line Q-learning using connectionist systems". W: (1994).

[12] Pieter Abbeel i in. „An application of reinforcement learning to aerobatic helicopter flight". W: *Advances in neural information processing systems* 19 (2007), s. 1.

[13] A Barto i RH Crites. „Improving elevator performance using reinforcement learning". W: *Advances in neural information processing systems* 8 (1996), s. 1017–1023.

[14] Marco Wiering i in. „Multi-agent reinforcement learning for traffic light control". W: *ICML*. 2000, s. 1151–1158.

[15] Hajime Kimura, Tom Yamashita i Shigenobu Kobayashi. „Reinforcement learning of walking behavior for a four-legged robot". W: *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*. T. 1. IEEE. 2001, s. 411–416.

[16] Sebastian B Thrun. „Efficient exploration in reinforcement learning". W: (1992).

[17] Eric Wiewiora. „Efficient Exploration for Reinforcement Learning". Prac. dokt. Citeseer, 2004.