

**Uniwersytet Jagielloński w Krakowie**

Wydział Fizyki, Astronomii i Informatyki Stosowanej

**Mateusz Jancarz**

Nr albumu: 1107719

# **Uczenie maszynowe z językiem Python**

Praca magisterska na kierunku Informatyka

Praca wykonana pod kierunkiem  
dra hab. Andrzeja Kapanowskiego  
Instytut Fizyki

Kraków 2015

## **Oświadczenie autora pracy**

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Kraków, dnia

Podpis autora pracy

## **Oświadczenie kierującego pracą**

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Kraków, dnia

Podpis kierującego pracą

Składam serdeczne podziękowania Panu doktorowi habilitowanemu Andrzejowi Kapanowskiemu za udzieloną pomoc oraz zaangażowanie w powstanie tej pracy.

## Streszczenie

Niniejsza praca przedstawia aplikację, która symuluje giełdę zakładów sportowych, oraz gracza-bota wykorzystującego uczenie maszynowe do zwiększenia skuteczności swojej gry.

Początek pracy zawiera wprowadzenie do dziedziny uczenia maszynowego oraz przybliża jeden z przykładów jego zastosowania w robotyce. Opisane zostają narzędzia wykorzystane w projekcie, w których skład wchodzi język Python oraz wybrane biblioteki, takie jak NumPy i Matplotlib.

Omówiono uczenie ze wzmocnieniem, jako nowe podejście w obszarze uczenia się maszyn. Opisano wszystkie składowe potrzebne do definicji tego typu algorytmów. Przedstawiono matematyczny model tego paradygmatu, czyli proces decyzyjny Markowa.

Zaprezentowano dwa popularne algorytmy z klasy zwanej metodami różnic czasowych: Q-learning oraz Sarsa. Zostają one przedstawione w możliwie kompletnej wersji uwzględniającej wykorzystanie tzw. śladów aktywności, które w teorii pomagają przyspieszyć proces nauki.

Stworzono implementację wymienionych algorytmów, na podstawie której zbudowana jest symulacja gry na giełdzie. Przy pomocy przeprowadzonych testów sprawdzono skuteczność zaprogramowanej symulacji. Szczegółowo opisana jest rola oraz dobór wartości poszczególnych parametrów nauki w kontekście rozwiązywanego problemu. Wspomnianą aplikację uzupełniają programy pomocnicze służące do wizualizacji danych giełdowych na wykresach oraz generowania sztucznych sekwencji zmian kursu.

**Słowa kluczowe:** sztuczna inteligencja, uczenie maszynowe, uczenie ze wzmocnieniem, metody różnic czasowych, algorytm Q-learning, algorytm Sarsa, ślady aktywności, proces decyzyjny Markowa

**English title:** Machine learning with Python

### **Abstract**

This thesis presents the application simulating sports betting markets and a bot player that performs transactions on markets and utilizes machine learning for improving it's performance.

In the beginning, the introduction to machine learning is provided with examples of it's effective usage in robotics. Development tool set used is presented, which contains Python language, NumPy and Matplotlib modules among the others. Reinforcement learning is presented as a novel approach in machine learning. Mathematic model of the approach is explained with Markov decision process.

Two concrete algorithms of temporal difference learning paradigm are presented: Q-learning and Sarsa. The description contains possibly complex versions, which contain usage of eligibility traces that, in theory, help to improve learning process efficiency.

Implementaion of mentioned algorithms is developed and used to build simulation of trading on betting exchange. Effectiveness of the application is examined via various experiments. The role and selection of all the learning parameter values is described in a context of the problem being solved. The application is supplemented with other tools built that are used to visualize exchange markets data and generate artificial price change sequences.

**Keywords:** artificial intelligence, machine learning, reinforcement learning, temporal differences learning, Q-learning algorithm, Sarsa algorithm, eligibility traces, Markov decision process

# Spis treści

<b>Spis tabel</b> . . . . .	4
<b>Spis rysunków</b> . . . . .	5
<b>Listingi</b> . . . . .	6
<b>1. Wstęp</b> . . . . .	7
1.1. Cele i założenia . . . . .	7
1.2. Znaczenie uczenia maszynowego . . . . .	7
1.3. Potrzeba zastosowania maszynowego uczenia się . . . . .	8
1.4. Zawartość pracy . . . . .	9
<b>2. Opis wykorzystywanych narzędzi</b> . . . . .	10
2.1. Wprowadzenie do języka Python . . . . .	10
2.2. Korzystanie z NumPy . . . . .	11
2.2.1. Podstawowe operacje . . . . .	11
2.2.2. Indeksowanie . . . . .	12
2.2.3. Brakujące wartości . . . . .	12
2.2.4. Funkcje matematyczne . . . . .	12
2.3. Korzystanie z Matplotlib . . . . .	13
2.4. Korzystanie z PyMongo . . . . .	14
<b>3. Uczenie maszynowe</b> . . . . .	15
3.1. Definicja uczenia się . . . . .	15
3.2. Proces uczenia się w kontekście programów komputerowych . . . . .	15
3.3. Rodzaje uczenia maszynowego . . . . .	17
<b>4. Uczenie się ze wzmocnieniem</b> . . . . .	18
4.1. Zadanie uczenia się ze wzmocnieniem . . . . .	18
4.2. Procesy decyzyjne Markowa . . . . .	19
4.3. Strategie . . . . .	20
4.4. Model rynku . . . . .	20
4.5. Klasy metod uczenia się ze wzmocnieniem . . . . .	23
4.6. Metody różnic czasowych . . . . .	23
4.6.1. Algorytm Q-learning . . . . .	23
4.6.2. Algorytm Sarsa . . . . .	24
4.6.3. Ślady aktywności - TD( $\lambda$ ) . . . . .	24
4.6.4. Q( $\lambda$ ) . . . . .	25
<b>5. Projektowanie aplikacji</b> . . . . .	26
5.1. Zastosowanie maszynowego uczenia się w konstruowaniu strategii giełdowych . . . . .	26
5.2. Giełdy zakładów sportowych . . . . .	28
5.2.1. Handel na giełdzie . . . . .	29
5.3. Charakterystyka danych giełdowych . . . . .	30
5.4. Elementy projektu . . . . .	32
5.4.1. Moduł z implementacjami algorytmów uczenia się . . . . .	32

5.4.2.	Moduł narzędzi pomocniczych . . . . .	33
5.4.3.	Moduł symulacji . . . . .	33
5.5.	Wizualizacja danych . . . . .	34
5.6.	Korzystanie z implementacji algorytmu nauki . . . . .	35
5.7.	Projektowanie symulacji . . . . .	38
5.7.1.	Definicja stanów . . . . .	38
5.7.2.	Definicja akcji . . . . .	39
5.7.3.	Definicja macierzy wartości Q . . . . .	40
5.7.4.	Proces nauki . . . . .	40
5.8.	Testy symulacji . . . . .	41
5.8.1.	Znaczenie i dobór parametrów algorytmu . . . . .	41
5.8.2.	Testy symulacji na rzeczywistych danych . . . . .	46
<b>6.</b>	<b>Podsumowanie . . . . .</b>	<b>48</b>
<b>A.</b>	<b>Uruchamianie projektu . . . . .</b>	<b>50</b>
A.1.	Wymagania . . . . .	50
A.2.	Konfiguracja źródła danych rynkowych . . . . .	50
A.3.	Uruchamianie generatora sztucznych danych . . . . .	51
A.4.	Aplikacja wizualizacji danych . . . . .	52
A.5.	Konfiguracja i uruchamianie symulacji nauki . . . . .	52
<b>B.</b>	<b>Kod źródłowy . . . . .</b>	<b>54</b>
	<b>Bibliografia . . . . .</b>	<b>59</b>

## Spis tabel

4.1	Rozkład prawdopodobieństwa przejść między stanami. . . . .	21
4.2	Strategie dla modelu rynku. . . . .	21
5.1	Przykłady dla wprowadzonych pojęć dla meczu piłkarskiego. . . . .	28
5.2	Opis pól w strukturze obiektu reprezentującego rynek. . . . .	32
5.3	Zestawienie rezultatów symulacji wykorzystującej wygenerowane dane rynkowe. . . . .	45
5.4	Zestawienie rezultatów symulacji gry na giełdzie zakładów sportowych wykorzystującej archiwalne dane. . . . .	46
A.1	Dostępne zmienne konfiguracyjne oraz atrybuty uruchomieniowe skryptu <code>simulation_launcher.py</code> . . . . .	53



## Spis rysunków

1.1	Schemat testowania różnych zachowań przez robota kroczącego. . . . .	9
2.1	Przykład wykresów wygenerowanych za pomocą biblioteki Matplotlib. . . . .	13
3.1	Uczenie się jako konkretyzacja algorytmu. . . . .	16
4.1	Schemat procesu uczenia się ze wzmocnieniem. . . . .	18
5.1	Model systemu wykorzystującego naukę przez prognozowanie. . . . .	27
5.2	Model systemu wykorzystującego uczenie przez wzmocnienie. . . . .	27
5.3	Okno aplikacji do wizualizacji danych giełdowych na wykresach. . . . .	35
5.4	Demonstracja sekwencji zmian ceny jednej z opcji rynkowych na wykresie. . . . .	38
5.5	Wstęgi Bollingera na wykresie zmiany ceny. . . . .	39
5.6	Rezultat symulacji wykorzystującej sztuczne dane z wartością parametru $\varepsilon$ ustawioną na 1. . . . .	42
5.7	Rezultat symulacji wykorzystującej sztuczne dane z wartością parametru $\varepsilon$ ustawioną na 0. . . . .	42
5.8	Rezultat symulacji wykorzystującej sztuczne dane z wartością parametru $\alpha$ ustawioną na 0.9. . . . .	43
5.9	Rezultat symulacji wykorzystującej sztuczne dane z wartością parametru $\gamma$ ustawioną na 0. . . . .	44

# Listingi

2.1	Inicjalizacja wybranych struktur danych w języku Python. . . . .	10
2.2	Sesja interaktywna z użyciem modułu NumPy. . . . .	11
2.3	Indeksowanie w NumPy. . . . .	12
2.4	Obsługa brakujących wartości w NumPy. . . . .	12
2.5	Funkcje matematyczne w NumPy. . . . .	12
2.6	Sesja interaktywna z użyciem Matplotlib. . . . .	13
2.7	Korzystanie z modułu PyMongo. . . . .	14
4.1	Algorytm uczenia się ze wzmocnieniem. . . . .	19
4.2	Algorytm Q-learning. . . . .	23
4.3	Algorytm Sarsa. . . . .	24
4.4	Algorytm Q-learning lambda. . . . .	25
5.1	Przykładowy obiekt rynku. . . . .	31
5.2	Przykładowa konstrukcja obiektu agenta. . . . .	36
5.3	Końcowa postać macierzy wartości Q dla symulacji na sztucznych danych uruchomionej z wartością parametru $\gamma$ ustawioną na 0. . . . .	45
A.1	Domyślna zawartość pliku konfiguracyjnego settings.cfg. . . . .	51
B.1	Implementacja algorytmów Q-learning oraz Sarsa zawarta w module td_learning. . . . .	54

# 1. Wstęp

Tematem niniejszej pracy jest *Uczenie maszynowe z językiem Python*. Praca stanowi wprowadzenie do tematyki uczenia maszynowego jako jednego z fragmentów szerokiego obszaru sztucznej inteligencji. Wspomniany w jej tytule język Python zastosowano w celu prostego osvajania się z poznawanymi algorytmami.

## 1.1. Cele i założenia

Celem pracy jest przedstawienie praktycznego zastosowania maszynowego uczenia się. Stworzono w jej ramach aplikację, która symuluje grę na giełdzie zakładów sportowych. Zaprogramowany gracz-bot przeprowadza transakcje, a dzięki wbudowanemu modułowi uczącemu stara się w miarę zdobywanego doświadczenia zwiększać skuteczność swojej gry. Całość projektu została zbudowana przy użyciu języka Python.

Celem pośrednim można określić fazę poznawania możliwości uczenia maszynowego ze wzmocnieniem oraz implementację konkretnych algorytmów nauki.

## 1.2. Znaczenie uczenia maszynowego

Ludzkość produkuje w dzisiejszych czasach ogromną ilość danych. Dzięki powszechnemu dostępowi do sieci Internet tworzenie, gromadzenie oraz przesyłanie danych jest bardzo proste i tanie. Danych już jest dużo, a ich ilość cały czas rośnie. Statystyki podają, że około 90% z nich zostało wytworzonych w ostatnich kilku latach [1]. Samo posiadanie nawet ogromnej ilości informacji może dzisiaj nie gwarantować przewagi konkurencyjnej w biznesie, więc coraz większy wpływ ma ich skuteczne wykorzystanie. Surowe dane są często niewiele warte, a liczy się wiedza z nich pozyskana. Fakt ten jest motorem napędowym do rozwoju dziedzin związanych z przetwarzaniem ogromnych ilości danych w celu formowania nietrywialnych wniosków z nich płynących.

Nic dziwnego, że o pojęciach takich jak *data mining*, *big data* czy *sztuczna inteligencja* jest coraz głośniejsze. Wśród nich jest również termin *uczenie maszynowe*. Zyskuje ono na popularności zwłaszcza w ostatniej dekadzie, mimo że zostało zaproponowane już w roku 1959 przez Arthura Samuela[2]. Zdefiniował on uczenie maszynowe jako *dziedzinę wiedzy, która przekazuje komputerom zdolność nauki bez potrzeby zaprogramowania ich wprost*. Warto wspomnieć, że aż do około połowy lat dziewięćdziesiątych na rynku prawie nie istniały przypadki komercyjnego zastosowania uczenia maszynowego [3].

Pierwszym tego typu projektem uznaje się oprogramowanie uczące się grać w warcaby stworzone w 1959 przez wspomnianego Arthura Samuela, ówczesnego pracownika firmy IBM [4]. Dzisiaj zastosowań jest wiele, poczynając od systemów rekomendacyjnych w sklepach internetowych, a kończąc na oprogramowaniu do przetwarzania języka naturalnego czy sterującego robotem.

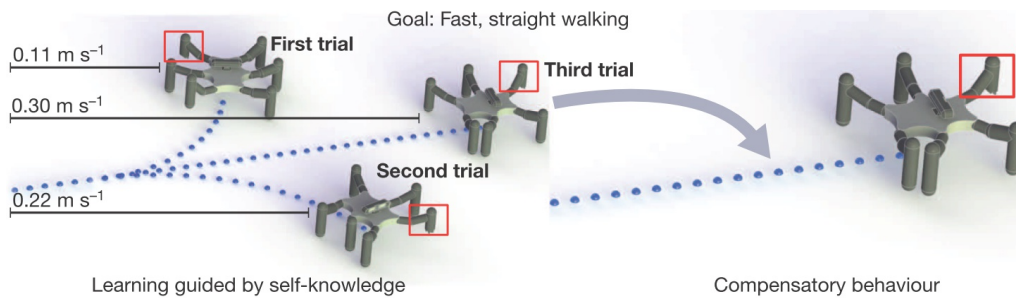
### 1.3. Potrzeba zastosowania maszynowego uczenia się

Sztuczna inteligencja jest szeroko stosowana w robotyce. Wykorzystuje się ją do tworzenia możliwie autonomicznych jednostek zdolnych do samodzielnej adaptacji do środowiska, a także przeróżnych czynników losowych. Często za przykład bierze się zdolność zwierząt do radzenia sobie w przypadku odniesienia ran. Analogiczna sytuacja może się przydarzyć w świecie robotów, dlatego ważnym jest, aby mogły się one do nich w odpowiedni sposób dostosowywać.

Nie sposób jest przewidzieć wszystkie tego typu sytuacje, dlatego optymalnym rozwiązaniem może być przekazanie robotom zdolności do samodzielnej nauki radzenia sobie z nimi. Obiecującym podejściem do tego typu przypadków jest *uczenie ze wzmocnieniem*, czyli nauka na bazie interakcji agenta ze środowiskiem. Czasami określane jest ono jako sztuczna inteligencja w mikroskali, ponieważ wykorzystywane w tym paradygmacie algorytmy muszą reagować w sposób autonomiczny, aby wykonać określony cel [5]. Ciekawym przykładem zastosowania uczenia ze wzmocnieniem jest 6-nożny robot kroczący, który został zaprogramowany tak, aby radzić sobie z szeroką gamą urazów mechanicznych [6].

Twórcy robota bazują na hipotezie, w której zwierzęta rozumieją przestrzeń dostępnych zachowań oraz ich potencjał na podstawie posiadanego doświadczenia i podejmują inteligentne wybory dotyczące prób testujących te zachowania w celu ich ewaluacji. Zdefiniowali oni mapę *zachowań do wartości*, która pozwala przewidywać skuteczność tysięcy różnych zachowań.

Na tej podstawie wymaganiem wstępnym w budowie tego rodzaju systemu staje się jedynie precyzyjne zdefiniowanie skończonej przestrzeni stanów i zachowań oraz określenie celu (w tym przypadku wydajnego poruszania się), a także ustalenie strategii do testowania par stan-zachowanie. Dzięki temu robot sam jest w stanie nauczyć się takich zachowań, które są optymalne do osiągnięcia celu. Nie zmienia się to nawet w przypadku urazów mechanicznych, bo w razie ich wystąpienia robot sam dostosowuje sposób chodzenia, aby zrekompensować poniesione straty. Schemat to obrazujący został przedstawiony na rysunku 1.1.



Rysunek 1.1: Schemat testowania różnych zachowań przez robota kroczącego.

Źródło: [6]

## 1.4. Zawartość pracy

Rozdział 1 zawiera wprowadzenie do tematyki opisywanej w pracy magisterskiej. W rozdziale 2 opisano podstawowe narzędzia wykorzystane do stworzenia projektu praktycznego. Krótkie wprowadzenie do uczenia maszynowego i uczenia się ze wzmocnieniem zawarto w rozdziałach 3 i 4. Projektowanie aplikacji przedstawiono w rozdziale 5. Rozdział ten zawiera również opis domeny, w której zastosowanie ma tworzona aplikacja, czyli giełdy zakładów sportowych. Pod koniec tej części pracy przedstawiono wyniki testów zaprogramowanej symulacji. Rozdział 6 stanowi podsumowanie wyników pracy. W dodatku A jest zawarty szczegółowy opis konfiguracji i sposobu uruchamiania projektu, dodatek B natomiast zawiera kod źródłowy zaimplementowanych algorytmów.

Literatura z dziedziny maszynowego uczenia się jest bardzo bogata. Dobrym wprowadzeniem do tematu jest książka Cichosza [7]. Zastosowanie Pythona do budowy systemów uczących się (ale bez uczenia się ze wzmocnieniem) jest opisane przez Richerta i Coelho [8]. Przewodnikiem po temacie uczenia się ze wzmocnieniem może być książka Suttona i Barto [9].

W projekcie praktycznych zastosowano następujące narzędzia<sup>1</sup>:

- Python 2.7 [10],
- NumPy 1.9 [11],
- Matplotlib 1.4 [12].

<sup>1</sup> Podane wersje były wykorzystywane w procesie implementacji. Aplikacja została przetestowana również na starszych wersjach, które wymieniono w sekcji A.1.

## 2. Opis wykorzystywanych narzędzi

Każdy większy projekt programistyczny bazuje na pewnym zestawie gotowych narzędzi takich jak biblioteki, moduły, standardy protokołów itp. W tym rozdziale zostaną opisane podstawowe narzędzia informatyczne przydatne przy pracy nad projektem z dziedziny uczenia maszynowego. Punktem wyjścia będzie oczywiście sam język Python.

### 2.1. Wprowadzenie do języka Python

Na wstępie tego rozdziału warto wyjaśnić powód wyboru głównego narzędzia znajdującego się w tytule pracy, czyli języka Python. Język ten nie bez powodu jest określany jako *wykonywalny pseudokod*. Kod w nim zapisany jest czytelny, a dzięki niewielkiemu narzutowi składniowemu relatywnie zwięzły. Spora część podstawowych elementów języka jest wbudowana w jego składnię, a przykładem może być tutaj inicjalizacja niektórych struktur danych, np. list lub słowników:

Listing 2.1: Inicjalizacja wybranych struktur danych w języku Python.

```
empty_list = []    # lub list ()
empty_tuple = ()  # lub tuple ()
empty_dict = {}   # lub dict ()
empty_set = set ()
```

Kolejnym, równie ważnym powodem wyboru tego języka jest jego popularność. Język jest udostępniany na licencji *open source*, co znacznie ułatwia społeczności aktywny wkład w jego rozwój poprzez tworzenie nowych narzędzi. Mimo bogatej biblioteki standardowej powstało wiele specjalistycznych bibliotek zewnętrznych. Większość z tych najpopularniejszych dotyczy zagadnień naukowych, w których to Python jest szczególnie popularny.

Dobrym przykładem jest moduł NumPy, który pomaga m.in. w przeprowadzaniu złożonych obliczeń na tablicach. Mocną stroną tej biblioteki jest fakt, że część definiowanych operacji wykonywana jest w czasie procesu kompilacji, co niewątpliwie pomaga w wydajnym przeprowadzaniu obliczeń. Dobrym uzupełnieniem jest moduł Matplotlib, który służy do generowania wykresów. Oba wymienione narzędzia będą stosowane w tej pracy. Ich opis znajduje się w kolejnych sekcjach rozdziału.

Programowanie w języku Python jest również proste w porównaniu do innych języków. Typy sprawdzane są dynamicznie, a za zarządzanie pamięcią odpowiada (podobnie jak np. w języku Java) *garbage collector*. Wymienione zalety stanowią jednak przyczynę stosunkowo niskiej wydajności w porównaniu do języków kompilowanych.

Popularność języka zapoczątkowała wersja 2.0 wydana w 2000 roku. Twórcy języka zdecydowali się na stworzenie nowej gałęzi, czego efektem było wydanie w roku 2008 wersji 3.0. Mimo na pozór jedynie subtelnych (choć wielu) różnic, wersja ta jest w znaczącej mierze niekompatybilna z poprzednią. Pomimo rozwoju nowej gałęzi, wcześniejsza wersja jest nadal wspierana i wciąż bardzo popularna. W pracy będzie wykorzystywana właśnie ona, a konkretnie wersja 2.7.

Praca z językiem wymaga interpretera tłumaczącego wysokopoziomowe instrukcje języka na kod maszynowy w czasie wykonywania programu. Stanowi on dodatkowo swoisty pomost między językiem, a systemem operacyjnym, co pozwala łatwo tworzyć oprogramowanie bez względu na platformę. Interpreter jest dostępny dla wszystkich popularnych systemów, a z niektórymi z nich (np. dystrybucjami z rodziny Linux) jest dostarczany.

## 2.2. Korzystanie z NumPy

NumPy [11] jest pakietem oferującym wielowymiarowe macierze, funkcje działające na macierzach, narzędzia algebry liniowej, transformatę Fouriera, liczby losowe, narzędzia do integracji kodu z C/C++ i Fortranem. Macierze powinny być tworzone za pomocą następujących funkcji: `numpy.array()`, `numpy.zeros()`, `numpy.ones()`, `numpy.empty()`, `numpy.arange()`.

### 2.2.1. Podstawowe operacje

Podstawowe operacje na macierzach przedstawia listing 2.2.

Listing 2.2: Sesja interaktywna z użyciem modułu NumPy.

```
>>> import numpy
>>> a = numpy.arange(6) # odmiana range()
>>> a # instancja klasy numpy.ndarray
array([0, 1, 2, 3, 4, 5])
>>> b = a.reshape(2, 3)
>>> b # inny widok na te same dane
array([[0, 1, 2],
       [3, 4, 5]])
>>> b[1][2] = 9 # zmiana widoczna w a i b
>>> c = a.reshape(3, 2).copy() # prawdziwa kopia
>>> b.ndim # liczba wymiarow (2 axes)
2
>>> b.shape
(2, 3)
>>> b.dtype # typ danych, np. int, float, complex
dtype('int64')
>>> a ** 2 # dzialania wykonywane elementwise
array([ 0, 1, 4, 9, 16, 81])
>>> a * 5 # zwracana jest nowa macierz
array([ 0, 5, 10, 15, 20, 45])
>>> a + 1
array([ 1, 2, 3, 4, 5, 10])
```

### 2.2.2. Indeksowanie

Silną stroną NumPy są różnorodne sposoby dostępu do macierzy. Możliwe jest użycie obiektu macierzy jako indeksu przykładowo w celu jej przefiltrowania.

Listing 2.3: Indeksowanie w NumPy.

```
>>> a[numpy.array([3, 4, 5])] # wskazujemy wybrane indeksy
array([3, 4, 9])
>>> a > 3 # warunek przenosi się na elementy
array([False, False, False, False,  True,  True], dtype=bool)
>>> a[a > 3]
array([4, 9])
>>> a[a > 3] = 3 # wybieramy elementy do wymiany
>>> a
array([0, 1, 2, 3, 3, 3])
```

### 2.2.3. Brakujące wartości

Przetwarzane dane czasem mogą zawierać nieprawidłowe wartości, które możemy oznaczyć z użyciem stałej `numpy.NaN`. NumPy pozwala wygodnie przetwarzać takie dane za pomocą specjalnych metod.

Listing 2.4: Obsługa brakujących wartości w NumPy.

```
# Symulujemy dane z nieprawidłowymi wartościami.
>>> x = numpy.array([1, 2, numpy.NaN, 3, 4])
>>> x
array([ 1.,  2., nan,  3.,  4.])
>>> numpy.isnan(x)
array([False, False,  True, False, False], dtype=bool)
>>> x[~numpy.isnan(x)] # zaprzeczenie logiczne macierzy
array([ 1.,  2.,  3.,  4.])
>>> numpy.mean(x[~numpy.isnan(x)])
2.5
```

### 2.2.4. Funkcje matematyczne

Ponadto biblioteka udostępnia szereg funkcji matematycznych, którymi można z łatwością przetwarzać całe tablice.

Listing 2.5: Funkcje matematyczne w NumPy.

```
>>> x = numpy.arange(4, dtype=float) # ustalony typ danych
>>> x
array([ 0.,  1.,  2.,  3.])
>>> numpy.exp(x)
array([ 1.,          2.71828183,  7.3890561 , 20.08553692])
>>> numpy.sqrt(x)
array([ 0.,          1.,          1.41421356,  1.73205081])
>>> numpy.sin(x)
array([ 0.,          0.84147098,  0.90929743,  0.14112001])
>>> y = numpy.array([5, 6, 7, 8]) # na bazie listy Pythona
>>> numpy.add(x, y) # działanie macierzowe
```



```

array([ 5,  7,  9, 11])
>>> numpy.dot(x, y) # iloczyn skalarny
44.0
>>> x.sum(), y.min(), y.max()
(6.0, 5, 8)

```

## 2.3. Korzystanie z Matplotlib

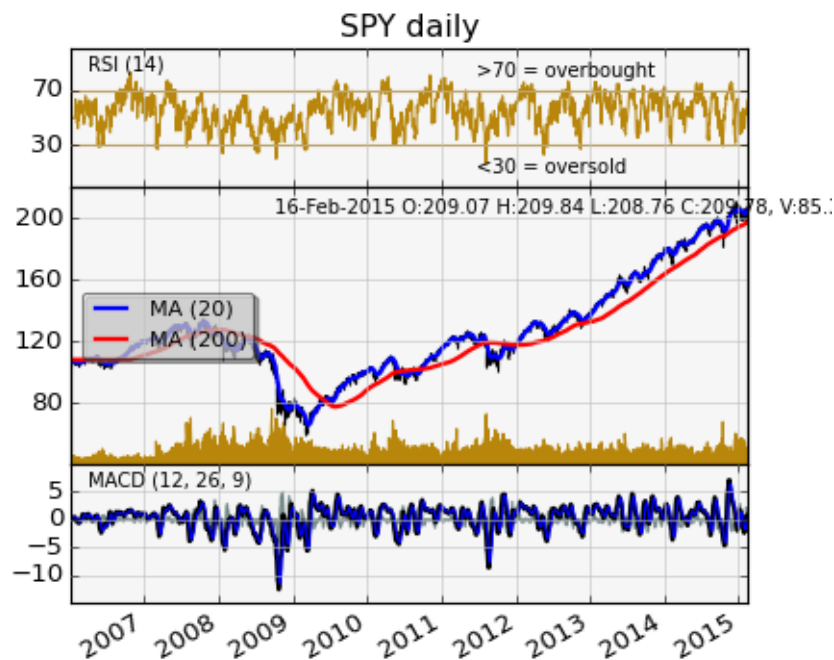
Matplotlib [12] jest to pythonowa biblioteka do tworzenia wykresów 2D i 3D w różnych formatach. Rysunek 2.1 prezentuje możliwości biblioteki. Na listingu 2.6 przedstawiono sposób wygenerowania prostego wykresu funkcji liniowej.

Listing 2.6: Sesja interaktywna z użyciem Matplotlib.

```

>>> import matplotlib.pyplot as plt # pakiet pyplot
>>> x = range(10)
>>> y = [2*i+1 for i in range(10)]
>>> plt.scatter(x, y) # Matlab interface
>>> plt.title("Funkcja liniowa y = 2 x + 1")
>>> plt.xlabel("x")
>>> plt.ylabel("y")
>>> plt.grid()
>>> plt.show()

```



Rysunek 2.1: Przykład wykresów wygenerowanych za pomocą biblioteki Matplotlib.

Źródło: [12]

## 2.4. Korzystanie z PyMongo

PyMongo jest biblioteką języka Python ułatwiającą pracę z bazami danych wykorzystującymi system MongoDB. Realizację połączenia z bazą danych oraz przeprowadzanie podstawowych operacji przy pomocy biblioteki PyMongo przedstawia listing 2.7.

Listing 2.7: Korzystanie z modułu PyMongo.

```
from pymongo import MongoClient

# Otwarcie polaczenia z serwerem na okreslonym hoscie i porcie.
client = MongoClient('localhost', 27017)

# Dostep do obiektu bazy.
db = client.test_database

# Dostep do okreslonej kolekcji.
users_collection = db.users_collection

new_user = {'firstname' : 'Mateusz',
            'lastname' : 'Jancarz',
            'hometown' : 'Krakow'}

# Dodanie obiektu (uzytkownika) do kolekcji.
users_collection.insert_one(new_user)

# Pozyskanie listy wszystkich uzytkownikow.
all_users = users_collection.find()

# Pozyskanie listy wszystkich uzytkownikow z Krakowa.
users_from_krakow = users_collection.find({'hometown' : 'Krakow'})

# Pozyskanie liczby uzytkownikow z Krakowa.
users_from_krakow.count()
```

## 3. Uczenie maszynowe

Maszynowe uczenie się jest to dziedzina interdyscyplinarna, czerpiąca głównie z informatyki oraz statystyki. Jako jeden z elementów składowych *sztucznej inteligencji* skupia się na budowie oraz analizie sztucznych systemów zdolnych do uczenia się na podstawie danych [7].

### 3.1. Definicja uczenia się

W procesie uczenia się można wyróżnić kilka kluczowych elementów, które formułują wymagania w stosunku do działań systemu, który można określić jako uczący się. Pierwszym z nich jest *zmiana*. Nie każda jednak zmiana w systemie może mieć charakter uczenia się. Jako kontrprzykład można tutaj podać zjawisko zapominania. Zmianą o jaką nam w tym przypadku chodzi jest taka, która w nosi *poprawę* do działania systemu. Taka korzystna zmiana jest czynnikiem wymaganym, ale nie wystarczającym. Wymaganą cechą poprawy jest jej *autonomiczność*, a więc źródło jej pochodzenia ma być w samym systemie. System, który się uczy sam zmienia się na lepsze. Za naukę nie można uznać dodanie nowej funkcjonalności do systemu, albo poprawę już istniejącej. Taką - mimo, że korzystną zmianę - nie możemy uznać za naukę. Zmiana choć autonomiczna, nie może być spowodowana samoistnie czy też losowo, ale pod wpływem czynników, bądź okoliczności zewnętrznych. Taki zewnętrzny wpływ możemy wtedy nazwać *doświadczeniem* zdobywanym przez system uczący się.

Definicja uczenia się, do której doprowadziły powyższe rozważania, może brzmieć następująco [7]:

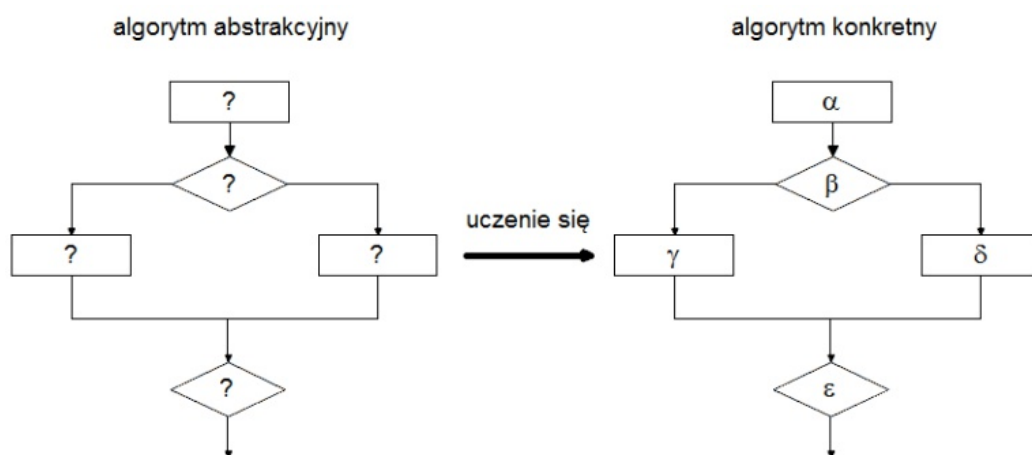
**Definicja 1.** *Uczeniem się systemu jest każda autonomiczna zmiana w systemie na podstawie doświadczeń, która doprowadzi do poprawy jakości jego działania.*

### 3.2. Proces uczenia się w kontekście programów komputerowych

Potrzeba uczenia się programów komputerowych bynajmniej nie jest związana z inżynierią oprogramowania i chęcią zmniejszenia wysiłku projektantów czy programistów. Motywacja do tworzenia tego typu systemów wynika z trudności w rozwiązywanych przez nie problemach, które często wymagają niekonwencjonalnego podejścia. W przypadku niektórych stawianych problemów bardziej opłaca się stworzyć algorytm dający maszynie zdolność do nauki rozwiązywania problemu zamiast zaprogramowania jej wprost. Może

to wynikać na przykład z nikłej wiedzy lub wręcz jej braku o środowisku pracy systemu.

Programy uczące się wykorzystują abstrakcyjny, w pewien sposób sparametryzowany algorytm wykonywania zadania. Uczenie się wówczas polega na takiej analizie danych historycznych, aby te parametry odpowiednio dobrać i w konsekwencji przekształcić w odpowiedni algorytm konkretny, rozwiązujący problem stawiany przez konstruktora systemu. Ilustruje to schemat 3.1.



Rysunek 3.1: Uczenie się jako konkretyzacja algorytmu.

Źródło: opracowanie własne na podstawie: [7]

Uzyskane w wyniku nauki parametry określa się w zależności od rodzaju wykonywanego zadania *wiedzą* lub *umiejętnością*. Często są one określane jako *hipoteza*, podkreślając fakt, iż najczęściej nie można ich uznać jako niezawodne i mogą wymagać rewizji [7].

### 3.3. Rodzaje uczenia maszynowego

Rozwijając temat uczenia maszynowego należy na samym początku uporządkować dotychczasową wiedzę o nim i przedstawić różne jego rodzaje, aby znać wachlarz możliwości, z których możemy wybierać konkretne metody najlepiej pasujące do charakterystyki rozwiązywanego problemu.

Szereg algorytmów uczenia się można podzielić według kilku kryteriów. Najczęściej spotyka się podział biorący pod uwagę tzw. informację trenującą. Przyjmijmy, że system uczący się ma za zadanie poprzez obserwację pewnych danych wejściowych generować na ich podstawie odpowiedzi, a uczenie się jest procesem zwiększania efektywności tego zabiegu. W takim przypadku informacja trenująca instruuje w pewien sposób ucznia co do właściwego sposobu odpowiadania. Kryterium w tym podziale jest to, czy taka informacja jest dostępna dla systemu - innymi słowy - czy uczeń posiada nauczyciela. Z tego względu wymienia się:

- uczenie z nadzorem (ang. *supervised learning*),
- uczenie bez nadzoru (ang. *unsupervised learning*).

Dodatkowo wyróżnia się trzecią grupę, o której wspomniano już we wstępie pracy, czyli uczenie ze wzmocnieniem, określane również jako naukę poprzez interakcję. Ten sposób nauki jest znacznie bardziej skupiony na celu nauki niż pozostałe podejścia. Również istnieje w nim zewnętrzne źródło informacji trenującej, ma ono jednak charakter raczej krytyka niż nauczyciela [13]. Pełni on rolę wartościującą dla prób testujących agenta uczącego się. Uczeń otrzymuje on od niego wzmocnienia uzależnione od skutków podejmowanych działań. System stara się maksymalizować wartości wzmocnień w długim horyzoncie czasowym, co formalnie ma reprezentować zwiększanie efektywności wykonywanych przez niego zadań.

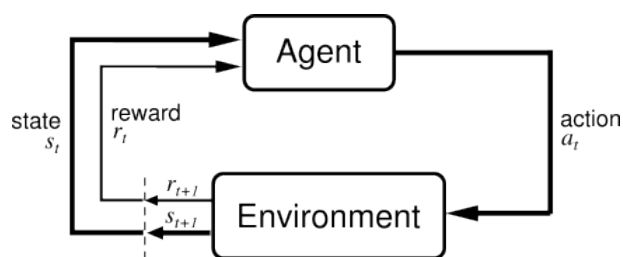
## 4. Uczenie się ze wzmocnieniem

W tym paradygmacie uczenia występuje dynamiczna interakcja z otoczeniem. Uczeń obserwuje stany środowiska, wykonuje akcje, obserwuje efekty tych akcji przez otrzymywanie nagród (wartości wzmocnienia). Zadaniem ucznia jest znalezienie strategii wybierania akcji, która prowadzi do długoterminowej maksymalizacji nagród [7].

Podstawy teoretyczne uczenia się ze wzmocnieniem są związane z procesami decyzyjnymi Markowa i metodami programowania dynamicznego. Zostaną przedstawione potrzebne pojęcia z tej dziedziny korzystając z prostego modelu rynku (giełdy).

### 4.1. Zadanie uczenia się ze wzmocnieniem

Jak wspomniano kluczowym aspektem uczenia ze wzmocnieniem jest nauka strategii wybierania akcji. Jej konstruowanie odbywa się na podstawie bezpośrednich interakcji ze środowiskiem metodą prób i błędów. Informacja trenująca ma charakter wartościujący (oceniający) jakość działania ucznia, a nie instruujący go, w jaki sposób ma działać. Scenariusz uczenia się ze wzmocnieniem można przedstawić za pomocą abstrakcyjnego algorytmu jak na rysunku 4.1.



Rysunek 4.1: Schemat procesu uczenia się ze wzmocnieniem.

Źródło: [14]

Na listingu 4.1 uogólniony algorytm uczenia się ze wzmocnieniem zapisany w pseudokodzie.

Listing 4.1: Algorytm uczenia się ze wzmocnieniem.

dla wszystkich kroków czasu  $t$  wykonaj:  
 obserwuj aktualny stan  $x_t$ ;  
 wybierz akcję  $a_t$  do wykonania w stanie  $x_t$ ;  
 wykonaj akcję  $a_t$ ;  
 obserwuj wzmocnienie  $r_t$  i następny stan  $x_{t+1}$ ;  
 ucz się na podstawie doświadczenia  $\langle x_t, a_t, r_t, x_{t+1} \rangle$ .

Środowisko jest przez ucznia *niekontrolowalne*. Uczeń powinien maksymalizować swoje nagrody *długoterminowo*. Najczęściej przyjmuje się, że uczeń powinien maksymalizować zdyskontowaną sumę otrzymanych nagród

$$\mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right], \quad (4.1)$$

gdzie  $\gamma \in [0, 1]$  jest *współczynnikiem dyskontowania* a  $r_t$  to *wzmocnienia* w kolejnych chwilach czasu  $t$ . Współczynnik  $\gamma$  jest miarą "dalekowzroczności" ucznia.

W przypadku uczenia się ze wzmocnieniem możemy mieć do czynienia z różnymi trybami nauki zależnymi od sposobu modyfikacji strategii decyzyjnej ucznia. My skupimy się na *trybie inkrementacyjnym*, gdzie strategia decyzyjna może podlegać modyfikacji w każdym kroku czasu pod wpływem pozyskiwanego doświadczenia.

## 4.2. Procesy decyzyjne Markowa

Modelem matematycznym zadania uczenia się ze wzmocnieniem jest problem decyzyjny Markowa [7].

**Definicja 2. (Proces decyzyjny Markowa)** *Proces decyzyjny Markowa jest zdefiniowany jako czwórka  $\langle X, A, \rho, \delta \rangle$ , przy czym*

- $X$  jest skończonym zbiorem stanów,
- $A$  jest skończonym zbiorem akcji,
- $\rho$  jest funkcją wzmocnienia,
- $\delta$  jest funkcją przejść stanów.

We wcześniejszej notacji definicja oznacza, że w każdym kroku czasu  $t$  nagroda  $r_t$  (liczba rzeczywista) jest realizacją zmiennej losowej  $\rho(x_t, a_t)$ , a stan  $x_{t+1}$  jest realizacją zmiennej losowej  $\delta(x_t, a_t)$ . Zgodnie z *własnością Markowa* funkcje  $\rho$  i  $\delta$  nie zależą od historii. Wygodnie jest wprowadzić oznaczenia:

$$R(x, a) = \mathbb{E}[\rho(x, a)], \quad (4.2)$$

$$P_{xy}(a) = \Pr[\delta(x, a) = y]. \quad (4.3)$$

### 4.3. Strategie

**Definicja 3. (Strategia)** *Strategią dla procesu decyzyjnego Markowa jest dowolna funkcja  $\pi: X \rightarrow A$ .*

Podana definicja określa strategię *stacjonarną* i *deterministyczną*, co upraszcza teorię. W praktyce algorytmy uczenia się ze wzmocnieniem wykorzystują z reguły strategie niestacjonarne i niedeterministyczne, które pozwalają na pracę w środowiskach zmiennych i nie do końca przewidywalnych. System posługuje się strategią  $\pi$ , jeżeli w każdym kroku  $t$  wykonuje on akcję  $a_t = \pi(x_t)$ . Do oceniania strategii służą *funkcja wartości* i *funkcja wartości akcji*.

**Definicja 4. (Funkcja wartości)** *Dla procesu decyzyjnego Markowa funkcja wartości ze względu na strategię  $\pi$  jest dla każdego stanu  $x \in X$  określona następująco:*

$$V^\pi(x) = E_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \mid x_0 = x \right]. \quad (4.4)$$

**Definicja 5. (Funkcja wartości akcji)** *Dla procesu decyzyjnego Markowa funkcja wartości akcji ze względu na strategię  $\pi$  jest dla każdej pary  $(x, a)$ ,  $x \in X$ ,  $a \in A$  określona następująco:*

$$Q^\pi(x, a) = E_\pi \left[ \rho(x, a) + \sum_{t=1}^{\infty} \gamma^t r_t \mid x_0 = x, a_0 = a \right]. \quad (4.5)$$

Warto zauważyć, że  $V^\pi(x) = Q^\pi(x, \pi(x))$ , czyli  $Q^\pi$  zawiera więcej informacji niż  $V^\pi$ .

### 4.4. Model rynku

Rozważmy prosty model rynku, który jest środowiskiem dla procesu decyzyjnego Markowa. Istnieje możliwość kupowania bądź sprzedaży jednego pakietu akcji. Jego cena może przyjmować dwie umowne wartości: 1 lub 2. Warto dodatkowo uwzględnić tendencję zmian kursu (rosnąca  $\uparrow$ , malejąca  $\downarrow$ , bez zmian  $\square$ ), dlatego w zbiorze  $X$  określamy cztery stany. Można powiedzieć, że w określeniu stanu jest zakodowana historia sięgająca jeden krok w przeszłość.

$$X = \{1\square, 1\downarrow, 2\square, 2\uparrow\} \quad (\text{zbiór stanów}), \quad (4.6)$$

$$A = \{0, 1\} \quad (\text{zbiór akcji}), \quad (4.7)$$

gdzie  $a = 0$  oznacza *sprzedaj lub nie kupuj*,  $a = 1$  znaczy *kupuj lub nie sprzedawaj*. Rozkłady prawdopodobieństw przejść nie zależą od akcji i są podane w tablicy 4.1.

Wartości oczekiwane nagród są określone poniżej, przy czym stała  $K > 0$  określa różnicę wartości pakietu akcji pomiędzy poziomami kursu 1 i 2:

$$R(x, a) = \begin{cases} (1-p)K & \text{dla } x = 1\downarrow \text{ i } a = 1, \\ pK & \text{dla } x = 1\square \text{ i } a = 1, \\ -(1-p)K & \text{dla } x = 2\uparrow \text{ i } a = 1, \\ -pK & \text{dla } x = 2\square \text{ i } a = 1, \\ 0 & \text{w pozostałych przypadkach.} \end{cases} \quad (4.8)$$



Tabela 4.1: Rozkład prawdopodobieństwa przejść między stanami. Przejścia nie zależą od akcji. Jeżeli  $p = 1$ , to środowisko jest deterministyczne i mamy powtarzający się ciąg przejść  $(1 \downarrow, 1\Box, 2 \uparrow, 2\Box)$ . Jeżeli  $p = 0.5$ , to mamy niedeterministyczne przeskoki między cenami 1 i 2.

$P_{xy}$	$y = 1 \downarrow$	$y = 1\Box$	$y = 2 \uparrow$	$y = 2\Box$
$x = 1 \downarrow$	0	$p$	$1 - p$	0
$x = 1\Box$	0	$1 - p$	$p$	0
$x = 2 \uparrow$	$1 - p$	0	0	$p$
$x = 2\Box$	$p$	0	0	$1 - p$

Tabela 4.2: Strategie dla modelu rynku.

Stany	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$
$1 \downarrow$	0	1	1	0
$1\Box$	0	1	1	1
$2 \uparrow$	0	1	0	1
$2\Box$	0	1	0	0

Badane strategie zamieszczono w tabeli 4.2. Analizę modelu warto rozpocząć od najprostszych strategii.

**Strategia  $\pi_1$ :** Ta strategia oznacza brak zaangażowania w kupowanie pakietu akcji. Wyznaczamy funkcję wartości i funkcję wartości akcji.

$$V^{\pi_1}(2 \uparrow) = V^{\pi_1}(2\Box) = V^{\pi_1}(1 \downarrow) = V^{\pi_1}(1\Box) = 0. \quad (4.9)$$

$$Q^{\pi_1}(1 \downarrow, 0) = 0, \quad Q^{\pi_1}(1 \downarrow, 1) = (1 - p)K, \quad (4.10)$$

$$Q^{\pi_1}(1\Box, 0) = 0, \quad Q^{\pi_1}(1\Box, 1) = pK, \quad (4.11)$$

$$Q^{\pi_1}(2 \uparrow, 0) = 0, \quad Q^{\pi_1}(2 \uparrow, 1) = -(1 - p)K, \quad (4.12)$$

$$Q^{\pi_1}(2\Box, 0) = 0, \quad Q^{\pi_1}(2\Box, 1) = -pK. \quad (4.13)$$

Zauważmy, że  $Q^{\pi_1}(x, a)$  odtwarza  $R(x, a)$ .

**Strategia  $\pi_2$ :** Ta strategia polega na stałym trzymaniu kupionego pakietu akcji. Obliczamy funkcję wartości.

$$V^{\pi_2}(1 \downarrow) = p\gamma V^{\pi_2}(1\Box) + (1 - p)[K + \gamma V^{\pi_2}(2 \uparrow)], \quad (4.14)$$

$$V^{\pi_2}(1\Box) = (1 - p)\gamma V^{\pi_2}(1\Box) + p[K + \gamma V^{\pi_2}(2 \uparrow)], \quad (4.15)$$

$$V^{\pi_2}(2 \uparrow) = p\gamma V^{\pi_2}(2\Box) + (1 - p)[-K + \gamma V^{\pi_2}(1 \downarrow)], \quad (4.16)$$

$$V^{\pi_2}(2\Box) = (1 - p)\gamma V^{\pi_2}(2\Box) + p[-K + \gamma V^{\pi_2}(1 \downarrow)]. \quad (4.17)$$

W przypadku deterministycznym  $p = 1$  mamy proste rozwiązanie:

$$V^{\pi_2}(1 \downarrow) = \gamma V^{\pi_2}(1\Box) = -V^{\pi_2}(2 \uparrow) = -\gamma V^{\pi_2}(2\Box) = \frac{\gamma K}{1 + \gamma^2}. \quad (4.18)$$

Dla przypadku  $p = 0.5$  otrzymujemy:

$$V^{\pi_2}(1 \downarrow) = V^{\pi_2}(1 \square) = -V^{\pi_2}(2 \uparrow) = -V^{\pi_2}(2 \square) = \frac{K}{2}. \quad (4.19)$$

Obliczamy funkcję wartości akcji.

$$Q^{\pi_2}(1 \downarrow, 0) = V^{\pi_2}(1 \downarrow) - (1 - p)K, \quad (4.20)$$

$$Q^{\pi_2}(1 \downarrow, 1) = V^{\pi_2}(1 \downarrow), \quad (4.21)$$

$$Q^{\pi_2}(1 \square, 0) = V^{\pi_2}(1 \square) - pK, \quad (4.22)$$

$$Q^{\pi_2}(1 \square, 1) = V^{\pi_2}(1 \square), \quad (4.23)$$

$$Q^{\pi_2}(2 \uparrow, 0) = V^{\pi_2}(2 \uparrow) + (1 - p)K, \quad (4.24)$$

$$Q^{\pi_2}(2 \uparrow, 1) = V^{\pi_2}(2 \uparrow), \quad (4.25)$$

$$Q^{\pi_2}(2 \square, 0) = V^{\pi_2}(2 \square) + pK, \quad (4.26)$$

$$Q^{\pi_2}(2 \square, 1) = V^{\pi_2}(2 \square). \quad (4.27)$$

**Strategia  $\pi_3$ :** Ta strategia jest optymalna, co można sprawdzić tworząc strategię zachłanną na bazie strategii  $\pi_1$  lub też obserwując funkcje wartości akcji. Funkcje wartości dla stanów przy stosowaniu strategii  $\pi_3$  możemy obliczyć z układu równań.

$$V^{\pi_3}(1 \downarrow) = p\gamma V^{\pi_3}(1 \square) + (1 - p)[K + \gamma V^{\pi_3}(2 \uparrow)], \quad (4.28)$$

$$V^{\pi_3}(1 \square) = (1 - p)\gamma V^{\pi_3}(1 \square) + p[K + \gamma V^{\pi_3}(2 \uparrow)], \quad (4.29)$$

$$V^{\pi_3}(2 \uparrow) = p\gamma V^{\pi_3}(2 \square) + (1 - p)\gamma V^{\pi_3}(1 \downarrow), \quad (4.30)$$

$$V^{\pi_3}(2 \square) = (1 - p)\gamma V^{\pi_3}(2 \square) + p\gamma V^{\pi_3}(1 \downarrow). \quad (4.31)$$

W szczególnym przypadku  $p = 1$  rozwiązanie jest proste:

$$V^{\pi_3}(2 \uparrow) = \gamma V^{\pi_3}(2 \square) = \gamma^2 V^{\pi_3}(1 \downarrow) = \gamma^3 V^{\pi_3}(1 \square) = \frac{\gamma^3 K}{1 - \gamma^4}. \quad (4.32)$$

Również w przypadku  $p = 0.5$  istnieją proste związki:

$$V^{\pi_3}(1 \downarrow) = V^{\pi_3}(1 \square) = \frac{(2 - \gamma)K}{4(1 - \gamma)}, \quad (4.33)$$

$$V^{\pi_3}(2 \uparrow) = V^{\pi_3}(2 \square) = \frac{\gamma K}{4(1 - \gamma)}. \quad (4.34)$$

Obliczamy funkcję wartości akcji.

$$Q^{\pi_3}(1 \downarrow, 0) = V^{\pi_3}(1 \downarrow) - (1 - p)K, \quad (4.35)$$

$$Q^{\pi_3}(1 \downarrow, 1) = V^{\pi_3}(1 \downarrow), \quad (4.36)$$

$$Q^{\pi_3}(1 \square, 0) = V^{\pi_3}(1 \square) - pK, \quad (4.37)$$

$$Q^{\pi_3}(1 \square, 1) = V^{\pi_3}(1 \square), \quad (4.38)$$

$$Q^{\pi_3}(2 \uparrow, 0) = V^{\pi_3}(2 \uparrow), \quad (4.39)$$

$$Q^{\pi_3}(2 \uparrow, 1) = V^{\pi_3}(2 \uparrow) - (1 - p)K, \quad (4.40)$$

$$Q^{\pi_3}(2 \square, 0) = V^{\pi_3}(2 \square), \quad (4.41)$$

$$Q^{\pi_3}(2 \square, 1) = V^{\pi_3}(2 \square) - pK. \quad (4.42)$$

## 4.5. Klasy metod uczenia się ze wzmocnieniem

Wyróżnia się trzy podstawowe klasy metod do rozwiązywania problemów uczenia się ze wzmocnieniem: programowanie dynamiczne, metody Monte Carlo, oraz metody różnic czasowych. Każda z klas ma swoje wady oraz zalety. Programowanie dynamiczne wymaga ścisłej definicji modelu środowiska. Metody Monte Carlo nie mają takiego wymagania, ponadto nie nadają się do rozwiązywania problemów inkrementacyjnie, co powoduje, że są bardziej wymagające pamięciowo. Metody różnic czasowych jako będące stosunkowo nowym podejściem w uczeniu ze wzmocnieniem łączą zalety obu wymienionych grup.

## 4.6. Metody różnic czasowych

Stosunkowo nowym podejściem w uczeniu się ze wzmocnieniem są tzw. metody różnic czasowych (ang. *Temporal Differences*, TD). Jest to klasa metod dla predykcji w wieloetapowych problemach predykcyjnych, czyli takich, w których poprawność przewidywań nie może być zweryfikowana natychmiast, lecz dopiero po pewnej liczbie kroków [13].

W tym paradygmacie podczas uczenia się funkcji wartości zaczynamy z arbitralnie zainicjowanej wielkości  $V_0$  i pewnej strategii  $\pi$ . Operacja aktualizacji funkcji wartości  $V$  dla stanu  $x_t$  w tej metodzie wygląda następująco:

$$V(x_t) \leftarrow V(x_t) + \alpha[r_t + \gamma V(x_{t+1}) - V(x_t)]. \quad (4.43)$$

Wyróżnia się dwa sposoby implementacji algorytmów TD:

- *on-policy* (ze strategią) - gdzie rozważana jest wartość najlepszej akcji dla obranej strategii;
- *off-policy* (bez strategii) - gdzie rozważana jest wartość najlepszej akcji ignorując aktualną strategię.

### 4.6.1. Algorytm Q-learning

Popularną implementacją podejścia TD jest algorytm Q-learning zaproponowany przez C. Watkina w 1989. Algorytm uczy się funkcji wartości akcji  $Q$ , co sugeruje jego nazwa. Określa się go jako *off-policy*, gdzie bezpośrednio o wyborze akcji  $a$  w stanie  $x$  decyduje wartość  $Q(x, a)$ . Uczeń może posługiwać się strategią inną niż ta, której się uczy. Algorytm Q-learning jest zaprezentowany na listingu 4.2.

Listing 4.2: Algorytm Q-learning.

```
zainicjalizuj  $Q(x, a)$  dla wszystkich stanów  $x$  i akcji  $a$ ;  
dla wszystkich kroków czasu  $t$  wykonaj:  
  obserwuj aktualny stan  $x_t$ ;  
  wybierz akcję  $a_t$  na bazie  $x_t$  i  $Q$ ;  
  wykonaj akcję  $a_t$ ;  
  obserwuj wzmocnienie  $r_t$  i następny stan  $x_{t+1}$ ;  
  wyznacz  $\Delta \leftarrow r_t + \gamma \max_a Q(x_{t+1}, a) - Q(x_t, a_t)$ ;  
  aktualizuj  $Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha \Delta$ .
```

W każdy kroku czasowym aktualizowana jest wartość funkcji wartości akcji  $Q$ . Formuła do jej wyliczenia wygląda następująco:

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha[r_t + \gamma \max_a Q(x_{t+1}, a) - Q(x_t, a_t)], \quad (4.44)$$

gdzie  $Q$  to wartość akcji,  $x$  to stan środowiska,  $t$  jest krokiem czasowym (kolejną iteracją),  $\alpha \in [0, 1]$  jest współczynnikiem uczenia się, a  $\gamma \in [0, 1]$  to współczynnik dyskontowania nagród.

Parametr  $\gamma$  reguluje znaczenie nagród krótkoterminowych w stosunku do tych otrzymywanych w dłuższym horyzoncie czasowym. Im niższa jest wartość parametru, tym mniejsze znaczenie mają nagrody długoterminowe, wartość  $\gamma = 1$  natomiast traktuje wszystkie wzmocnienia jednakowo, niezależnie od momentu ich otrzymania.

Bardzo często przy prezentacji tej formuły parametr  $\alpha$  jest pomijany, z racji ustalenia jego wartości na 1. Taka postać rzeczy jest odpowiednia dla środowisk o charakterze stacjonarnym, czyli niezmiennym w czasie. W innym przypadku należy rozważyć zmniejszenie jego wartości, powodując u agenta częściowe zapominanie zdobywanej wiedzy.

#### 4.6.2. Algorytm Sarsa

Bardzo bliski algorytmowi Q-learning jest algorytm Sarsa (ang. *State, Action, Reward, State, Action*). Reguła aktualizacji funkcji  $Q$  ma tutaj następującą postać:

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha[r_t + \gamma Q(x_{t+1}, a_{t+1}) - Q(x_t, a_t)], \quad (4.45)$$

w której nie występuje maksymalizacja wartości akcji w następnym stanie, ale wartość akcji, jaka faktycznie w tym stanie zostanie wybrana do wykonania. Zwykle używa się do uczenia funkcji wartości akcji strategii, której algorytm się uczy, co zalicza algorytm Sarsa do kategorii *on-policy*. Szczagóły algorytmu przedstawia listing 4.3.

Listing 4.3: Algorytm Sarsa.

```

zainicjalizuj  $Q(x, a)$  dla wszystkich stanów  $x$  i akcji  $a$ ;
dla wszystkich kroków czasu  $t$  wykonaj:
  jeżeli  $t = 0$  to:
    obserwuj aktualny stan  $x_0$ ;
    wybierz akcję  $a_0$  na podstawie  $x_0$  i  $Q$ ;
  wykonaj akcję  $a_t$ ;
  obserwuj wzmocnienie  $r_t$  i następny stan  $x_{t+1}$ ;
  wybierz akcję  $a_{t+1}$  na podstawie  $x_{t+1}$  i  $Q$ ;
  wyznacz  $\Delta \leftarrow r_t + \gamma Q(x_{t+1}, a_{t+1}) - Q(x_t, a_t)$ ;
  aktualizuj  $Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha \Delta$ .

```

#### 4.6.3. Ślady aktywności - TD( $\lambda$ )

Metody różnic czasowych mogą być parametryzowane *współczynnikiem świeżości*  $\lambda \in [0, 1]$ . W standardowej wersji algorytmu ma on wartość zero, więc właściwym jest określenie go jako TD(0). Warto zatem rozważyć

$\lambda > 0$ , gdyż może to w wielu przypadkach przyspieszyć proces uczenia się. Takie podejście określa się mianem śladów aktywności (ang. *eligibility traces*). Zastosowanie tej metody jest szczególnie ważne w przypadkach, kiedy wzmocnienia są w pewien sposób odroczone, czyli nie mogą zostać przekazane w każdym kroku czasowym.

Zaktualizowana formuła 4.43 dla dowolnej wartości  $\lambda > 0$  prezentuje się następująco:

$$V(x_t) \leftarrow V(x_t) + \alpha[r_t + \gamma V(x_{t+1}) - V(x_t)]e_x(t). \quad (4.46)$$

Jak widać nasze równanie to teraz iloczyn poprzedniej wersji przez nowy parametr  $e_x(t)$  zwany śladem aktywności stanu  $x$  w kroku  $t$ , gdzie  $x$  jest stanem, dla którego w danym momencie następuje aktualizacja funkcji wartości. Główną zmianą jest bowiem fakt, że reguła jest stosowana nie dla pojedynczego stanu, ale dla wszystkich stanów w danym kroku czasowym.

#### 4.6.4. Q( $\lambda$ )

Znanych jest kilka implementacji śladów aktywności dla algorytmu Q-learning. Najpopularniejszym rozwiązaniem jest zaproponowane przez Watkinsa. Wykorzystuje ono podejście podobne jak w 4.46 z tą różnicą, że ścieżka śladów aktywności jest ucinana w momencie pierwszego wyboru akcji drogą eksploatacji - zakładając, że posługujemy się strategią  $\epsilon$ -zachłanną. Czasem jednak nie stosuje się tego wykluczenia, co może przynieść korzyści w przypadkach, kiedy uczeń posługuje się strategią zachłanną w znaczącej większości przypadków.

Rozszerzona o ślady aktywności wersja algorytmu Q-learning prezentuje się w pseudokodzie następująco:

Listing 4.4: Algorytm Q-learning lambda.

```

zainicjalizuj  $Q(x, a)$  oraz  $e(x, a) \leftarrow 0$  dla wszystkich  $x$  i  $a$ .
powtórz dla każdego epizodu:
  zainicjalizuj  $x, a$ ;
  powtórz dla każdego kroku w epizodzie:
    wykonaj  $a$ , obserwuj  $r, x'$ ;
    wybierz  $a'$  dla  $x'$  na podstawie  $Q$ ;
     $a^* \leftarrow \operatorname{argmax}_b Q(x', b)$ ;
     $\Delta \leftarrow r + \gamma Q(x', a^*) - Q(x, a)$ ;
     $e(x, a) \leftarrow e(x, a) + 1$ ;
    dla wszystkich  $x$  i  $a$ :
       $Q(x, a) \leftarrow Q(x, a) + \alpha \Delta e(x, a)$ ;
      jeśli  $a' = a^*$ , to  $e(x, a) \leftarrow \gamma \lambda e(x, a)$ ,
      w przeciwnym razie  $e(x, a) \leftarrow 0$ ;
     $x \leftarrow x'; a \leftarrow a'$ .
do momentu, aż  $x$  jest stanem terminalnym.

```

## 5. Projektowanie aplikacji

Rozdział poświęcony jest projektowi tworzonemu w ramach tej pracy. Początek przedstawia zastosowanie uczenia maszynowego w branży inwestycyjnej. Następnie zaprezentowany jest temat giełd zakładów sportowych, czyli domeny tworzonej aplikacji. W kolejnej części opisane są elementy składające się na projekt oraz przedstawione są szczegóły implementacyjne. Końcówka rozdziału zawiera testy zaprogramowanej symulacji.

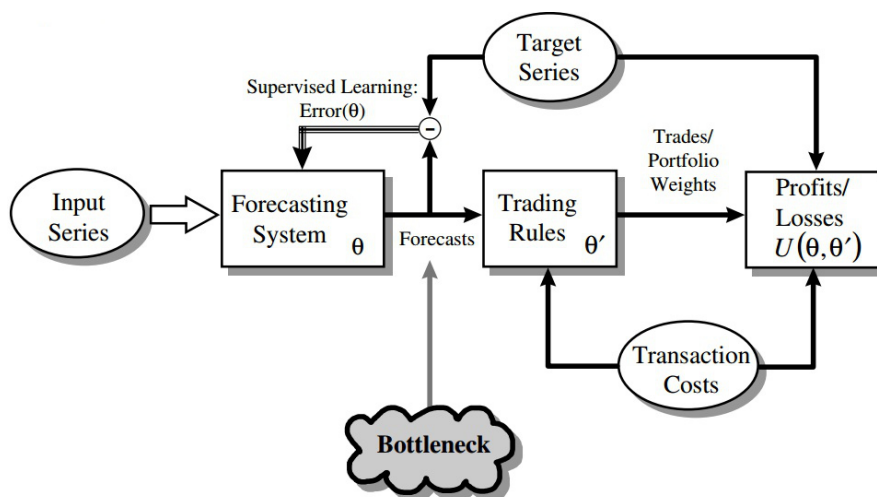
### 5.1. Zastosowanie maszynowego uczenia się w konstruowaniu strategii giełdowych

Sztuczna inteligencja mimo, że małymi krokami, to sukcesywnie wkracza do naszego codziennego życia. Jedną z dziedzin, w których naprawdę może ona rozwinąć skrzydła jest branża inwestycyjna, a konkretnie giełdy papierów wartościowych i inne podobne jej rynki finansowe. Odpowiednie jej zastosowanie pomaga w przeprowadzaniu tzw. spekulacji giełdowych, czyli przewidywania zmian cenowych, co ma prowadzić do skutecznego inwestowania kapitału.

Jest to idealne środowisko do zastosowania elementów sztucznej inteligencji. W tej chwili jest to obszar w pełni skomputeryzowany, a dostęp do danych giełdowych w formie cyfrowej jest powszechny, a w niektórych przypadkach znacząco ułatwiony. Wiele podmiotów oferuje bowiem dostęp do swoich danych oraz usług poprzez udostępnianie własnego API (ang. *Application Programming Interface*), które upraszcza tworzenie własnych narzędzi.

Te i inne czynniki powodują, że obecnie około 84% wszystkich transakcji giełdowych jest przeprowadzanych bez czynnego udziału człowieka. W 1960 roku średni czas przetrzymywania zakupionej akcji wynosił 4 lata. W roku 2000 wartość ta spadła do 8 miesięcy. Według statystyk z czerwca 2014 czas ten to 20 sekund, a obecnie już poniżej 10 [15].

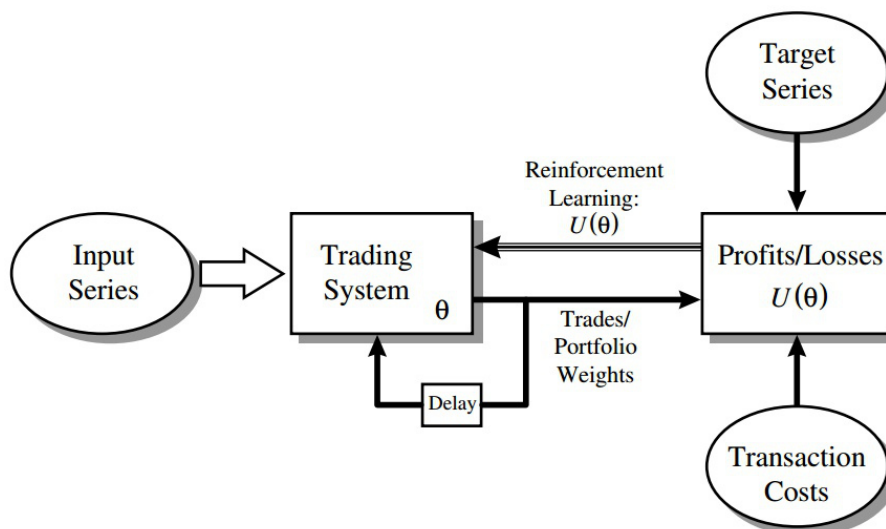
Istnieją różne podejścia w budowaniu tego typu systemów wykorzystując uczenie maszynowe. Tradycyjnym rozwiązaniem jest przewidywanie przyszłej ceny zasobu na podstawie wartości obserwowanych w przeszłości. Według tak stworzonych prognoz generowane są sygnały kupna, bądź sprzedaży, które potem przekazywane są do komponentu odpowiedzialnego za przeprowadzanie transakcji giełdowych. W publikacji [16] wymienia się kilka wad takiego rozwiązania, m.in. brak możliwości uwzględnienia kosztów przeprowadzanych transakcji. Główną trudnością w tym podejściu jest sformułowanie prognoz przy uwzględnieniu wszystkich istotnych czynników wpływających na wysokość kursu. Model tego rozwiązania przedstawiono na ilustracji 5.1.



Rysunek 5.1: Model systemu wykorzystującego naukę przez prognozowanie.

Źródło: [16]

Wyda się, że pozornie prostszym w implementacji oraz skuteczniejszym rozwiązaniem jest podejście do problemu wykorzystując opisywane w pracy uczenie ze wzmocnieniem. Eliminuje to skomplikowany proces przewidywania przyszłych cen i powoduje, że zachowanie agenta oraz jego skuteczność zależą wyłącznie od wykonywanych przez niego transakcji. Znika zatem wymaganie uwzględnienia wprost wszystkich czynników, które są w tym podejściu reprezentowane przy pomocy wzmocnień otrzymywanych przez agenta. Schemat 5.2 przedstawia ogólną architekturę takiego systemu.



Rysunek 5.2: Model systemu wykorzystującego uczenie przez wzmocnienie.

Źródło: [16]

Korzystając z tych wniosków autor postanowił stworzyć implementację opisywanego systemu uwzględniającą generowanie możliwe trafnych sygna-

łów kupna oraz sprzedaży na podstawie doświadczenia zdobywanego podczas gry. Aplikacja nie obejmuje mechanizmu przeprowadzania rzeczywistych transakcji, natomiast symuluje go stosując pewne uproszczenie. Założono w nim, że każdorazowy sygnał spowoduje automatyczny zakup, co w rzeczywistych warunkach nie jest zapewnione.

## 5.2. Giełdy zakładów sportowych

Dane, z których korzysta aplikacja, to rzeczywiste dane pochodzące z tzw. giełdy zakładów sportowych. Giełdę taką można określić jako podmiot zajmującą się przyjmowaniem zakładów pieniężnych związanych z różnymi dyscyplinami sportowymi, czyli tzw. bukmacher. W przeciwieństwie jednak do tradycyjnych jednostek stronami handlu nie jest giełda i gracz, ale sami gracze. Giełda jest jedynie pośrednikiem i udostępnia odpowiednie mechanizmy do przeprowadzania transakcji. Mechanizm ten przypomina uproszczoną formę znaną np. z giełd papierów wartościowych.

Zakłady mogą być składane za danym rezultatem, bądź przeciwko, co w teorii prowadzi do tego, że można je odprzedawać. Stwarza to możliwość spekulacji giełdowej i czerpanie zysków bez względu na końcowy rezultat wydarzenia sportowego.

Przed opisem mechanizmów giełdowych warto zdefiniować kilka mniej lub bardziej formalnych pojęć:

- **wydarzenie:** Dowolne wydarzenie sportowe, w ramach którego można zakupić zakład, np. mecz piłkarski, walka bokserska itp.
- **rynek:** W ogólnym rozumieniu jest to obszar giełdy, w ramach którego gracze handlują na ściśle określonych zasadach. W ramach każdego z wydarzeń na giełdzie udostępniony jest szereg rodzajów rynków.
- **opcja:** Rezultat danego wydarzenia możliwy do obstawienia, spośród kilku na danym rynku. Termin *opcja* będzie stosowane wymiennie z określeniem *rezultat*.
- **kierunek zakładu:** Typ zakładu jaki można zakupić na danej opcji. Możliwe jest zagranie za daną opcją lub jej przeciwieństwem (równoznaczne z obstawieniem każdego innego rezultatu). Zakład za określany będzie pojęciem *back*, a przeciw jako *lay*.

Przykłady dla niektórych wprowadzonych pojęć przedstawia tabela 5.1.

Tabela 5.1: Przykłady dla wprowadzonych pojęć dla meczu piłkarskiego.

Wydarzenie	Rynki	Opcje
Real Madrid vs FC Barcelona	Kurs meczowy	wygra Real Madrid
		wygra FC Barcelona
		mecz zakończy się remisem
	Poniżej/powyżej 2.5 gola	padnie poniżej 2.5 goli
padnie powyżej 2.5 goli		



### 5.2.1. Handel na giełdzie

Każdy rynek posiada z góry określoną liczbę możliwych rezultatów. Cena obstawienia jednego z nich jest równa odwrotności prawdopodobieństwa jego zajścia. Ich suma dla wszystkich rezultatów wynosi więc w każdym momencie w przybliżeniu 1. Jak zostało wspomniane możliwe jest zagranie za danym rezultatem (wygrana w przypadku jego wystąpienia) lub przeciwko (wygrana w przypadku każdego innego rezultatu). Gra polega na wystawieniu oferty na danej opcji za określoną stawkę z odpowiednim kierunkiem. Oferta pozostaje na rynku aż do przyjęcia jej przez innego gracza lub do momentu zamknięcia rynku bądź wycofania jej przez wystawiającego. Oferta może zostać przyjęta przez innego uczestnika gry, jeśli myśli on dokładnie odwrotnie i wystawia on zakład po tej samej cenie, ale w odwrotnym do kierunku naszej oferty.

Aby lepiej zrozumieć opisany mechanizm, zaprezentowany zostanie przykład opisanej transakcji zaczerpnięty z artykułu . Gracz 1 przewiduje, że pewne wydarzenie sportowe zakończy się rezultatem  $\omega$ . Stawia on zakład typu *back* za stawkę  $\mu$ . Aby to uczynić akceptuje aktualnie najlepszą oczekującą ofertę *lay*, bądź proponuje swoją cenę  $\rho$ . Gracz 2, który uważa, że wydarzenie nie zakończy się rezultatem  $\omega$  może przyjąć ofertę gracza 1, bądź wystawić własną. W przypadku przyjęcia istniejącej oferty gracz 2 wystawia zakład typu *lay* za cenę  $\rho$ . Cena tego zakładu to wtedy  $\rho - 1$  razy liczba jednostek, za którą gra. Po zakończeniu wydarzenia, którego dotyczy rynek zyski obu graczy będą wyglądać następująco:

$$p(\text{gracz 1}) = \begin{cases} \mu(\rho - 1) & \text{jeśli } \omega \text{ to rezultat wydarzenia,} \\ -\mu & \text{w przeciwnym wypadku.} \end{cases} \quad (5.1)$$

$$p(\text{gracz 2}) = \begin{cases} -\mu(\rho - 1) & \text{jeśli } \omega \text{ to rezultat wydarzenia,} \\ \mu & \text{w przeciwnym wypadku.} \end{cases} \quad (5.2)$$

Istnieje możliwość odsprzedaży zakupionego zasobu po korzystnej (lub też nie) cenie. W tym celu należy wystawić ponownie ofertę na tej samej opcji w przeciwnym kierunku. Od różnicy kursów obu transakcji będzie zależeć ewentualny zysk lub strata. Ich wartości możliwe są do wyliczania za pomocą równania

$$p = \begin{cases} \mu(\rho_1 - \rho_2) & \text{jeśli pierwszy zakład to } \textit{back} \text{ oraz } \omega \text{ to końcowy rezultat,} \\ \mu(\rho_2 - \rho_1) & \text{jeśli pierwszy zakład to } \textit{lay} \text{ oraz } \omega \text{ to nie jest końcowy rezultat,} \\ 0 & \text{w przeciwnym wypadku.} \end{cases} \quad (5.3)$$

gdzie  $p$  oznacza profit (stratę jeśli wartość ujemna),  $\mu$  stawkę, czyli sumę zainwestowanych jednostek pieniężnych w zakład,  $\rho_1$  to cena zakupu zakładu, a  $\rho_2$  to cena jego odsprzedaży.

Niemniej jednak, jeśli wynik wydarzenia będzie różny od tego, który był przedmiotem transakcji, gracz nic nie zyska, ale też nic nie straci. Pożądanym zabiegiem w tym przypadku jest podział zysku lub straty między wszystkie

opcje. Profit lub strata się zmniejszy, ale za to nie będzie zależny od końcowego rezultatu wydarzenia.

Aby tego dokonać należy stawkę sprzedaży powiększyć dodatkowo o niewielką jej część wyliczając zgodnie z wzorem 5.4.

$$\mu_2 = \begin{cases} \mu \frac{\rho_1}{\rho_2} & \text{jeśli pierwszy zakład to } \textit{back}, \\ \mu \frac{\rho_2}{\rho_1} & \text{jeśli pierwszy zakład to } \textit{lay}. \end{cases} \quad (5.4)$$

Wartość profitu po takiej operacji można wyliczyć przy pomocy poniższych równań.

$$p = \begin{cases} \frac{\mu(\rho_1 - \rho_2)}{\rho_2} & \text{jeśli pierwszy zakład to } \textit{back}, \\ \frac{\mu(\rho_2 - \rho_1)}{\rho_1} & \text{jeśli pierwszy zakład to } \textit{lay}. \end{cases} \quad (5.5)$$

Jak można zaobserwować, różnice w cenach podejmowanych zakładów oznaczają profit. Skuteczność w prognozowaniu tendencji zmian kursu jest kluczowa w czerpaniu regularnych zysków na tego typu rynkach.

Opisane procesy zostały przedstawione tylko w celu informacyjnym. Tworzona aplikacja nie obejmuje mechanizmów przeprowadzania transakcji na giełdzie, umożliwia jedynie symulację takiej gry na rynkach archiwalnych. Znając przedstawione mechanizmy giełdowe oraz wykorzystując stworzony moduł do uczenia została zaprogramowana symulacja samodzielnego gracza-bota zdolnego do podwyższania skuteczności swojej gry na podstawie zdobywanego doświadczenia.

### 5.3. Charakterystyka danych giełdowych

Posiadane dane dotyczą ponad 400 rynków typu *Kurs meczowy* dla meczów piłkarskich rozgrywanych w przeciągu miesięcy marzec-czerwiec w popularnych ligach europejskich. Dane te zawierają między innymi nazwę wydarzenia sportowego, datę jego rozpoczęcia czy sumę przetransferowanych na rynku pieniędzy. Najważniejszym ich elementem z punktu widzenia rozwiązywanego problemu jest lista zmian kursu dla wszystkich opcji w ramach rynku. Rozkład tych zmian pochodzi z okresu do pięciu dni wstecz od rozpoczęcia meczu.

Dane przechowywane są w formacie JSON (JavaScript Object Notation), często wykorzystywanym do reprezentacji danych. Jego strukturę stanowią obiekty złożone z par klucz-wartość. Format pozwala tworzyć struktury zagnieźdzone składające się z wielu obiektów, a także list. Jego format przypomina literały słowników oraz list w języku Python. Każdy rynek reprezentowany jest przez pojedynczy obiekt JSON. Przykładową strukturę takiego obiektu przedstawia listing 5.1. Opis pól obiektu rynkowego został zawarty w tabeli 5.2.

Dane archiwizowane są w bazie danych opartej na systemie MongoDB, który pozwala przechowywać je w postaci obiektów JSON. Alternatywnym sposobem ich zapisu są pliki tekstowe.

Listing 5.1: Przykładowy obiekt rynku.

```

{
  "_id" : "1.118452104",
  "marketStartTime" : ISODate("2015-05-05T20:45:00.000Z"),
  "competitionName" : "UEFA Champions League",
  "eventName" : "Juventus v Real Madrid",
  "totalMatched" : 4099833.45,
  "marketName" : "Match Odds",
  "collectStartTime" : ISODate("2015-04-30T21:43:49.152Z"),
  "runners" : [
    {
      "runnerName" : "Juventus",
      "runnerId" : "2423",
      "oddChanges" : [
        {
          "collectDelay" : 0.00,
          "oddLink" : 3.25
        },
        {
          "collectDelay" : 623.60,
          "oddLink" : 3.15
        },
        {
          "collectDelay" : 3366.69,
          "oddLink" : 3.20
        }
      ]
    },
    {
      "runnerName" : "Real Madrid",
      "runnerId" : "2426",
      "oddChanges" : [
        {
          "collectDelay" : 0.00,
          "oddLink" : 2.58
        },
        {
          "collectDelay" : 451.60,
          "oddLink" : 2.60
        },
        {
          "collectDelay" : 30959.29,
          "oddLink" : 2.62
        }
      ]
    },
    {
      "runnerName" : "The Draw",
      "runnerId" : "58805",
      "oddChanges" : [
        {
          "collectDelay" : 0.00,
          "oddLink" : 3.35
        },
        {
          "collectDelay" : 2411.50,
          "oddLink" : 3.30
        },
        {
          "collectDelay" : 2432.60,
          "oddLink" : 3.35
        }
      ]
    }
  ]
}

```

Tabela 5.2: Opis pól w strukturze obiektu reprezentującego rynek.

Nazwa pola	Opis przechowywanej wartości
_id	Identyfikator rynku w systemie giełdowym.
marketStartTime	Czas rozpoczęcia wydarzenia, którego dotyczy rynek.
competitionName	Nazwa rozgrywek, w ramach których przeprowadzane jest wydarzenie sportowe, którego dotyczy rynek.
eventName	Nazwa wydarzenia sportowego.
totalMatched	Łączna kwota przetransferowanych na rynku pieniędzy (wartość podana w walucie Euro).
marketName	Typ rynku.
collectStartTime	Czas rozpoczęcia zbierania danych na temat rynku.
runners	Lista zawierająca informacje na temat wszystkich opcji na rynku.
runnerName	Nazwa opcji.
runnerId	Identyfikator opcji w systemie giełdowym.
oddChanges	Lista obiektów zawierających informacje o zmianach kursu opcji. Każdy obiekt dotyczy jednej zmiany.
collectDelay	Różnica momentu kolejnej zmiany kursu i czasu, w którym rozpoczęto zbieranie informacji na temat rynku (pole collectStartTime) podana w sekundach.
oddLink	Cena opcji po zmianie.

## 5.4. Elementy projektu

Projekt aplikacji został podzielony na trzy moduły: ml\_algos, ml\_simulation, ml\_utils.

### 5.4.1. Moduł z implementacjami algorytmów uczenia się

W module ml\_algos została zawarta implementacja omawianych w tej pracy algorytmów różnic czasowych Q-learning i Sarsa obejmująca mechanizm śladów aktywności. Celowo została oddzielona od kodu symulacji giełdowej, aby mogła być w prosty sposób zastosowana do innych problemów, nie tylko tych opisywanych w tej pracy. Bardziej szczegółowy opis korzystania z tego modułu zawarto w sekcji 5.6.

Implementacja powstała na bazie algorytmów zawartych w książce [9]. W obecnej formie może ona być użyta do nauki w środowiskach o charakterze stochastycznym, dla których dodatkowo istnieje trudność w zdefiniowaniu modelu. Dzięki zastosowaniu śladów aktywności algorytmy dobrze spisują się w przypadkach, gdy konsekwencje podejmowanych decyzji nie są możliwe do oceny w każdym kroku nauki.

#### 5.4.2. Moduł narzędzi pomocniczych

Moduł `ml_utils` zawiera wszelkie narzędzia pomocnicze, z których korzystają pozostałe części aplikacji. Dotyczą one głównie wczytywania danych używanych w symulacji oraz ich przetwarzania i wizualizacji. Elementy modułu to `common_utils`, `data_loading`, `data_visualizer`, `graph_generator`, `message`, `simulation_data_generator`, `simulation_utils`.

Skrypt `data_loading` udostępnia dwie klasy pozwalające na wczytywanie danych z pliku JSON bądź bazy danych MongoDB. Dodatkowo mieści się tam klasa odpowiedzialna za aktualizację lokalnej bazy danych o dane znajdujące się w bazie zdalnej, w której gromadzone są wszystkie dane udostępniane przez giełdę i wykorzystywane w aplikacji.

Skrypt `data_visualizer` to aplikacja okienkowa, której zadaniem jest wizualizacja danych na wykresach. Została stworzona w celu weryfikacji poprawności zbieranych danych oraz obrazowania zmienności kursu na rynkach dla ułatwienia pracy nad tworzoną symulacją. Szczegółowy opis aplikacji znajduje się w sekcji 5.5.

W skryptach `common_utils` oraz `simulation_utils` zgrupowano szereg metod statycznych kolejno ogólnych, dotyczących zapisu do pliku czy przetwarzania tekstów komunikatów oraz tych związanych z symulacją, które pomagają głównie w przetwarzaniu danych.

Kolejny skrypt w tym module to `simulation_data_generator`, który służy do generowania sztucznych danych rynkowych o określonym modelu. Tego typu uproszczone dane są przydatne do analizy symulacji i dopasowywania parametrów algorytmu nauki. Opis jego uruchamiania zawarty jest w sekcji A.3.

Skrypt `graph_generator` przejmuje całkowicie odpowiedzialność generowania wykresów, a `message` zawiera stałe tekstowe wykorzystywane w projekcie.

#### 5.4.3. Moduł symulacji

Moduł `ml_simulation` jest odpowiedzialny za przeprowadzenie procesu symulacji. Nie obejmuje on działań związanych z uczeniem się, do tego celu wykorzystuje osobny moduł `ml_algos`. Zadaniem tego modułu jest dostarczenie danych do symulacji, koordynacja symulacji o zdefiniowanych parametrach, następnie udostępnienie wyniku w postaci tekstowej, a w niektórych przypadkach także wykresu. Skrypty wchodzące w skład modułu to: `merged_simulation`, `simulation`, `simulation_launcher`, `simulation_result`.

Główny skrypt tego modułu to `simulation`, w którym zawarte są klasy `SingleMarketSimulation` oraz `AllMarketsSimulation` odpowiedzialne kolejno za przeprowadzenie symulacji gry na jednym rynku oraz za koordynację tego procesu dla wszystkich jednostek rynkowych.

Do uruchamiania symulacji służy skrypt `simulation_launcher`. Pełna instrukcja dotycząca konfiguracji oraz użytkowania znajduje się w sekcji A.5.

Skrypt `simulation_result` jest odpowiedzialny za przetworzenie rezultatu symulacji oraz wyświetlenie jej wyniku. Ponieważ algorytmy zawierają pewien czynnik losowy zaistniała potrzeba uruchamiania ich wielokrotnie oraz uśredniania rezultatów, w celu oceny efektywności algorytmów. Za zebranie wyników wielu symulacji oraz ich uśrednienie odpowiada skrypt `merged_simulation`.

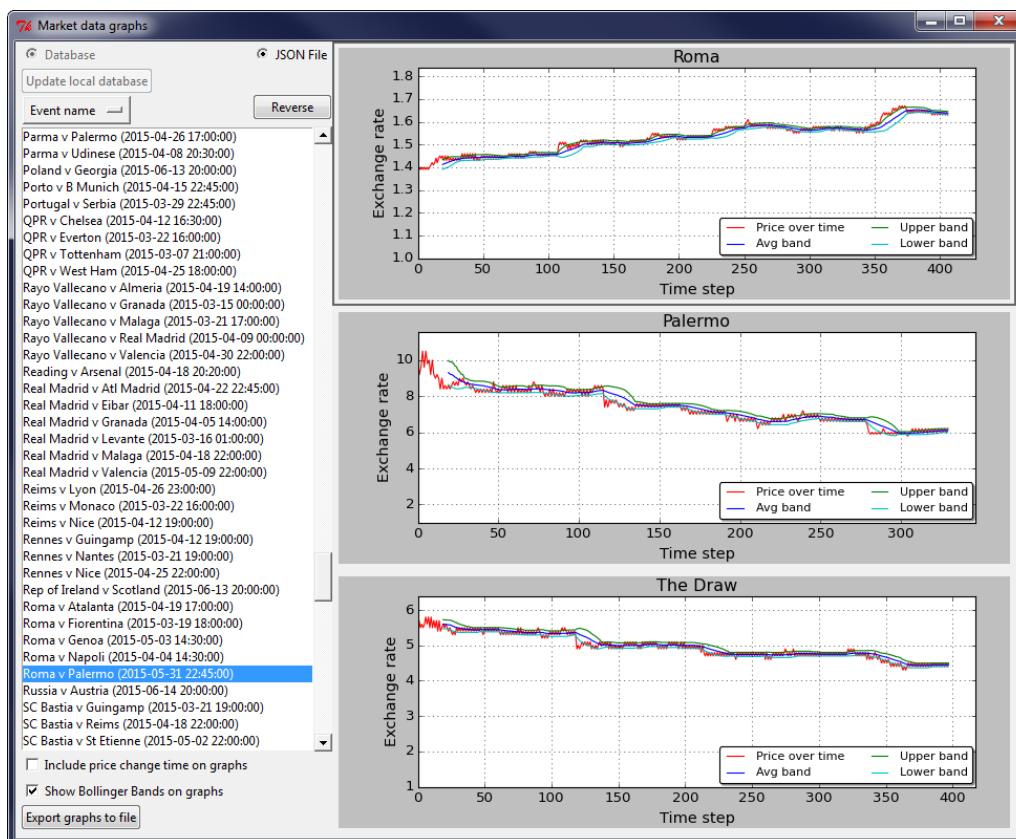
## 5.5. Wizualizacja danych

Pierwszą z aplikacji stworzonych w ramach niniejszej pracy jest narzędzie pomocnicze pozwalające na przedstawienie danych rynkowych na wykresach. Aplikacja udostępnia interfejs graficzny, który został zbudowany przy pomocy standardowej biblioteki języka Python - Tkinter, natomiast do generowania wykresów wykorzystano moduł Matplotlib.

Interfejs aplikacji dzieli się na dwie części. Po lewej stronie umieszczona jest lista wydarzeń sportowych związanych z dostępnymi danymi rynkowymi oraz kontrolki obsługujące funkcjonalność aplikacji. Poniżej ich lista:

- Kontrolki typu radio do wyboru typu źródła danych. W opublikowanej wersji jedynym dostępnym rodzajem jest plik JSON.
- Przycisk aktualizacji lokalnej bazy danych z bazy zdalnej, do której bezpośrednio spływają dane giełdowe. Opcja jest domyślnie zablokowana.
- Lista wyboru właściwości, po której ma być sortowana lista rynków oraz przycisk odwracania kolejności.
- Pole checkbox, którego włączenie powoduje uwzględnienie czasu zmiany ceny na wykresach.
- Pole checkbox, którego włączenie powoduje wyświetlenie wskaźnika wstęg Bollingera na wykresach.
- Przycisk do generowania obecnie wyświetlanych wykresów do plików graficznych.

Po wybraniu nazwy wydarzenia, którego dotyczy rynek w prawym panelu wyświetlają się wykresy dotyczące danych ze wszystkich opcji. Instrukcja uruchamiania aplikacji została zawarta w sekcji A.4. Rysunek 5.3 przedstawia okno aplikacji.



Rysunek 5.3: Okno aplikacji do wizualizacji danych giełdowych na wykresach.

Źródło: opracowanie własne

## 5.6. Korzystanie z implementacji algorytmu nauki

Przygotowana symulacja wykorzystuje implementacje dwóch algorytmów z rodziny metod różnic czasowych - Q-learning i Sarsa. Całość obsługi procesu nauki znajduje się w skrypcie `td_learning.py`. Zawarta jest w nim definicja klasy głównej `TDAgent`. Aby przeprowadzić proces nauki należy utworzyć obiekt tej klasy za pomocą udostępnionego konstruktora, a następnie wywoływać na nim zdefiniowane metody według scenariusza, który zostanie opisany w dalszej części tej sekcji. Przykładową konstrukcję obiektu klasy `TDAgent` przedstawia listing 5.2:

Listing 5.2: Przykładowa konstrukcja obiektu agenta.

```

1 import numpy
2 import ml_project.ml_algos.td_learning as td
3
4 q = numpy.array([
5     [0.0, 0.0, 0.0],
6     [0.0, 0.0, 0.0],
7     [0.0, 0.0, 0.0],
8     [0.0, 0.0, 0.0]
9 ])
10
11 e_tracing = td.ETrace(tracing_type=td.ETracingMethod.WATKINS,
12                      factor=0.4)
13
14 qa = td.TDAgent(q, learning_rate=0.4,
15                discount_factor=0.9,
16                chance_to_explore=0.05,
17                e_tracing=e_tracing,
18                alg=td.TDAlgorithm.Q_LEARNING,
19                terminal_states=[])

```

Jak widać konstruktor przyjmuje szereg argumentów. Pierwszym z nich jest macierz wartości dla par stan-akcja, która technicznie może być listą dwuwymiarową lub obiektem typu `numpy.ndarray` modułu `NumPy`. Każdy z wierszy oznacza stan w jakim może znaleźć się agent w trakcie nauki. Kolumny natomiast oznaczają zakres akcji jakie może wykonać agent. Z punktu widzenia modułu uczącego nie ma znaczenia jakie są to stany oraz akcje. Są one traktowane kolejno jako liczby z zakresu  $[0, m - 1]$  oraz  $[0, n - 1]$ , gdzie  $m$  to liczba wierszy, a  $n$  to liczba kolumn we wspomnianej macierzy. Liczby w poszczególnych komórkach oznaczają wartość akcji podejmowanej w określonym stanie. Jeśli w którymkolwiek ze stanów nie ma możliwości podjęcia pewnej akcji, to wartość dla tej pary ma być ustalona na stałą `numpy.NaN`. Standardowo macierz powinna być inicjalizowana zerami, które w procesie nauki będą aktualizowane nowymi wartościami.

Kolejne parametry konstruktora to:

- `learning_rate` - wartość  $\alpha$  ze wzorów 4.44 oraz 4.45,
- `discount_factor` - wartość  $\gamma$  ze wzorów 4.44 oraz 4.45,
- `chance_to_explore` - wartość  $\varepsilon$  w strategii  $\varepsilon$ -zachłannej.

Argument `e_tracing` odpowiada za metodę śladów aktywności. Jest on opcjonalny i może przyjmować wartość `None`, w przeciwnym wypadku należy przekazać obiekt klasy `ETrace`. W linii 11 powyższego listingu pokazano przykładową konstrukcję. Klasa pełni rolę opakowania dla dwóch wartości - metody śladów aktywności oraz współczynnika świeżości  $\lambda$ . Pierwszy parametr podajemy jako liczbę całkowitą: 0 - metoda Watkinsa, 1 - metoda naiwna. W celach pomocniczych stworzono klasę `ETracingMethod`, która zawiera stałe z tymi wartościami. Drugi parametr to liczba rzeczywista z przedziału  $[0, 1]$ , która ustala wspomniany współczynnik  $\lambda$ .

Kolejny parametr konstruktora określa algorytm nauki. Przyjmuje wartość 0 dla Q-learning lub 1 dla Sarsa. Obie stałe są określone w klasie `TDAlgorithm`.



Po stworzeniu obiektu typu `TDAgent` należy zdefiniować w następujący sposób stan początkowy agenta:

```
td_agent.set_initial_state(0)
```

Na tym kończy się etap inicjalizacji agenta. W tym momencie możliwe jest rozpoczęcie iteracyjnego procesu nauki. Na listingu 5.6 przedstawiono proces składający się z 10 kroków.

```
1 for step in xrange(10):
2     chosen_action = td_agent.choose_action_to_perform()
3
4     reward = generate_reward(chosen_action)
5     td_agent.provide_reward_for_last_action(reward)
6
7     next_state = determine_next_state(chosen_action)
8     td_agent.go_to_next_state(next_state)
```

Na początku każdej iteracji należy wywołać metodę `choose_action_to_perform`, która wybiera akcję do podjęcia w tym kroku czasowym oraz zwraca liczbę całkowitą oznaczającą identyfikator akcji. Następnym krokiem jest wykonanie tej akcji oraz odebranie wzmocnienia ze środowiska. W listingu dzieje się to w metodzie `generate_reward`, za której definicję odpowiada użytkownik opisywanego modułu. Dla przykładu może się w niej znaleźć wywołanie polecenia zrobienia kroku przez robota i wygenerowanie nagrody na podstawie tego, na jaką odległość po jego wykonaniu przybliżył się do określonego celu. W przypadku tworzonej symulacji nagroda jest generowana tylko po wykonaniu akcji zamknięcia zakładu i jest równa wartości otrzymanego zysku bądź straty.

Otrzymane wzmocnienie przekazywane jest do agenta za pośrednictwem metody `provide_reward_for_last_action`. Oprócz wzmocnienia należy ustalić nowy stan, w którym znajdzie się agent po wykonaniu akcji. W formie liczby całkowitej przekazujemy go do agenta metodą `go_to_next_state`.

Jak widać w procesie nauki biblioteka bierze na siebie tylko część odpowiedzialności. Sprawą naturalną jest, że użytkownik z niej korzystający jest zobowiązany do zaprogramowania rzeczywistej interakcji ze środowiskiem i odbioru wzmocnień. W dodatku implementacja nie przewiduje definicji modelu środowiska, czego powodem jest wymóg ustalania każdorazowo nowych stanów agenta po wykonywaniu akcji.

## 5.7. Projektowanie symulacji

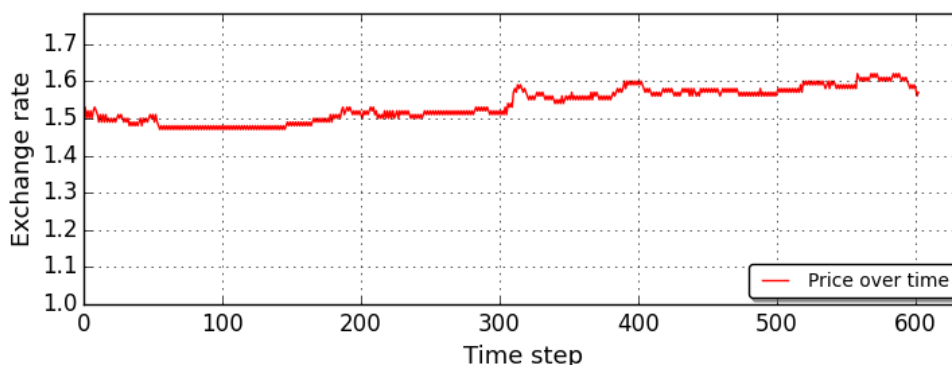
### 5.7.1. Definicja stanów

W procesie definiowania stanów dla algorytmu nauki należy uwzględnić zarówno stany gracza uzależnione od tego, czy posiada on zakład oraz stany w jakich może znaleźć się środowisko, w tym przypadku rynek giełdowy, a ściślej cena danej opcji. Po ich ustaleniu rzeczywisty zbiór stanów ma postać iloczynu kartezjańskiego obydwu zbiorów.

W przypadku agenta zestaw stanów zdefiniowano następująco:

- brak pozycji,
- pozycja przewidująca spadek kursu,
- pozycja przewidująca wzrost kursu.

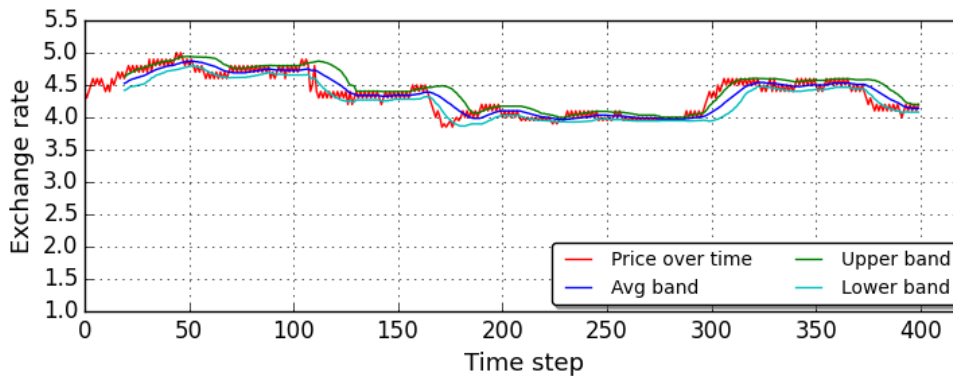
Kwestia definicji stanów środowiska jest bardziej złożona. Naturalnym podejściem wydaje się być zdefiniowanie zakresu stanów na bazie cen, które przyjmowała dana opcja od chwili otwarcia rynku. Przykładowo dla opcji, której cena na przestrzeni czasu prezentuje się jak na wykresie 5.4, przedział ten obejmowałby zakres  $[1.48, 1.62]$ , czyli wartości od minimum do maksimum ceny. Daje to łącznie 15 stanów środowiska, więc tworząc iloczyn kartezjański zbioru stanów agenta oraz zbioru stanów środowiska otrzymujemy łącznie 45 stanów. Liczba ta z pewnością wydaje się być za wysoka, mając na uwadze fakt, że agent dla optymalnego działania musi przetestować akcje podejmowane w każdym ze stanów.



Rysunek 5.4: Demonstracja sekwencji zmian ceny jednej z opcji rynkowych na wykresie.

Źródło: opracowanie własne

W tym podejściu można dostrzec inne problemy jak np. zależność zbioru stanów od opcji, co rodzi np. trudność w porównywaniu rezultatów symulacji między rynkami. Dodatkowo częstym zabiegiem byłoby powiększanie zbioru stanów o nowe wartości w przypadku zmian ceny na niespotykane dotąd wartości. Przy każdorazowym dołożeniu stanu wartość akcji w nim podejmowanych byłaby zerowa, czyli agent nie posiadałby o niej żadnej wiedzy.



Rysunek 5.5: Wstęgi Bollingera na wykresie zmiany ceny.

Źródło: opracowanie własne

Warto w tym przypadku zastosować tzw. wskaźniki analizy technicznej, które powstały z myślą o opisywaniu stanów rynków finansowych. Jednym z nich są wstęgi Bollingera oparte na zmienności cen. Zakłada on, że cena porusza się w obrębie trzech wstęg:

- środkowej - będącej  $n$ -okresową średnią ruchomą,
- górnej - będącej  $k$ -krotnością  $n$ -okresowego odchylenia standardowego powyżej środkowej wstęgi,
- dolnej - będącej  $k$ -krotnością  $n$ -okresowego odchylenia standardowego poniżej środkowej wstęgi.

Na wykresie przedstawiono opisane wstęgi dla sekwencji cen pewnej opcji rynkowej. Stany rynku można określić na podstawie tego, w którym z przedziałów wyznaczonych przez wstęgi może znaleźć się cena. Zdefiniowane zostały 4 stany:

- cena powyżej górnej wstęgi,
- cena między górną i środkową wstęgą,
- cena między środkową i dolną wstęgą,
- cena poniżej dolnej wstęgi.

### 5.7.2. Definicja akcji

W przypadku definicji zbioru akcji możliwych do wykonania przez agenta warto zwrócić uwagę na akcję bierną, czyli brak interakcji z giełdą w danym kroku czasowym. Zdaniem autora warto rozróżnić tę akcję w zależności od tego czy gracz aktualnie posiada zakład.

Zdefiniowany następujący zestaw akcji:

- brak interakcji z giełdą nie posiadając zakładu (akcja bierna),
- kupno zakładu przewidując spadek ceny (zakład typu *back*),
- kupno zakładu przewidując wzrost ceny (zakład typu *lay*),
- odsprzedaż (zamknięcie) zakładu,
- utrzymywanie zakładu (akcja bierna).

Należy mieć na względzie fakt, że niektórych akcji nie można przeprowadzić w pewnych stanach. Dla przykładu nie można odsprzedać zakładu jeśli

wcześniej nie został on przez nas zakupiony lub kupić zakład posiadając już jeden.

### 5.7.3. Definicja macierzy wartości Q

Na podstawie zdefiniowanych wcześniej stanów oraz akcji można ustalić początkową postać macierzy Q, która zostanie przekazana do modułu uczącego. Każdemu z trzech stanów agenta należy przyporządkować cztero-elementowny zbiór stanów środowiska. Poniżej fragment tak uformowanej listy stanów:

- brak pozycji i cena powyżej górnej wstęgi,
- pozycja przewidująca spadek kursu i cena powyżej górnej wstęgi,
- pozycja przewidująca wzrost kursu i cena powyżej górnej wstęgi,
- brak pozycji i cena między górną i środkową wstęgą,
- pozycja przewidująca spadek kursu i cena między górną i środkową wstęgą itd.

Dokładając do tego zbiór akcji otrzymujemy macierz  $12 \times 5$  zaprezentowaną poniżej.

$$Q = \begin{pmatrix} 0.0 & 0.0001 & 0.0001 & NAN & NAN \\ NAN & NAN & NAN & 0.0 & 0.0 \\ NAN & NAN & NAN & 0.0 & 0.0 \\ 0.0 & 0.0001 & 0.0001 & NAN & NAN \\ NAN & NAN & NAN & 0.0 & 0.0 \\ NAN & NAN & NAN & 0.0 & 0.0 \\ 0.0 & 0.0001 & 0.0001 & NAN & NAN \\ NAN & NAN & NAN & 0.0 & 0.0 \\ NAN & NAN & NAN & 0.0 & 0.0 \\ 0.0 & 0.0001 & 0.0001 & NAN & NAN \\ NAN & NAN & NAN & 0.0 & 0.0 \\ NAN & NAN & NAN & 0.0 & 0.0 \end{pmatrix} \quad (5.6)$$

Należy zwrócić uwagę na kolumny 2. i 3., przechowujące wartości akcji kolejno: *zagraj na spadek*, *zagraj na wzrost* dla poszczególnych stanów. Niezerowa wartość ma zachęcić agenta do gry już od początku interakcji z giełdą. Zakładając, że gracz nie posiada żadnej wiedzy na temat danego rynku nie ma uzasadnienia dla jakiegokolwiek zwłoki przy rozpoczęciu procesu nauki. Liczba jest niewielka, aby nie wpływać na zachowanie agenta w dłuższym horyzoncie czasowym. Dla wartości akcji, których nie można wykonać w danym stanie została ustalona wartość NAN (Not a Number).

### 5.7.4. Proces nauki

Po przekazaniu macierzy rozpoczynamy iteracyjny proces nauki analogiczny do tego zaprezentowanego na listingu 5.6 dla każdej ceny z posiadanej sekwencji cen danej opcji. Za wzmocnienie uznajemy rezultat poczynionej transakcji, czyli otrzymany zysk, bądź poniesioną stratę. Wynosi ono dla

każdej akcji 0, prócz tej, odpowiedzialnej za zamknięcie zakładu (kolumna 4 macierzy Q).

Za wyliczenie zysku odpowiada metoda zaimplementowana na podstawie wzoru 5.5. Stawka, którą dysponuje gracz wynosi domyślnie 10 jednostek. Metoda uwzględnia koszty prowizji ponoszone tylko w przypadku zyskowych transakcji, wynoszące do 5% profitu.

Agent inicjalizowany jest ze stanem *brak pozycji* i w każdym kroku ustalany na podstawie wykonywanych przez agenta akcji. Stan środowiska jest ustalany w każdym kroku zgodnie z tym w jakim przedziale wyznaczonym ze pomocą wstęg Bollingera znajduje się aktualna cena opcji.

## 5.8. Testy symulacji

### 5.8.1. Znaczenie i dobór parametrów algorytmu

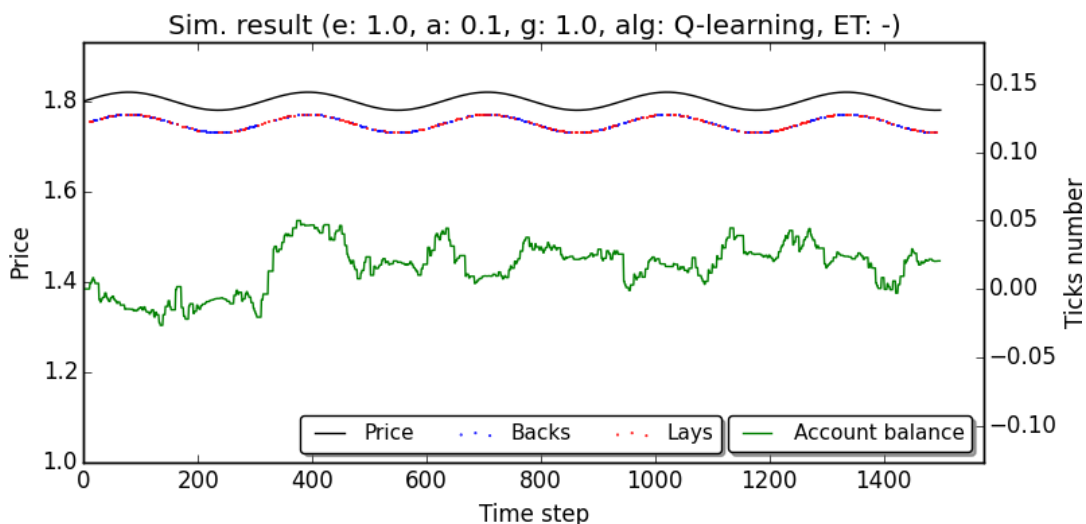
Na przykładzie wykonanych testów symulacji przedstawiona zostanie istota parametrów algorytmów z równań 4.44 oraz 4.45. Testy zostały wykonane z użyciem sztucznych danych o modelu sinusoidy z kilkoma okresami. W każdym przypadku zastosowano algorytm Q-learning bez użycia śladów aktywności. Niektóre charakterystyczne rezultaty symulacji zostały zaprezentowane na wykresach. Zawierają one informacje:

- o sekwencji zmian ceny bez uwzględnienia czasu zmiany - (czarna linia),
- sygnały kupna zakładu typu *back* (niebieskie punkty),
- sygnały kupna zakładu typu *lay* (czerwone punkty),
- poziom salda, poczynając od 0 (zielona linia).

#### 5.8.1.1. Prawdopodobieństwo wykonania akcji eksplorującej $\varepsilon$

Jako pierwszy zostanie omówiony parametr  $\varepsilon$ , od którego zależy jest stosowana przez algorytm strategia  $\varepsilon$ -zachłanna. Oznacza ona podejmowanie decyzji najkorzystniejszych w aktualnym kroku czasowym, robiąc co jakiś czas wyjątki wykonując tzw. akcje eksplorujące, które mają przyspieszyć powiększanie się zakresu przetestowanych akcji. Wspomniany parametr oznacza prawdopodobieństwo z jakim taka decyzja może zostać wybrana.

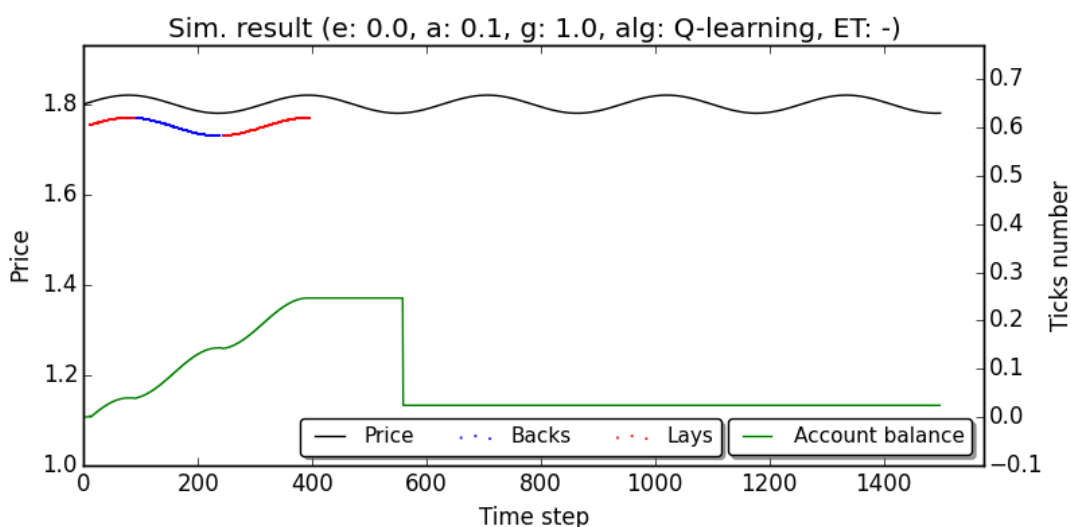
Ustalenie wartości tego parametru na wartość 1 oznacza w praktyce całkowicie losowy dobór akcji w każdym kroku. Obrazuje to wykres , na którym saldo konta gracza nie wykazuje żadnych tendencji w jedną bądź drugą stronę.



Rysunek 5.6: Rezultat symulacji wykorzystującej sztuczne dane z wartością parametru  $\epsilon$  ustawioną na 1.

Źródło: opracowanie własne

Parametr przyjmujący zaś wartość 0 skazuje algorytm na tendencyjność uzależnioną od decyzji podejmowanych w początkowej fazie nauki. W każdym kroku czasowym agent wybiera zawsze akcję z najwyższą wartością. Jak widać na wykresie 5.7 macierz wartości akcji wyglądała w pewnym momencie tak, że najbardziej opłacalną decyzją z punktu widzenia strategii okazało się zaprzestanie gry na giełdzie, co też agent uczynił patrząc na utrzymujący się poziom salda od okolic 580 kroku.



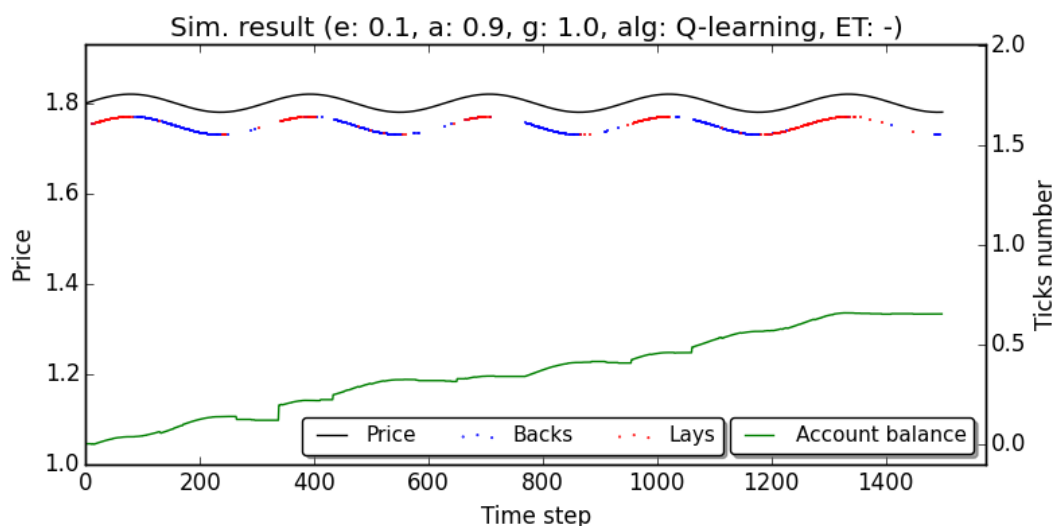
Rysunek 5.7: Rezultat symulacji wykorzystującej sztuczne dane z wartością parametru  $\epsilon$  ustawioną na 0.

Źródło: opracowanie własne

Niska wartość tego parametru gwarantuje dobre wyniki w początkowej fazie nauki, ale mniej optymalne w dłuższym horyzoncie czasowym, ponieważ ewaluacja akcji następuje wolniej w miarę jak zawęża się dobór akcji. Dla testowanego modelu za wartość bliską optymalnej można uznać 0.1.

### 5.8.1.2. Współczynnik uczenia się $\alpha$

Współczynnik uczenia się  $\alpha$  wpływa na to, jak szybko nowa wiedza będzie nadpisywać starsze informacje. Wartość 0 spowoduje brak jakiegokolwiek nauki, wartość 1, że agent będzie brał pod uwagę jedynie najbardziej aktualną wiedzę i taka też wartość będzie optymalna dla środowisk deterministycznych. Problemy o charakterze stochastycznym często wymagają niewielkiej wartości parametru. Dla rozważanej sekwencji danych za bliską optymalnej można uznać wartość 0.9, co w pewnym stopniu udowadnia wykres 5.8.

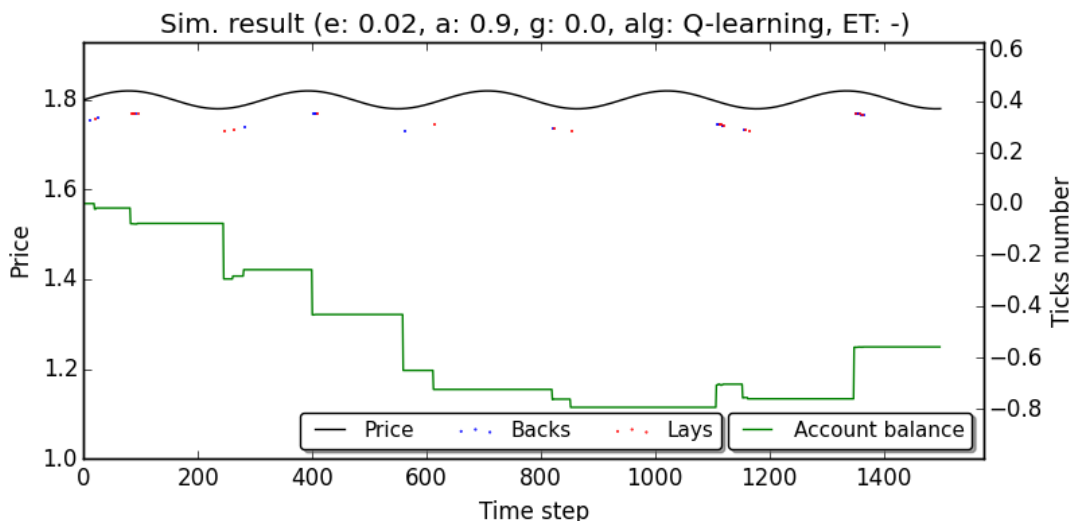


Rysunek 5.8: Rezultat symulacji wykorzystującej sztuczne dane z wartością parametru  $\alpha$  ustawioną na 0.9.

Źródło: opracowanie własne

### 5.8.1.3. Współczynnik dyskontowania nagród $\gamma$

Parametr  $\gamma$  określa stopień ważności przyszłych nagród. Wartość 1 powoduje, że wszystkie nagrody, niezależnie od momentu otrzymania będą traktowane jednakowo. Zerowa wartość przyczyni się do tego, że rozważane będą jedynie te akcje, które bezpośrednio prowadzą do otrzymywania nagród. Może mieć to niezbyt dobre konsekwencje, zwłaszcza w przypadkach, kiedy nagrody przynoszą tylko niektóre z akcji. Wykres 5.9 prezentuje ciekawy przypadek z tym związany.



Rysunek 5.9: Rezultat symulacji wykorzystującej sztuczne dane z wartością parametru  $\gamma$  ustawioną na 0.

Aby zrozumieć, co spowodowało taki obrót spraw, należy przyjrzeć się końcowej postaci macierzy wartości  $Q$ , którą prezentuje listing 5.3. Jak widać wartości niezerowe są obecne jedynie w kolumnie czwartej, przechowującej wartości akcji zamknięcia zakładu, która jako jedyna gwarantuje nagrody. Zgodnie z istotą współczynnika dyskontowania inne akcje nie były wartościowane w czasie działania algorytmu. Dlatego też akcje otwarcia zakładu w różnych kierunkach są podejmowana równie często, ponieważ mają w każdym momencie taką samą wartość. Fakt ten można stwierdzić na podstawie ilości sygnałów kupna - wystarczy porównać ilość niebieskich i czerwonych punktów. Nieco gorzej widać to na wykresie salda, gdzie momentami jego poziom wzrasta, ale prawie za każdym razem następuje to po relatywnie krótkim utrzymywaniu zakładu. Później jednak widać tendencję do nietrafnego wyboru między akcjami zamknięcia oraz utrzymania zakładu. Agent w przypadku wybrania akcji kupna zakładu, dla którego akcja zamknięcia posiada wartość ujemną ma zawsze do wyboru:

- zamknięcie zakładu ( $Q_a < 0$ ),
- utrzymywanie zakładu ( $Q_a = 0$ ).

Wybierana jest prawie zawsze akcja druga jako posiadająca wyższą wartość. Taki zakład jest zamykany tylko w dwóch przypadkach: wystąpienia polecenia wykonania akcji eksplorującej bądź zmiany stanu środowiska, tj. przedziału (wyznaczonego przez wstęgi Bollingera) w jakim znajduje się cena opcji. Zaistnienie tego drugiego można stwierdzić na podstawie kroków czasowych w jakich występują sygnały kupna. Skupiają się one okolicach lokalnych minimów i maksimów funkcji. Wtedy dla nienotowanego wcześniej stanu ceny giełdowej wartość akcji zamknięcia oraz utrzymania zakładu ma tę samo, zerową wartość.

Kiedy agent podejmuje zakład, dla którego zamknięcia zwykle przynosiły zysk (według wartości  $Q$ ), ma on następujące możliwości:

- zamknięcie zakładu ( $Q_a > 0$ ),



— utrzymywanie zakładu ( $Q_a < 0$ ).

Agent zawsze zamyka zakład, ponieważ według strategii jest to optymalne. Tym samym nigdy nie pozwala rosnać zyskom, co robi zawsze w przypadku zakupu potencjalnie mniej wartościowych zakładów.

Listing 5.3: Końcowa postać macierzy wartości Q dla symulacji na sztucznych danych uruchomionej z wartością parametru  $\gamma$  ustawioną na 0.

```

[[
  [ 0.    0.    0.    nan    nan ],
  [ nan  nan  nan -0.0097  0. ],
  [ nan  nan  nan  0.0006  0. ],
  [ 0.    0.    0.    nan    nan ],
  [ nan  nan  nan -0.0087  0. ],
  [ nan  nan  nan  0.0008  0. ],
  [ 0.    0.    0.    nan    nan ],
  [ nan  nan  nan -0.0002  0. ],
  [ nan  nan  nan -0.0097  0. ],
  [ 0.    0.    0.    nan    nan ],
  [ nan  nan  nan  0.0017  0. ],
  [ nan  nan  nan -0.0196  0. ]]

```

#### 5.8.1.4. Optymalny algorytm i parametry

Na koniec tej części testów skonfrontowano ze sobą oba algorytmy, czyli Q-learning i Sarsa również w wersjach ze śladami aktywności. Tabela prezentuje uśrednione wyniki z 10 symulacji przeprowadzonych dla każdego zestawu parametrów.

Tabela 5.3: Zestawienie rezultatów symulacji wykorzystującej sztuczne dane rynkowe.

Wersja algorytmu <sup>1</sup>	Ogólny zysk (j.) <sup>2</sup>	Zyskowość transakcji (%) <sup>3</sup>			Liczba transakcji	Zysk na transakcję (j.)
Q-learning	0.47	85.3	3.7	10.0	468.60	0.00088
Sarsa	0.96	74.1	2.4	23.5	332.70	0.00304
Q-learning ś. a.	0.91	83.4	4.3	12.3	421.50	0.00216
Sarsa ś. a.	0.44	71.3	2.8	25.9	428.60	0.00101

<sup>1</sup> Wartości parametrów:  $\epsilon$ : 0.1,  $\alpha$ : 0.9,  $\gamma$ : 1.0,  $\lambda$ : 0.4, typ ś. a.: Watkinsa.

<sup>2</sup> Uzyskany w przypadku gry stawką równą 10 jednostek.

<sup>3</sup> Udział transakcji przynoszących kolejno: zauważalny zysk, śladowy zysk lub stratę, zauważalną stratę.

Najważniejszymi parametrami są ogólny zysk oraz zysk na transakcję. W przypadku obu z nich algorytm Sarsa bez użycia śladów aktywności góruje nad resztą. Gracz z niego korzystający handluje rzadziej od reszty, przy czym ogólny zysk jest najwyższy. Swoistym paradoksem jest tutaj fakt, że również częściej podejmował on błędne decyzje w porównaniu do obu wersji algorytmu Q-learning.

### 5.8.2. Testy symulacji na rzeczywistych danych

Tabela 5.4 przedstawia wyniki symulacji dla różnych wersji algorytmów. Dla każdej z nich zaprezentowano uśrednione wyniki z 5 przeprowadzonych symulacji z użyciem wszystkich posiadanych danych rynkowych. Uwzględniono tylko te opcje, dla których liczba zaobserwowanych zmian ceny wynosiła przynajmniej 100, a średnia cena nie przekraczała wartości 2.7. Pierwsze kryterium miało na celu wyeliminowanie relatywnie mało popularnych rynków. Wymaganie co do średniej ceny wprowadzono dlatego, że wyższe kursy mogą generować nieproporcjonalnie wysokie zyski lub straty, co mogłoby w dużym stopniu zaciemnić rezultat symulacji. Z tej przyczyny gracz nie podjął gry jedynie na niewielkiej części. Wartości wszystkich parametrów dobrano na podstawie wykonywanych testów i są one bliskie optymalnym. Warto zwrócić uwagę na wartość parametru świeżości  $\alpha$ , która jest znacznie niższa od tej, która była optymalna w przypadku poprzedniego modelu danych. Wskazuje ona na znacznie bardziej losowy charakter zmienności ceny w przypadku rzeczywistych danych giełdowych.

Tabela 5.4: Zestawienie rezultatów symulacji gry na giełdzie zakładów sportowych wykorzystującej archiwalne dane.

Wersja algorytmu <sup>1</sup>	Ogólny zysk (j.) <sup>2</sup>	Zyskowność transakcji (%) <sup>3</sup>			Zyskowność rynków (%) <sup>4</sup>			Liczba transakcji na rynek <sup>5</sup>	Zysk na transakcję (j.)
Q-learning	2034.4	70.3	12.3	17.5	80.6	0.0	19.4	129.2	0.0391
Sarsa	1526.1	66.5	15.0	18.5	80.3	0.0	19.7	108.9	0.0348
Q-learning ś. a.	2377.7	70.9	13.7	15.4	87.7	0.0	12.3	141.4	0.0417
Sarsa ś. a.	1676.1	65.6	17.4	17.1	86.4	0.0	13.6	114.7	0.0362

<sup>1</sup> Wartości parametrów:  $\varepsilon$ : 0.05,  $\alpha$ : 0.1,  $\gamma$ : 1.0,  $\lambda$ : 0.4, typ ś. a.: Watkinsa.

<sup>2</sup> Uzyskany w przypadku gry stawką równą 10 jednostek.

<sup>3</sup> Udział transakcji przynoszących kolejno: zauważalny zysk, śladowy zysk lub stratę, zauważalną stratę.

<sup>4</sup> Udział rynków, na których transakcje ogółem przyniosły kolejno: zauważalny zysk, śladowy zysk lub stratę, zauważalną stratę.

<sup>5</sup> Średnia liczba transakcji na rynek.

Z załączonego zestawienia wynika, że użycie śladów aktywności faktycznie wpływa na polepszenie jakości procesu nauki. Lepiej poradził sobie z

zadaniem algorytm Q-learning, co wskazuje, że dla tego typu danych jego formuła aktualizacji jest bardziej odpowiednia. Nie dość, że przeprowadzał on transakcje zauważalnie częściej niż inne, to średni zysk na transakcję był wyższy. Warto porównać w przypadku każdego algorytmu różnice udziałów z kolumn 3. i 4. Informuje ona na to, że graczom faktycznie udaje się zwiększyć skuteczność w czasie gry na rynku.

## 6. Podsumowanie

Celem pracy było zbudowanie aplikacji symulującej grę na giełdzie zakładów sportowych z wykorzystaniem algorytmów uczenia się ze wzmocnieniem. Przedstawiono w sposób formalny ogólne zasady, według których działają konkretne algorytmy tego typu. Przybliżono proces decyzyjny Markowa jako model matematyczny zadania uczenia się ze wzmocnieniem. W jego ramach zdefiniowano autorski uproszczony model giełdy, podkreślając niedeterministyczny charakter tego środowiska.

Mając na względzie rozwiązywany problem skupiono się na klasie metod różnic czasowych jako stosunkowo nowym podejściu w tym rodzaju uczenia się maszyn. Przedstawiono możliwie rozbudowane wersje algorytmów konkretnych - Q-learning oraz Sarsa. Uwzględniono w nich współczynnik uczenia się  $\alpha$ , którego zmniejszona wartość odgrywa ważną rolę w środowiskach o dużym stopniu losowości. Opisano ślady aktywności jako remedium dla problemu odroczonej nagrody (ang. *delayed rewards*).

Przy użyciu języka Python oraz narzędzi pomocniczych zaimplementowano wymienione algorytmy. Stworzony moduł nauki został wykorzystany do zbudowania symulacji przeprowadzania transakcji giełdowych przez gracza-bota. By to osiągnąć szczegółowo zdefiniowano charakter środowiska pracy agenta uczącego się oraz możliwe interakcje.

Skuteczność nauki zbadano na rzeczywistych danych giełdowych oraz na wygenerowanych uproszczonych sekwencjach zmian kursu. Przeprowadzono testy porównujące różne wersje algorytmów. Również na ich podstawie określono rolę oraz dobór poszczególnych parametrów nauki. Wyniki przeprowadzanych symulacji prezentowano nie tylko w formie statystyk, ale także w postaci wykresów. Mimo wprowadzonych uproszczeń można stwierdzić, że zastosowane algorytmy bardzo dobrze poradziły sobie z tym zadaniem. W miarę postępu prac zauważono potencjalne usprawnienia, które mogłyby się przyczynić do uzyskania jeszcze lepszych wyników.

Relatywnie prostym do wprowadzenia ulepszeniem byłaby zmiana wartości parametrów takich jak  $\alpha$  i  $\lambda$  w miarę postępowania procesu nauki, gdyż obecnie zainicjowane wartości pozostają niezmiennie. Do rozważenia byłoby również użycie bardziej złożonej polityki wyboru akcji niż zastosowana  $\epsilon$ -zachłanna, np. strategii Boltzmanna. Bardziej złożonym ulepszeniem byłoby daleko idące rozbudowanie procesu nauki agenta. W obecnym rozwiązaniu obejmuje on grę tylko i wyłącznie na jednym rynku. W tej chwili więc za każdym razem grę rozpoczyna nowy, niedoświadczony acz skory do nauki gracz. Naturalnym usprawnieniem byłoby stworzenie mechanizmu czerpania ogólnej i uniwersalnej wiedzy, pozwalającego wyciągać wnioski z długookresowej gry na wielu rynkach.

Język Python dobrze sprawdził się jako narzędzie do implementacji opi-

sywanych algorytmów. Stworzył on jednakże pewne ograniczenia związane z wydajnym przeprowadzanych symulacji. Dla każdego z rynków zgodnie z używanym modelem nauki może ona być wykonana niezależnie, czego następstwem było podjęcie próby zrównoleglenia tego procesu. Niestety wykorzystywany interpreter języka Python uniemożliwia rozwiązanie tego problemu w tradycyjnym podejściu, czyli posługując się wątkami (ang. *threads*). Dzieje się to z powodu zastosowania tzw. Global Interpreter Lock (GIL), który jest swoistym zabezpieczeniem przed wykonywaniem wielu wątków jednocześnie. Jest to niestety jeden z mankamentów najpopularniejszej implementacji języka Python (nieobecny np. w Jython czy IronPython) i mimo, że istnieją rozwiązania zastępcze, to według autora nie na tyle atrakcyjne, aby je zastosować.

Autor pracy jest świadom faktu, że niniejsza praca jest jedynie wstępem do jakże rozległego obszaru uczenia maszynowego, niemniej jednak stanowi zachętę do prowadzenia dalszych badań w tej dziedzinie.

## A. Uruchamianie projektu

Rozdział zawiera pełną instrukcję konfiguracji i użytkowania wszystkich aplikacji stworzonych w ramach tej pracy.

### A.1. Wymagania

Wymagania co do środowiska uruchomieniowego zostały zdefiniowane na podstawie tego, na jakich systemach operacyjnych oraz przy użyciu których wersji zastosowanych narzędzi udało się uruchomić wszystkie aplikacje w ramach projektu. W przypadku tych drugich zostały podane najstarsze wersje, dla których nie stwierdzono problemów.

- system operacyjny z rodziny Windows lub Linux (wraz ze środowiskiem graficznym),
- interpreter języka Python w wersji 2.7 (wymagana jego ścieżka w zmiennej środowiskowej PATH),
- biblioteka NumPy w wersji 1.6,
- biblioteka Matplotlib w wersji 1.1.0.

Dodatkowym wymaganiem jest dodanie do zmiennej systemowej PYTHONPATH ścieżki z aplikacją. Można to wykonać otwierając terminal odpowiedni dla posiadanego systemu operacyjnego oraz wykonując polecenie podane poniżej. Przykłady te zakładają, że SOURCE\_PATH to katalog nadrzędny względem folderu ml\_project:

- w systemie Linux:

```
SOURCE_PATH=/home/mgr-jancarz/src/main
export PYTHONPATH=${PYTHONPATH}:${SOURCE_PATH}
```

- w systemie Windows:

```
set SOURCE_PATH=C:\mgr-jancarz\src\main
set PYTHONPATH=%PYTHONPATH%;%SOURCE_PATH%
```

### A.2. Konfiguracja źródła danych rynkowych

Aplikacje data\_visualizer.py oraz simulation\_launcher.py mogą korzystać z dwóch typów źródeł z danymi rynkowymi - bazy danych MongoDB oraz pliku JSON. Ponieważ pierwszy typ wymaga dodatkowej instalacji systemu bazodanowego wraz z aplikacją dostarczone są pliki z danymi. Domyślnie wykorzystywane są 2 pliki:

- `marketData-all.json` - plik zawierający rzeczywiste archiwalne dane dotyczące 438 rynków sportowych,
- `marketData-mock.json` - plik ze sztucznymi danymi generowanymi skrypcem `data_generator.py` (jego opis znajduje się w sekcji A.3).

Konfiguracja dotycząca źródła danych oraz innych ustawień dla obu aplikacji zawarta jest w pliku `settings.cfg` zaprezentowanym na listingu A.1.

Listing A.1: Domyślna zawartość pliku konfiguracyjnego `settings.cfg`.

```
[DB Access]

db_server_ip =
db_port =
db_name =

[ Configuration ]

allow_db_as_data_source = False
data_source_type = f
data_source_filename = marketData-all.json
; data_source_filename = marketData-mock.json

alpha = 0.4
gamma = 1.0
epsilon = 0.1
alg = Q

markets_to_process = -1
simulations_to_run = 1
time_steps_to_omit = 1
```

Należy zwrócić uwagę na poniższe zmienne:

- `allow_db_as_data_source`: Ustala możliwość wykorzystywania bazy danych jako źródła danych (dotyczy aplikacji `data_visualizer.py`). Domyślnie ustawiona na `False` i taką wartość należy pozostawić.
- `data_source_type`: Typ wykorzystywanego źródła danych. Dozwolone są wartości: `d` (baza danych) oraz `f` (plik JSON). Należy pozostawić domyślną wartość zmiennej.
- `data_source_filename`: Nazwa pliku z danymi, z którego korzysta aplikacja, jeśli ustawiony typ źródła danych to plik. Przewidziane wartości tej zmiennej to `marketData-all.json` lub `marketData-mock.json`.

### A.3. Uruchamianie generatora sztucznych danych

Skrypt `simulation_data_generator.py` pozwala generować sztuczne dane rynkowe. Po jego uruchomieniu w pliku `marketData-mock.json` zapisywany jest 1 obiekt z danymi. Możliwe jest wygenerowanie danych w jednym z trzech predefiniowanych modeli danych. Typ modelu ustawiany jest za pomocą atrybutu `-m/--model`, natomiast liczbę wartości do wygenerowania ustala atrybut `-n/--number` (domyślna wartość to 2000).

Obsługiwane wartości atrybutu modelu danych:

- L/l - funkcja liniowa malejąca,
- G/g - rozkład Gaussa,
- S/s - sinusoida.

Dane zostają wygenerowane po wywołaniu poniższego polecenia:

```
python simulation_data_generator.py [opcjonalne argumenty]
```

## A.4. Aplikacja wizualizacji danych

Szczegółowy opis aplikacji znajduje się w sekcji 5.5. Program uruchamiany jest za pomocą polecenia:

```
python data_visualizer.py
```

## A.5. Konfiguracja i uruchamianie symulacji nauki

Aplikacja symulująca grę na giełdzie zakładów sportowych korzysta z konfiguracji zawartej w pliku `settings.cfg`. Ustawione w niej wartości zmiennych mogą być nadpisywane za pomocą parametrów uruchomieniowych skryptu. Zmienne związane ze źródłami danych zostały przedstawione w sekcji A.2. W tabeli A.1 znajduje się opis pozostałych zmiennych oraz atrybutów skryptu.

Symulacja uruchamiana jest za pomocą polecenia:

```
python simulation_launcher.py [opcjonalne argumenty]
```

Po zakończeniu symulacji jej rezultat zostaje wydrukowany w konsoli systemowej oraz trafia do pliku `result.txt`.



Tabela A.1: Dostępne zmienne konfiguracyjne oraz atrybuty uruchomieniowe skryptu `simulation_launcher.py`.

Zmienna	Atrybut	Opis
—	-h, --help	Wyświetlenie instrukcji użytkownika skryptu.
alpha	-a, --alpha	Ustawienie współczynnika uczenia się $\alpha \in [0, 1]$ .
gamma	-g, --gamma	Ustawienie współczynnika dyskontowania nagród $\gamma \in [0, \infty]$ .
epsilon	-e, --epsilon	Ustawienie prawdopodobieństwa podejmowania decyzji losowych (tzw. eksplorujących) $\varepsilon \in [0, 1]$ .
alg	--alg	Konfiguracja wykorzystywanego w symulacji algorytmu nauki. Wartość zmiennej lub atrybutu to ciąg znakowy z następującymi elementami składowymi:  Znak 1. - implementacja algorytmu <ul style="list-style-type: none"> <li>• <math>Q/q</math> (Q-learning)</li> <li>• <math>S/s</math> (Sarsa)</li> </ul> Znak 2. - metoda śladów aktywności (opcjonalnie) <ul style="list-style-type: none"> <li>• <math>W/w</math> (Watkinsa)</li> <li>• <math>N/n</math> (naiwna)</li> </ul> Kolejne znaki - wartość liczbowa współczynnika świeżości (opcjonalnie) <ul style="list-style-type: none"> <li>• Liczba rzecz. z przedziału <math>[0,1]</math></li> </ul> Przykładowe wartości atrybutu/zmiennej: <ul style="list-style-type: none"> <li>• <math>Q</math></li> <li>• <math>sN0.1</math></li> <li>• <math>Qw0.42</math></li> </ul>
markets_to_process	-m, --markets	Liczba rynków giełdowych do rozegrania w ramach symulacji. Wartość -1 oznacza wszystkie dostępne rynki.
simulations_to_run	-s, --sims	Liczba symulacji do przeprowadzenia.
time_steps_to_omit	-o, --omit	Liczba całkowita oznaczająca liczbę kroków czasowych do pomijania. Wartość większa niż 1 przyspiesza symulację, ale powoduje, że jest mniej precyzyjna.
—	--config	Polecenie skorzystania ze specjalnego pliku konfiguracyjnego <code>simulation_config.json</code> , w którym można w prosty sposób zdefiniować wiele zestawów parametrów symulacji. Powoduje pominięcie wszystkich innych atrybutów z wyjątkiem <code>--ds</code> oraz <code>-h/--help</code> .

## B. Kod źródłowy

W tym dodatku zamieszczono fragment pliku `td_learning.py`, w którym znajduje się implementacja algorytmów Q-learning oraz Sarsa. Plik zawiera definicję klasy `TDAgent` reprezentującej agenta w procesie nauki.

Kluczowym fragmentem skryptu są metody `__learn_ql` oraz `__learn_sarsa`, w których wyliczany jest składnik  $\Delta$ , czyli błąd TD potrzebny do aktualizacji macierzy wartości  $Q$ . Jej aktualizacja odbywa się w metodzie `__update_q_matrix` na podstawie wyliczonego składnika.

Listing B.1: Implementacja algorytmów Q-learning oraz Sarsa zawarta w module `td_learning`.

```
# -*- coding: utf-8 -*-
__author__ = 'Mateusz Jancarz'

import numpy
import random

class TDAlgorithm:
    """
    Klasa opakowujaca stale z identyfikatorami algorytmow.
    """
    Q_LEARNING = 0
    SARSA = 1

class ETracingMethod:
    """
    Klasa opakowujaca stale z identyfikatorami metod sladow aktywnosci.
    """
    WATKINS = 0
    NAIVE = 1

class ETrace:
    """
    Klasa reprezentujaca typ sladow aktywnosci.
    :param tracing_type: identyfikator metody
    :param factor: wartosc wspolczynnika swiezosci
    """

    def __init__(self, tracing_type, factor):
        self.type = tracing_type
        self.factor = factor

class TDAgent:
```

```

"""
Klasa reprezentujaca agenta uczacego sie metodami roznic czasowych.
:param q: poczatkowa postac macierzy wartosci Q
:param learning_rate: wskaznik zapamietywania
:param chance_to_explore: prawdopodobienstwo podjecia akcji eksplorujacej
:param discount_factor: wspolczynnik dyskutowania
:param e_tracing: obiekt reprezentujacy typ sladow aktywnosci
:param alg: identyfikator algorytmu konkretnego
:raise ValueError: wyrzucany w przypadku bledow walidacji parametrow
"""

def __init__(self, q,
             learning_rate=0.1,
             chance_to_explore=0.25,
             discount_factor=0.8,
             e_tracing=None,
             alg=TDAlgorithm.Q_LEARNING):

    self.Q = TDAgent.__validated_q_matrix(q)

    for arg in (learning_rate, chance_to_explore, discount_factor):
        if not isinstance(arg, (int, long, float)) and not (0 <= arg <= 1):
            raise ValueError('Learning rate and chance to explore must be a
                              'number in [0,1].')

    if not isinstance(discount_factor,
                      (int, long, float)) and discount_factor < 0:
        raise ValueError('Discount factor must be a positive number.')
```

```

    if e_tracing is not None and not isinstance(e_tracing, ETrace):
        raise ValueError(
            'Argument e_tracing should be an instance of ETrace class.')
```

```

    else:
        self.e = numpy.array(self.Q)
        self.e[~numpy.isnan(self.e)] = 0.0

    self.Q = self.__validated_q_matrix(q)

    self.epsilon = chance_to_explore
    self.gamma = discount_factor
    self.alpha = learning_rate

    self.current_state = None
    self.current_action = None
    self.action_in_next_state = None
    self.last_action_reward = None

    self.e_tracing = e_tracing
    self.alg = alg

    @staticmethod
    def __validated_q_matrix(array):
        """
        Sprawdza wymiary wejsciowej tablicy nastepnie zwraca jako numpy.ndarray.
        :param array: tablica (dwuwymiarowa lista lub numpy.ndarray)
        :return: :raise ValueError: wyrzucany w przypadku bledow walidacji
        """

```

```

is_not_list = array is None or not (
    isinstance(array, numpy.ndarray) or isinstance(array, list))

if is_not_list:
    raise ValueError('Initial Q array must be a two-dimensional list '
                    'or numpy.ndarray.')
if not len(array) > 0:
    raise ValueError('Provided array is empty.')

q = numpy.array(array)
if len(q.shape) < 2:
    raise ValueError('Provided array has wrong format.')
return q

def set_initial_state(self, state):
    """
    Ustawia stan początkowy agenta.
    :param state: indeks stanu
    """
    if not isinstance(state, (int, long)) or \
        not 0 <= state < self.Q.shape[0]:
        raise ValueError('Initial state identifier must be a positive '
                        'integer that is lower than Q matrix rows number '
                        '(which is ' + str(self.Q.shape[0]) + ').')

    if self.current_state is None:
        self.current_state = state

def __get_possible_actions(self, state=None):
    """
    Zwraca liste mozliwych do wykonania akcji dla okreslonego stanu.
    :param state: indeks stanu
    :return: lista z indeksami akcji
    """
    state = self.current_state if state is None else state
    return (~numpy.isnan(self.Q[state])).nonzero()[0]

def __get_most_valuable_action(self, state=None):
    """
    Zwraca indeks akcji z najwyzsza (lub losowa w przypadku rownych
    wartosci) wartoscia Q dla podanego lub obecnie przyjmowanego stanu.
    :param state: indeks stanu
    :return: indeks akcji
    """
    in_state = self.current_state if state is None else state

    available_actions = self.Q[in_state]

    most_val_actions = numpy.argwhere(
        available_actions == numpy.nanmax(
            available_actions))

    return random.choice(most_val_actions)[0] if len(
        most_val_actions) > 1 else most_val_actions[0][0]

def __choose_action_e_greedily(self, state=None):
    """

```

```

Zwraca indeks akcji zgodnej z polityka e-zachlanna wzgledem wartosci Q
dla podanego lub obecnie przyjmowanego stanu.
:param state: indeks stanu
:return: indeks akcji
"""
in_state = state if state is not None else self.current_state

choose_best_action = self.epsilon < random.random()

return self.__get_most_valuable_action(in_state) \
        if choose_best_action \
        else random.choice(self.__get_possible_actions(in_state))

def choose_action_to_perform(self):
    """
    Zwraca i ustawia indeks akcji optymalnej pod wzgledem wartosci
    Q zgodnie ze strategia e-zachlanna.
    :return: indeks akcji
    """
    self.last_action_reward = None
    self.current_action = self.__choose_action_e_greedily()
    return self.current_action

def provide_reward_for_last_action(self, reward):
    """
    Umozliwia przekazanie do obiektu ucznia wzmocnienia otrzymanego po
    wykonaniu akcji.
    :param reward: wartosc wzmocnienia
    """
    if not isinstance(reward, (int, long, float)):
        raise ValueError('Reward must be a number.')
    self.last_action_reward = reward

def __learn_ql(self, next_state):
    """
    Wylicza i zwraca jeden ze skladnikow aktualizacji dla algorytmu
    Q-learning.
    :param next_state: indeks stanu
    :return: wartosc skladniku delta
    """
    max_action_value = numpy.nanmax(
        [self.Q[next_state, action] for action in
         self.__get_possible_actions(next_state)])

    delta = self.last_action_reward + self.gamma * max_action_value - \
            self.Q[self.current_state, self.current_action]

    return delta

def __learn_sarsa(self, next_state):
    """
    Wylicza i zwraca jeden ze skladnikow aktualizacji dla algorytmu SARSA.
    :param next_state: indeks stanu
    :return: wartosc skladniku delta
    """
    self.action_in_next_state = self.__choose_action_e_greedily(
        next_state)

```

```

delta = self.last_action_reward + self.gamma * \
        self.Q[next_state, self.action_in_next_state] - \
        self.Q[self.current_state, self.current_action]

return delta

def __update_q_matrix(self, delta):
    """
    Aktualizuje macierz Q.
    :param delta: wartosc skladnika delta
    """
    if self.e_tracing is not None:

        self.e[self.current_state, self.current_action] += 1.
        most_val_action = self.__get_most_valuable_action()

        for x in xrange(self.Q.shape[0]):
            for y in xrange(self.Q.shape[1]):

                self.Q[x, y] += delta * self.alpha * self.e[x, y]

                if self.current_action == most_val_action:
                    self.e[x, y] *= self.gamma * self.e_tracing.factor
                else:
                    if self.e_tracing.type == ETracingMethod.WATKINS:
                        self.e[x, y] = 0.

    else:
        self.Q[self.current_state, self.current_action] \
            += delta * self.alpha

def go_to_next_state(self, next_state):
    """
    Umozliwia zdefiniowanie stanu, w ktorym ma sie znalezc agent
    po wykonaniu akcji. Dodatkowo przeprowadza czynnosci zwiazane
    z wartosciowaniem akcji.
    :param next_state: indeks stanu
    :raise ValueError: wyrzucany w przypadku braku wcześniejszego
        wywołania metody provide_reward_for_last_action()
    """
    if self.last_action_reward is None:
        raise ValueError('Reward for the last action was not provided.')

    self.current_action = self.current_action \
        if self.action_in_next_state is None \
        else self.action_in_next_state

    delta = self.__learn_ql(next_state) \
        if self.alg == TDAlgorithm.Q_LEARNING \
        else self.__learn_sarsa(next_state)

    self.__update_q_matrix(delta)
    self.current_state = next_state

```

---

## Bibliografia

- [1] Science Daily, *Big Data, for better or worse: 90% of world's data generated over last two years*, 2013,  
<http://www.sciencedaily.com/releases/2013/05/130522085217.htm>
- [2] Wikipedia, Machine learning, 2015,  
[https://en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning).
- [3] Tom M. Mitchell, *The Discipline of Machine Learning*, School of Computer Science Carnegie Mellon University Pittsburgh, 2006.
- [4] Wikipedia, Arthur Samuel, 2015,  
[https://en.wikipedia.org/wiki/Arthur\\_Samuel](https://en.wikipedia.org/wiki/Arthur_Samuel).
- [5] Michael L. Littman, *Reinforcement learning improves behaviour from evaluative feedback*, Nature Insight, vol. 521, 2015.
- [6] Antoine Cully, Jeff Clune, Danesh Tarapore, Jean-Baptiste Mouret, *Robots that can adapt like animals*, Nature Insight, vol. 521, 2015.
- [7] Pawel Cichosz,  
*Systemy uczące się*, Wydawnictwa Naukowo-Techniczne, Warszawa, 2000.
- [8] Willi Richert, Luis Pedro Coelho,  
*Building Machine Learning Systems with Python*, Packt Publishing, 2013.
- [9] Richard S. Sutton, Andrew G. Barto,  
*Reinforcement Learning: An Introduction*, MIT Press, Cambridge, 1998.
- [10] Python Programming Language - Official Website,  
<http://www.python.org/>.
- [11] Pakiet NumPy, 2015,  
<http://www.numpy.org/>.
- [12] Biblioteka Matplotlib, 2015,  
<http://matplotlib.org/>.
- [13] *Systemy Eksperckie*, Zakład Elektrotechniki i Informatyki Uniwersytetu Rzeszowskiego,  
<http://www.neurosoft.edu.pl/tkwater/tk/se.pdf>
- [14] Schemat uczenia ze wzmocnieniem - ilustracja,  
<https://web.stanford.edu/group/pdplab/pdphandbook/handbookch10.html>
- [15] FuturistSpeaker, *Artificial Intelligence will be Crashing the Stock Market in 3, 2, 1...*, 2015,  
<http://www.futuristspeaker.com/2014/06/artificial-intelligence-will-be-crashing-the-stock-market-in-3-2-1/>
- [16] John Moody, Lihong Wu, Yuansong Liao, Matthew Saffell  
*Performance functions and reinforcement learning for trading systems and portfolios*, Oregon Graduate Institute of Science and Technology Department of Computer Science and Engineering, Portland, OR, 1997.
- [17] Juan M. Alberola, Ana Garcia-Fornes, Agustin Espinosa  
*Price Prediction in Sports Betting Markets*, Departament de Sistemes Informàtics i Computació, Universitat Politècnica de València, Camí de Vera s/n. 46022, València, Spain