JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 16 (2020) 233–240
Received: 27 August 2020
Accepted: 9 September 2020

Analysis of selected methods of creating artificial intelligence on the example of a popular card game

Analiza wybranych metod tworzenia sztucznej inteligencji na przykładzie popularnej gry w karty

Łukasz Gałka*, Mariusz Dzieńkowski

Department of Computer Science, Lublin University of Technology, ul. Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The aim of the article was to analyze selected methods of creating artificial intelligence in a popular card game. Two experiments were conducted: with a human and with a computer. The following algorithms were analyzed: random, min-max, based on a neural network, statistical and statistical with the use of "cheating" technique. The examined parameters were as follows: efficiency, execution time, number of implementation code lines, implementation time and training duration. The indicator with the greatest impact on the selection of the most optimal method was efficiency. The research has shown no difference in efficiency for the neural network-based algorithm and the statistical algorithm. In other cases, the differences in this feature were significant. The use of the "cheating" technique has increased the efficiency.

Keywords: artificial intelligence; machine learning; algorithm efficiency evaluation; computer games

Streszczenie

Celem artykułu była analiza wybranych metod tworzenia sztucznej inteligencji w popularnej grze w karty. Zostały przeprowadzone dwa eksperymenty: z człowiekiem oraz z komputerem. Analizie poddano algorytmy: losowy, min-max, bazujący na sieci neuronowej, statystyczny oraz statystyczny z użyciem techniki „oszukiwania". Zbadano takie parametry jak: skuteczność, czas wykonania, liczbę linii kodu implementacji, czas implementacji oraz czas trwania treningu. Wskaźnikiem mającym największy wpływ na wybór najbardziej optymalnej metody była skuteczność. Badania wykazały brak różnic w skuteczności dla algorytmu bazującego na sieci neuronowej i algorytmu statystycznego. W pozostałych przypadkach różnice tej cechy były istotne. Użycie techniki „oszukiwania" zwiększyło skuteczność.

Słowa kluczowe: sztuczna inteligencja; uczenie maszynowe; ocena skuteczności algorytmów; gry komputerowe

1. Wstęp

Postęp techniczny i zwiększenie wydajności obliczeniowej komputerów pozwoliły na stworzenie sztucznej inteligencji [1]. Poprzez dziesięciolecia sztuczna inteligencja była wykorzystywana w rozwiązywaniu najtrudniejszych problemów współczesnego świata. Komputery potrafią podejmować skomplikowane decyzje, bazując na danych i informacjach, których ludzki umysł nie jest w stanie samodzielnie przetworzyć [2]. W praktyce używa się pojęcia uczenia maszynowego ze względu na specyfikację podejmowanych przez komputer decyzji [3]. Wyróżnia się dwa główne nurty: uczenie z nadzorem oraz uczenie bez nadzoru. Uczenie z nadzorem polega na wykorzystaniu przez algorytm opracowanych gotowych wyników. Natomiast uczenie bez nadzoru nie wykorzystuje wyników, ale próbuje je przewidzieć poprzez analizę danych wejściowych [3, 4]. Do czołowych rozwiązań można zaliczyć: techniki eksploracji danych, sieci neuronowe, metody grupowania danych, metody statystyczne, drzewa decyzyjne czy algorytmy ewolucyjne [5–10]. Często techniki sztucznej inteligencji wykorzystuje się do rozwiązywania problemów, których precyzyjne rozwiązanie jest niewykonalne obliczeniowo. Przy implementacji należy brać pod uwagę dostępne gotowe narzędzia pozwalające na realizację konkretnych algorytmów. Popularnym językiem z bogatą kolekcją bibliotek uczenia maszynowego jest język Java. Należy tu wymienić najpopularniejsze biblioteki takie jak: RapidMiner, Weka, Deeplearning4j [11].

Pod koniec XX wieku rozpoczęła się era gier komputerowych, które również zaczęły używać inteligentnych rozwiązań w celu sterowania wirtualnymi światami [12]. Rzeczywiste rozgrywki zaczęły być przenoszone do wirtualnej rzeczywistości. Przykładem jest gra kar-

*Corresponding author
Email address: lukasz.galka2@pollub.edu.pl (Ł. Gałka)

ciana „Tysiąc” pierwotnie rozgrywana przy stołach wyłącznie z graczami będącymi ludźmi. W dobie superszybkich komputerów i mikroprocesorów istnieje możliwość gry z przeciwnikami sterowanymi przez maszyny i mogącymi dorównać inteligencją człowiekowi. Programiści implementują interakcję pomiędzy człowiekiem, a programem komputerowym w sposób bardzo naturalny i podobny do rzeczywistych interakcji graczy [13,14]. Dobór odpowiednich metod sztucznej inteligencji jest kluczowym aspektem. Algorytmy muszą być wydajne obliczeniowo i działać w czasie rzeczywistym. Ponadto muszą reprezentować odpowiedni poziom trudności rozgrywki, aby w pełni sprostać wymaganiom nawet najbardziej wymagających umysłów. Istnieje wiele rodzajów gier, które można podzielić na następujące gatunki: sportowe, zręcznościowe, platformowe, klasyczne, karciane, RPG, FPS czy RTS. Każdy gatunek cechują odmienne metody sterowania światem wirtualnym i jego elementami. Głównym celem artykułu jest analiza wybranych metod tworzenia sztucznej inteligencji i w wyniku tej analizy wybór najlepszej metody.

2. Aplikacja testowa

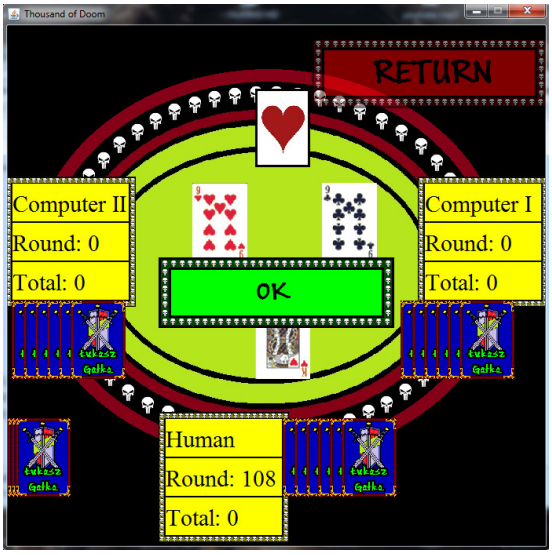
W celu analizy podstawowych parametrów metod tworzenia sztucznej inteligencji stworzono aplikację do rozgrywki w grę karcianą „Tysiąc”. Pełna wersja aplikacji została zaprogramowana w wieloplatformowym języku Java [15–17] przy użyciu graficznego interfejsu użytkownika bazującego na bibliotece Swing [18,19] oraz magazynu danych w postaci bazy danych SQLite [20,21].

2.1. Opis aplikacji

Gra obejmuje takie elementy jak: tasowanie kart, rozdawanie kart, licytację, zbieranie „musu”, wymianę kart, podbicie stawki, rozgrywkę, meldowanie atutu, zbieranie lew. W rozgrywce uczestniczy trzech graczy. Zachowanie zawodników, polegające na wyborze algorytmu rozgrywki dla graczy komputerowych można określić w ustawieniach gry. Każdy gracz może być człowiekiem lub komputerem wyposażonym w sztuczną inteligencję opartą na jednym z pięciu algorytmów: losowego, minmax, opartego na sieci neuronowej, statystycznego oraz statystycznego z „oszukiwaniem”. Rysunek 1 przedstawia interfejs, dzięki któremu możliwa jest zmiana ustawień gry.

Do celów badawczych powstały dwie, okrojone wersje aplikacji: do rozgrywki z człowiekiem oraz do rozgrywki wyłącznie między graczami komputerowymi. Obie wersje gry zawierają te same funkcjonalności oraz te same etapy gry. W celu dokonania pomiarów, a następnie porównań badawczych dokonano pewnych wstępnych założeń: w obu wersjach pozostawiono tylko etap rozgrywki, natomiast pozostałe etapy takie jak: licytacja, zbieranie „musu”, wymiana kart oraz podbijanie stawki zostały usunięte. Dla wersji z rozegraniami z graczem ludzkim gracz pierwszy był sterowany przez człowieka, natomiast w drugiej wersji był sterowany kolejno poprzez graczy komputerowych dla wszystkich pięciu algorytmów rozgrywki. Cechy wspólne dwóch wersji to zachowanie graczy drugiego i trzeciego. Gracz drugi był sterowany przez kolejne zaimplementowane algorytmy rozgrywki. Gracz trzeci był graczem sterowanym przez sztuczną inteligencję na bazie algorytmu losowego. Zrzut ekranu z interfejsu rozgrywki przedstawiono na Rysunku 2.

Rysunek 1: Zrzut ekranu z menu opcji



Rysunek 2: Zrzut ekranu z rozgrywki

2.2. Zasady gry

W rozgrywce używa się talii 24 kart w czterech kolorach: pik, trefl, karo, kier przy wartościach: dziewiątka, dziesiątka, walet, dama, król, as. Każda karta posiada przyporządkowaną wartość punktową: dziewiątka - 0 pkt., walet - 2 pkt., dama - 3 pkt., król - 4 pkt., dziesiątka - 10 pkt., as - 11 pkt. Gracz rozpoczynający rundę, wyrzuca dowolną kartę. Kolejni gracze dorzucają karty w kolorze pierwszej karty. W przypadku braku

karty w podanym kolorze gracz może wyrzucić dowolną kartę. Gracz nie ma obowiązku przebijania wyrzuconych kart. Karty zbiera gracz, który posiada najwyższą punktowo kartę w kolorze pierwszej karty. Dodatkowo para dama i król w jednym kolorze może zostać zameldowana poprzez wyrzucenie jednej karty z tej pary, w przypadku, gdy gracz rozpoczyna rundę. Gracz, który melduje, otrzymuje dodatkowe punkty w zależności od koloru meldunku: pik - 40 pkt., trefl - 60 pkt., karo - 80 pkt., kier - 100 pkt. Ponadto kolor meldunku staje się atutem. W przypadku kolejnych meldunków stary atut traci ważność i jest zastępowany nowym kolorem meldunku. Atut koloru polega na możliwości przebicia kart w innych kolorach w przypadku, gdy gracz nie posiada już kart w kolorze pierwszej wyrzuconej karty [22, 23].

2.3. Algorytmy

W badanej grze zaimplementowano 5 algorytmów bazujących na różnych technikach tworzenia sztucznej inteligencji. Wszystkie zaimplementowane rozwiązania zachowują zgodność z przedstawionymi zasadami gry w „Tysiąc" zaprezentowanymi w poprzednim rozdziale.

2.3.1. Algorytm losowy

Jest to prosty algorytm, oparty na losowaniu karty z dostępnej puli kart. Pulę kart do wyrzucenia tworzy się, bazując na zasadach gry, w taki sposób, aby w puli znajdowały się tylko te karty, które mogą być wyrzucone w danym rozegraniu. Fragment kodu źródłowego algorytmu losowego przedstawiono na Listingu 1.

Listing 1: Fragment kodu źródłowego algorytmu losowego

```
@Override
public Card getPlayingCard(PlayableTable ↩
    table) {
    List<Card> playingCards = ↩
        ThousandRulesHelper.↩
        getPlayingCards(table);
    Card playingCard = playingCards.get(↩
        random.nextInt(playingCards.size↩
        ()));
    //...
    return playingCard;
}
```

2.3.2. Algorytm min-max

Algorytm ten poszukuje rozwiązania problemu w taki sposób, aby maksymalizować zyski oraz minimalizować straty. W przypadku rozpatrywanej gry karcianej jako zysk traktowana będzie liczba punktów uzyskana poprzez zbiórkę lew oraz meldowanie atutu. Natomiast w przypadku braku możliwości zebrania lewy algorytm będzie zmniejszał liczbę punktów, które gracz utraci poprzez przegranie rundy. Charakterystyczną cechą tego rozwiązania jest konieczność znajomości taktyki gry. Jest to duże ograniczenie, natomiast pozwala w prosty i stosunkowo szybki sposób osiągnąć obraz inteligentnych zachowań. Przy implementacji w pierwszym kroku sprawdzana jest możliwość meldunku atutu. Jest to spowodowane największą wartością punktową, jaką gracz może otrzymać w danych rozegraniu. Jeżeli meldunek jest możliwy, zostanie wykorzystany, w przeciwnym razie sprawdzeniu zostaje poddany warunek przebicia kart na stole rozgrywki. W przypadku możliwości przebicia zostanie rzucona odpowiednia kart. W przeciwnym przypadku zostanie wyrzucona karta powodująca najmniejszą stratę punktów. Fragment kodu źródłowego algorytmu min-max przedstawiono na Listingu 2.

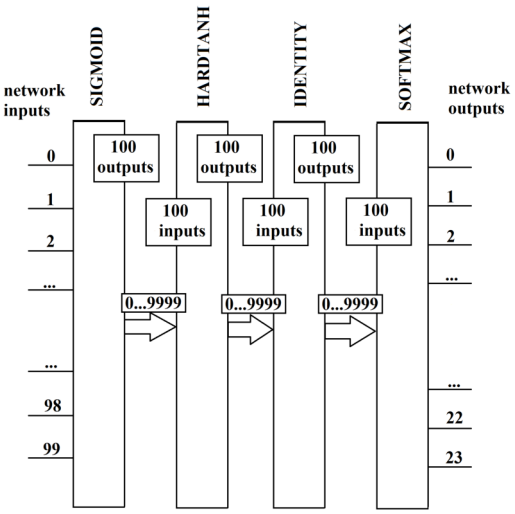
Listing 2: Fragment kodu źródłowego algorytmu min-max

```
@Override
public Card getPlayingCard(PlayableTable ↩
    table) {
    Card playingCard = null;
    List<Card> playingCards = ↩
        ThousandRulesHelper.↩
        getPlayingCards(table);

    if (table.getActualPlayerIndex() == ↩
        table.getActualWinnerIndex()) {
        int maxDispatchIndex = ↩
            getMaxDispatchIndex(↩
            playingCards);
        //...
    } else {
        int maxOvertrumpIndex = ↩
            getOvertrumpCardIndex(table, ↩
            playingCards);
        //...
    }
    //...
    return playingCard;
}
```

2.3.3. Algorytm bazujący na sieci neuronowej

Kolejnym zaimplementowanym rozwiązaniem jest algorytm bazujący na sieci neuronowej. Koncepcja ta oparta jest na wielowarstwowej sieci neuronowej ze wsteczną propagacją błędu. Trening sieci przeprowadzono na danych pozyskanych z wielokrotnych rozegrań algorytmu statystycznego ze zwiększoną liczbą analizowanych gier losowych. Utworzono dwa zbiory danych: uczący z milionem rekordów oraz walidacyjny ze stu tysiącami rekordów. Trening prowadzono do momentu, w którym błąd na zbiorze walidacyjnym zaczął się zwiększać. Struktura sieci została przedstawiona na Rysunku 3. Sieć zawiera 4 warstwy z różnymi funkcjami aktywacji. Pierwsza warstwa wejściowa posiada 100 wejść. Pierwsze 24 wejścia są odpowiedzialne za karty gracza używającego jednego z pięciu algorytmów, kolejne 24 wejścia są odpowiedzialne za kartę gracza poprzedniego, kolejne 24 wejścia są odpowiedzialne za kartę gracza następnego, kolejne 24 wejścia to zebrane karty na stole rozgrywki i ostatnie 4 wejścia są odpowiedzialne za kolor atutu. Sieć zawiera 24 wyjścia stanowiące prawdopodobieństwa wyrzucenia kart, które dadzą najlepsze wyniki dla algorytmu. Fragment kodu źródłowego algorytmu opar-



Rysunek 3: Struktura sieci neuronowej wykorzystanej w aplikacji

tego na sieci neuronowej przedstawiono na Listingu 3.

Listing 3: Fragment kodu źródłowego algorytmu opartego na sieci neuronowej

```
@Override
public Card getPlayingCard(PlayableTable ↩
    table) {
    List<Card> playingCards = ↩
        ThousandRulesHelper.↩
        getPlayingCards(table);

    double[][] featuresArray = new double↩
        [1][100];
    CardsTranslator.setCardsToVector(0, ↩
        featuresArray[0], table.↩
        getActualPlayer().getCards());
    //...
    INDArray features = Nd4j.create(↩
        featuresArray);
    INDArray predicted = network.output(↩
        features, false);
    double[] predictedArray = predicted.↩
        toDoubleVector();
    //...
}
```

2.3.4. Algorytm statystyczny

Algorytm bazuje na obliczeniach statystycznych, rozgrywa wielokrotne losowe rundy z kartami widocznymi w danej rundzie. Jest on kosztowny obliczeniowo ze względu na dużą liczbę rozegrań wykonywaną w celu podjęcia najlepszej decyzji o rzuceniu najbardziej optymalnej karty w danym rozegraniu. Duża zaletą algorytmu jest brak wiedzy programisty na temat taktyki używanej w grze. Przy każdym rozegraniu jest inicjalizowany stół będący kopią aktualnie rozgrywanego, z tą różnicą, że nieznane karty graczy są losowo rozdawane przeciwnikom. Następnie algorytm przeprowadza wielokrotne rozegrania całego rozdania z użyciem algorytmu losowego dla wszystkich graczy. Wyniki poszczególnych kart są sumowane i wybierany jest wynik dający największą liczbę punktów. Fragment kodu źródłowego algorytmu statystycznego przedstawiono na Listingu 4.

Listing 4: Fragment kodu źródłowego algorytmu statystycznego

```
@Override
public Card getPlayingCard(PlayableTable ↩
    table) {
    int sampleNumber = 100;
    //...
        oneRoundRandomTableContinuation.↩
            initializeWithoutCheating(↩
            table);
        oneRoundRandomTableContinuation.↩
            play(canPlayCards.get(j));
        sampleTable[j] += ↩
            oneRoundRandomTableContinuation↩
            .getPlayer(table.↩
            getActualPlayerIndex()).↩
            getScore();
    //...
}
```

2.3.5. Algorytm statystyczny z „oszukiwaniem”

Ostatnim rozpatrywanym algorytmem jest algorytm statystyczny z użyciem techniki „oszukiwania”. Jest to technika szeroko stosowana w branży gier komputerowych w wielu jej wariantach. W celu zwiększenia wydajności lub skuteczności algorytmu stosuje się niedopuszczalne w normalnych warunkach metody, których działanie jest niezauważalne dla gracza sterowanego przez człowieka. W niniejszym przypadku zmodyfikowano poprzedni algorytm statystyczny w taki sposób, aby inicjalizacja losowego stołu była idealną kopią kart, które posiadają gracze. Dzięki temu posunięciu algorytm losowy przeprowadza symulację dla dokładnie takich kart, jakie posiadają przeciwnicy. Fragment kodu źródłowego algorytmu statystycznego z użyciem techniki „oszukiwania” przedstawiono na Listingu 5.

Listing 5: Fragment kodu źródłowego algorytmu statystycznego z użyciem techniki „oszukiwania”

```
@Override
public Card getPlayingCard(PlayableTable ↩
    table) {
    int sampleNumber = 100;
    //...
        oneRoundRandomTableContinuation.↩
            initializeWithCheating(table)↩
            ;
        oneRoundRandomTableContinuation.↩
            play(canPlayCards.get(j));
        sampleTable[j] += ↩
            oneRoundRandomTableContinuation↩
            .getPlayer(table.↩
            getActualPlayerIndex()).↩
            getScore();
    //...
}
```

Tabela 1: Średnia liczba punktów dla gracza 2

algorytm gracza 1	algorytm gracza 2				
	losowy	min-max	sieć neuronowa	statystyczny	statystyczny z „oszukiwaniem”
człowiek	44,582	64,626	70,708	76,688	83,282
losowy	52,878	77,843	84,890	85,581	87,407
min-max	43,773	68,916	74,522	75,400	76,423
sieć neuronowa	40,277	66,312	71,807	71,799	74,141
statystyczny	40,087	65,160	70,522	70,694	73,800
statystyczny z „oszukiwaniem”	38,760	64,346	70,052	70,307	72,764

3. Metodyka badań

Zostały przeprowadzone dwa eksperymenty:

- z graczem sterowanym przez człowieka
- z graczem sterowanym przez komputer

W pierwszym badaniu człowiek rozgrywał rundy z kolejnymi zaimplementowanymi algorytmami, gdzie rozegrano po 500 rozdań na każdy algorytm. W drugim badaniu przeprowadzono eksperyment krzyżowy dla wszystkich par algorytmów, gdzie rozegrano po 10000 rozdań na każdą parę algorytmów.

Dokonano pomiarów następujących parametrów:

- skuteczność
- czas wykonania
- liczba linii kodu
- czas implementacji
- czas trwania treningu.

Wszystkie parametry poddane zostały standaryzacji według wzoru:

$$P_{i\_std} = \frac{P_i - P_{min}}{P_{max} - P_{min}} \tag{1}$$

gdzie P_{i_std} jest zestandaryzowaną wartością danego parametru i-tego algorytmu, P_i jest wartością danego parametru i-tego algorytmu, P_{min} i P_{max} są odpowiednio minimalną i maksymalną wartością danego parametru dla wszystkich algorytmów. W przypadku wielokrotnych pomiarów brano pod uwagę średnią arytmetyczną wartości parametrów.

3.1. Parametry

Przy pomiarze skuteczności przyjęto jako miarę liczbę punktów zdobytych przez gracza drugiego. Wskaźnik czasu wykonania został zmierzony dla każdego algorytmu po 40000 próbek na każdy algorytm. Czas mierzono przy użyciu funkcji System.nanoTime() języka Java. Do analizy liczby linii kodu źródłowego wzięto pod uwagę wyłącznie kod stworzony w czasie implementacji algorytmu. Podczas zliczania nie brano pod uwagę pustych linii oraz komentarzy. Ponadto uwzględniono wyłącznie kod badanego algorytmu. Każdy algorytm był rozpatrywany niezależnie. Drugim parametrem powiązanym z wytworzeniem rozwiązania był czas implementacji. Każdy algorytm był programowany przez tego samego programistę. Dokonano pomiaru czasu implementacji wyłącznie tej części implementacji, która była odpowiedzialna za dany algorytm. Nie uwzględniano oprogramowania innych części aplikacji. Ostatnim rozpatrywanym parametrem był koszt związany z czasem trwania treningu rozwiązań opartych na poszczególnych algorytmach. Ten parametr dotyczy tylko algorytmu bazującego na sieci neuronowej i był równy 45 godzinom dla sieci przedstawionej we wcześniejszych rozdziałach. Trening był prowadzony na zbiorze uczącym, liczącym milion rekordów. Trening prowadzono do momentu, aż błąd na zbiorze walidacyjnym zaczął rosnąć. Zbiór walidacyjny obejmował 100 tysięcy rekordów. Dobór liczebności zbiorów uczącego i walidacyjnego był uwarunkowany wykorzystanym do obliczeń sprzętem komputerowym. Dane do treningu zostały wygenerowane przez algorytm statystyczny ze zwiększoną liczbą rozegrań [24, 25].

3.2. Środowisko testowe i narzędzia badawcze

Wszystkie czynności zostały przeprowadzone na komputerze Dell Inspiron 3543 z procesorem Intel Core i7-5500U, wyposażonym w 16GB pamięci RAM oraz zainstalowanym systemem operacyjnym Windows 7 Professional 64-bit. Do pomiaru liczby linii kodu źródłowego został wykorzystany program Cloc [26]. Analiza statystyczna, omówiona w rozdziale 3.3.1 została wykonana za pomocą dwóch aplikacji: Matlab R2019b oraz Statistica 13 64-bit.

3.3. Wyniki

3.3.1. Skuteczność

Wyniki pomiarów przedstawiono w Tabeli 1. Można tam zauważyć różne skuteczności zaimplementowanych rozwiązań. Ponadto wyniki można uszeregować według średnich od najsłabszego do najsilniejszego: losowy, min-max, sieć neuronowa, statystyczny, statystyczny z „oszukiwaniem”.W celu analizy statystycznej otrzymanych wyników pod kątem istotności różnic pomiędzy algorytmami sprawdzono spełnienie warunku o normalności rozkładów badanych danych dla testu jednoczynnikowej analizy wariancji. Normalność rozkładów rozpatrywanych danych sprawdzono testem Kołmogorowa-Smirnowa przy poziomie istotności 0,05. Niestety założenia o normalności nie zostały spełnione i w każdym przypadku wartość istotności wynosiła $p<0{,}01$. Z tego względu istotność różnic pomiędzy algorytmami sprawdzono nieparametrycznym testem rang Kruskala-Wallisa na poziomie istotności 0,05. Wyniki testów przedstawiono w Tabeli 3. Na podstawie testu

Tabela 2: Wyniki porównań wielokrotnych testem Dunna dla algorytmów wykazujących różnice skuteczności

algorytm I	algorytm II	algorytm gracza 1					
		człowiek	losowy	min-max	sieć neuronowa	statystyczny	statystyczny z „oszukiwaniem”
losowy	min-max	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001
losowy	sieć neuronowa	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001
losowy	statystyczny	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001
losowy	statystyczny z „oszukiwaniem”	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001
min-max	sieć neuronowa	p=0,4588	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001
min-max	statystyczny	p=0,0053	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001
min-max	statystyczny z „oszukiwaniem”	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001	p<0,0001
sieć neuronowa	statystyczny	p=1,0000	p=1,0000	p=1,0000	p=1,0000	p=1,0000	p=1,0000
sieć neuronowa	statystyczny z „oszukiwaniem”	p=0,0077	p=0,0026	p=0,0667	p=0,0099	p<0,0001	p=0,0016
statystyczny	statystyczny z „oszukiwaniem”	p=0,5797	p=0,0749	p=1,0000	p=0,0029	p<0,0001	p=0,0193

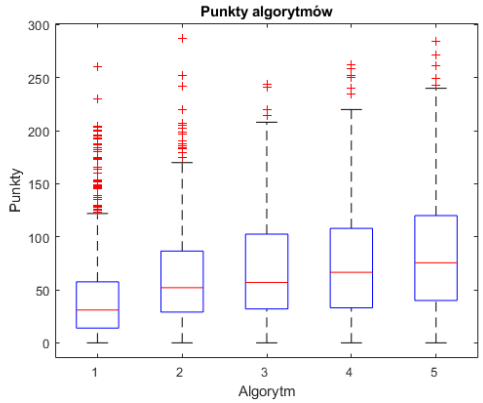
Tabela 3: Wyniki testów Kruskala-Wallisa

algorytm gracza 1	p-wartość
człowiek	p < 0,0001
losowy	p < 0,0001
min-max	p < 0,0001
sieć neuronowa	p < 0,0001
statystyczny	p < 0,0001
statystyczny z „oszukiwaniem”	p < 0,0001

można zauważyć istotne statystycznie różnice w każdej z badanych grup (p < 0,0001). Ponadto wykonano testy Dunna na poziomie istotności 0,05 w celu sprawdzenia, które podgrupy danych wykazują różnice. Wyniki testów Dunna dla rozpatrywanych algorytmów przedstawiono w Tabeli 2. W grupach gdzie p-wartość była na poziomie poniżej 0,05, różnice były istotne statystycznie. Dla pary algorytmów statystycznego oraz bazującego na sieci neuronowej widoczny był brak różnic we wszystkich badanych grupach. Pozostałe pary w większości wypadków wykazują pomiędzy sobą różnice istotne statystycznie. Wykres pudełkowy przedstawiający rozkład liczby punktów gracza drugiego dla grupy z graczem ludzkim przedstawiono na Rysunku 4.

3.3.2. Czas wykonania

Drugim badanym parametrem był czas wykonania algorytmu powiązany z jego złożonością obliczeniową. Wyniki pomiarów przedstawiono w Tabeli 4. W celu analizy różnic pomiędzy czasami sprawdzono spełnienie założeń o normalności rozkładów testem Kołmogorowa-Smirnowa przy poziomie istotności 0,05. Niestety założenie o normalności nie zostało spełnione i w każdym przypadku wartość istotności wynosiła p<0,01. Z tego względu istotność różnic pomiędzy czasami wykonania algorytmów sprawdzono testem rang Kruskala-



Rysunek 4: Wykres pudełkowy rozkładu liczby punktów gracza drugiego dla grupy z graczem ludzkim oraz algorytmów: losowego(1), min-max(2), sieci neuronowej(3), statystycznego(4) oraz statystycznego z „oszukiwaniem”(5)

Wallisa przy poziomie istotności 0,05. P-wartość wyniosła p<0,0001, przez co różnice w czasach są istotne statystycznie. W celu sprawdzenia, które pary różnią się, wykonano testy Dunna przy poziomie istotności 0,05. Wyniki testów dla porównań wielokrotnych przedstawiono w Tabeli 5. Można zauważyć, że każda para algorytmów posiada istotne statystycznie różnice czasów wykonania. Można je uszeregować od najszybszego do najwolniejszego: losowy, min-max, sieć neuronowa,

Tabela 4: Średni czas wykonania algorytmów

algorytm	średni czas wykonania [ms]
losowy	0,014
min-max	0,064
sieć neuronowa	4,632
statystyczny	22,518
statystyczny z „oszukiwaniem”	18,138

Tabela 5: Wyniki porównań wielokrotnych czasów wykonania testem Dunna

algorytm I	algorytm II	p-wartość
losowy	min-max	$p < 0{,}0001$
losowy	sieć neuronowa	$p < 0{,}0001$
losowy	statystyczny	$p < 0{,}0001$
losowy	statystyczny z „oszukiwaniem”	$p < 0{,}0001$
min-max	sieć neuronowa	$p < 0{,}0001$
min-max	statystyczny	$p < 0{,}0001$
min-max	statystyczny z „oszukiwaniem”	$p < 0{,}0001$
sieć neuronowa	statystyczny	$p < 0{,}0001$
sieć neuronowa	statystyczny z „oszukiwaniem”	$p < 0{,}0001$
statystyczny	statystyczny z „oszukiwaniem”	$p < 0{,}0001$

statystyczny z „oszukiwaniem”, statystyczny. Wykorzystanie techniki „oszukiwania” spowodowało zmniejszenie się czasu wykonania algorytmu, w porównaniu ze standardowym algorytmem statystycznym.

3.3.3. Liczba linii kodu

Kolejnym badanym parametrem była właściwość związana z wytworzeniem poszczególnego rozwiązania. Wyniki pomiarów przedstawiono w Tabeli 6. Najkrótszą implementację pod względem liczby linii kodu miał algorytm losowy. Przy algorytmie bazującym na sieci neuronowej sama implementacja sieci była niewielka, natomiast dodatkowy kod związany z wytrenowaniem sieci był bardzo obszerny. W tym przypadku może istnieć możliwość użycia pre-trenowanych rozwiązań, które zmniejszyłyby koszt treningu sieci. Przy porównaniu algorytmu statystycznego z użyciem techniki „oszukiwania” oraz bez jej użycia liczba linii kodu była na podobnym poziomie.

Tabela 6: Liczba linii kodu dla algorytmów

algorytm	liczba linii kodu
losowy	54
min-max	226
sieć neuronowa rdzeń	163
sieć neuronowa trening	1167
statystyczny	384
statystyczny z „oszukiwaniem”	367

3.3.4. Czas implementacji

Drugim parametrem związanym z wytworzeniem rozwiązania był czas jego implementacji. Wyniki czasów implementacji są proporcjonalne do liczby linii kodu dla poszczególnych algorytmów (Tabela 7). Najszybszy w implementacji był algorytm losowy. W tym przypadku również należałoby się zastanowić nad koniecznością treningu sieci neuronowej i wykorzystaniem pretrenowanego rozwiązania ze względu na duży narzut na trening. Oba rozwiązania statystyczne mają identyczny czas implementacji.

Tabela 7: Czas implementacji algorytmów

algorytm	czas implementacji [godziny]
losowy	0,25
min-max	2,33
sieć neuronowa rdzeń	4,17
sieć neuronowa trening	25,92
statystyczny	16,50
statystyczny z „oszukiwaniem”	16,50

3.3.5. Ocena końcowa algorytmów

W niniejszym podrozdziale przedstawiono oceny poszczególnych algorytmów. Tabela 8 przedstawia uzyskane wyniki badanych algorytmów dla poszczególnych kryteriów. W przypadku parametru skuteczności można zauważyć dużą różnicę pomiędzy algorytmem losowym, a pozostałymi algorytmami. Dla pozostałych parametrów algorytm losowy i min-max uzyskują podobne wartości. Algorytm bazujący na sieci neuronowej w większości przypadków ma najgorsze parametry. Rozwiązania statystyczne cechują się dużym podobieństwem wskaźników.

4. Wnioski i podsumowanie

Badania różnych metod tworzenia sztucznej inteligencji na przykładzie gry w „Tysiąc” pozwoliły scharakteryzować techniki wykorzystywane do tworzenia inteligencji w grach komputerowych. W pracy zbadano takie parametry jak: skuteczność inteligencji, czas wykonania, liczbę linii kodu, czas implementacji oraz czas trwania treningu rozwiązań.

Badania wskazały istotne różnice we wziętych pod uwagę w badaniach kryteriach oceny algorytmów. Ponadto wykorzystanie technik „oszukiwania” pozwala zwiększyć skuteczność bez pogorszenia pozostałych parametrów algorytmu. Najwyższą skuteczność wykazują algorytmy bazujące na obliczeniach statystycznych, ale charakteryzują się one najgorszymi czasami wykonania. Dobrą alternatywą dla obliczeń statystycznych może być algorytm bazujący na sieci neuronowej, gdzie przy niewielkiej utracie skuteczności zyskuje się wysoki zysk w czasie wykonania. Najgorszą skuteczność wykazuje algorytm losowy, choć był on wykorzystywany w obu algorytmach statystycznych. Na uwagę zasługuje rozwiązanie min-max, w którym przy stosunkowo dużej skuteczności uzyskano bardzo dobre pozostałe parametry. W przypadku lepszego dopracowania tej metody mogłaby ona zyskać jeszcze większą skuteczność. Niestety algorytm min-max wymaga od programisty wiedzy na temat taktyki gry. Może to zniechęcić programistów w przypadku bardziej złożonych problemów.

W przypadku parametru objętości kodu źródłowego i powiązanym z nim czasem implementacji najgorszym

Tabela 8: Zbiorcze oceny parametrów dla badanych algorytmów

parametr	algorytm				
	losowy	min-max	sieć neuronowa	statystyczny	statystyczny z „oszukiwaniem”
skuteczność	0,00	0,70	0,88	0,92	1,00
czas wykonania	0,00	0,00	0,21	1,00	0,81
liczba linii kodu	0,00	0,13	1,00	0,26	0,25
czas implementacji	0,00	0,08	1,00	0,63	0,63
czas treningu	0,00	0,00	1,00	0,00	0,00

rozwiązaniem była sieć neuronowa. Spowodowane było to dużym narzutem na wytworzenie sieci i napisanie oprogramowania do treningu. Jednocześnie sieć posiada skuteczność zbliżoną do algorytmu statystycznego w przypadku średniej liczby punktów. Badania pokazały brak istotnych statystycznie różnic w skuteczności algorytmu bazującego na sieci neuronowej i algorytmu statystycznego. Z tego względu algorytm statystyczny można zastąpić algorytmem opartym na sieci neuronowej i zaoszczędzić zasoby obliczeniowe. Dodatkowo w przypadku bardziej popularnych zastosowań istnieje możliwość wykorzystania pre-trenowanej sieci, co może skutkować rezygnacją z czasochłonnego treningu.

Literatura

[1] D. S. Cohen, T.J. Park, S.A Bustamante, Producing Games: From Business and Budgets to Creativity and Design, Routledge, 2010.

[2] T. Walsh, To żyje! Sztuczna inteligencja, Wydawnictwo Naukowe PWN, 2018.

[3] K. Wołk, Zabawa ze sztuczną inteligencją, Psychoskok, 2018.

[4] S. Nowaczyk, Frontiers in Artificial Intelligence and Applications, Thirteenth Scandinavian Conference on Artificial Intelligence, Tom 278, Holandia, 2015.

[5] T. Burczyński, W. Cholewa, W. Moczulski, Methods of artificial intelligence, Silesian University of Technology, 2009.

[6] S. Castellano, Artificial Intelligence, Genuine Learning, Talent Development, Tom 73, Wydanie 10, 2019.

[7] J. Łęski, Systemy neuronowo-rozmyte, Wydawnictwo WNT, 2008.

[8] L. Rutkowski, Metody i techniki sztucznej inteligencji, Wydawnictwo Naukowe PWN, 2009.

[9] Z. Shi, Advanced Artificial Intelligence, World Scientific, Singapur, 2011.

[10] B. Zohuri, M. Moghaddam, Neural Network Driven Artificial Intelligence : Decision Making Based on Fuzzy Logic. Nova Science Publisher, Nowy Jork, 2017.

[11] N. Joshi, Hands-On Artificial Intelligence with Java for Beginners : Build Intelligent Apps Using Machine Learning and Deep Learning with Deeplearning4j, Packt Publishing, 2018.

[12] Przegląd branży gier wideo, https://www.wepc.com/news/video-game-statistics, [13.07.2020].

[13] G. N. Yannakakis, J. Togelius, Artificial intelligence and games, Cham: Springer, 2018.

[14] G. N. Yannakakis, J. Togelius, A Panorama of Artificial and Computational Intelligence in Games, IEEE Transactions on Computational Intelligence and AI in Games, 2015.

[15] D. Brackeen, Java : tworzenie gier, Wydawnictwo Helion, Gliwice, 2004.

[16] J. S. Harbour, Beginning Java SE 6 Game Programming, MA: Course PTR, Boston, 2012.

[17] B. Kaluza, Machine Learning in Java, Packt Publishing, 2016.

[18] W. McAllister, J. Fritz, Programming Fundamentals Using Java : A Game Application Approach, Dulles, Virginia: Mercury Learning & Information, 2015.

[19] S. Pięta, M. Ścibisz, M. Wiśniewski, Podstawy tworzenia interfejsu graficznego aplikacji desktopowych w języku Java, Oficyna Wydawnicza Politechniki Warszawskiej, 2019.

[20] S. K. Aditya, P. Mohanta, V. K. Karn, Android SQLite Essentials, Packt Publishing, 2014.

[21] R. A. Johnson, Java Database Connectivity Using Sqlite: A Tutorial, Java Database Connectivity Using Sqlite: A Tutorial, Proceedings of the Academy of Information & Management Sciences, 2014.

[22] B. Rigal, S. Kupisz, Gry karciane, Helion, 2007.

[23] B. Whiter, J. Kluziński, Gry karciane, K.E. Liber 2004.

[24] T. B. Alakus, R. Das, I. Turkoglu, An Overview of Quality Metrics Used in Estimating Software Faults, International Artificial Intelligence and Data Processing Symposium, 2019.

[25] L. Buglione, A. Abran, Measurement Process: Improving the ISO 15939 Standard, Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, 2014.

[26] Dokumentacja oprogramowania CLOC, https://github.com/AlDanial/cloc, [13.07.2020].