PRO1D Raport z tworzenia AI do gry Uno

Autor: Cyprian Szewczak

numer studenta: s28492

1. Wprowadzenie

Gry karciane, takie jak Uno, stanowią doskonałe środowisko testowe do badań nad algorytmami uczenia maszynowego, ze względu na ich niepełną informację oraz wysoką losowość. Celem przedstawionego projektu jest analiza skuteczności algorytmu C4.5 w rozwiązywaniu problemów decyzyjnych w takiej grze. Wykorzystanie tego algorytmu umożliwia zbadanie, w jakim stopniu złożone drzewa decyzyjne mogą poprawić efektywność rozgrywek, minimalizując wpływ losowości i optymalizując strategie gry. Implementacja obejmuje także ograniczenie liczby graczy, co pozwala na lepszą analizę zachowań AI i porównanie jej skuteczności w stosunku do innych strategii heurystycznych.

2. Zasady gry

2.1. Ogólne zasady

Na początku gry rozdaje się po 7 kart każdemu graczowi i jedną z talii kładzie się na środek. Gracz musi dopasować swoją kartę numerem, kolorem lub symbolem do odkrytej karty. Jeżeli gracz nie posiada żadnej karty pasującej do tej odkrytej, musi pociągnąć kartę z talii. Jeśli wyciągnięta karta pasuje do odkrytej, jeszcze w tej samej kolejce gracz może ją dołożyć. Jeżeli nie – ruch ma kolejny gracz. Nie ma przymusu w dokładaniu kart.

2.1.1 Karty występujące w grze

Łącznie w podstawowej wersji gry występuje 108 kart:

- 19 kart z cyframi od 0 do 9, koloru niebieskiego
- 19 kart z cyframi od 0 do 9, koloru czerwonego
- 19 kart z cyframi od 0 do 9, koloru zielonego
- 19 kart z cyframi od 0 do 9, koloru żółtego
- 8 kart oznaczonych symbolem „+2” – 2 niebieskie, 2 czerwone, 2 zielone i 2 żółte
- 8 kart Odwrócenia, oznaczonych zakręconymi strzałkami – 2 niebieskie, 2 czerwone, 2 zielone i 2 żółte
- 4 Dzikie karty oznaczone symbolem „+4”
- 8 kart Pominięcia, oznaczonych symbolem „ϕ” – po 2 na każdy kolor
- 4 Dzikie karty, oznaczone czterokolorowym kółkiem

> img 1. Wizualizacja kart

2.1.2 Karty funkcyjne

Postój – następny gracz traci (stoi) kolejkę
Zmiana kierunku – karta zmieniająca kierunek gry na przeciwny
Weź dwie – następny gracz bierze dwie karty
Wybierz kolor + Weź cztery – zagrywający kartę deklaruje zmianę koloru na dowolnie przez siebie wybrany, następny gracz bierze 4 karty.

Wybierz kolor – zagrywający kartę deklaruje zmianę koloru na dowolnie przez siebie wybrany (jeden z kolorów dostępnych w grze)

Karty Postój, Zmiana kierunku, Weź dwie można kłaść do
danego koloru karty na szczycie – natomiast karty Wybierz kolor + Weź cztery można
kłaść na dowolną kartę.

Karty +4 nie wolno przebijać drugą taką samą kartą!

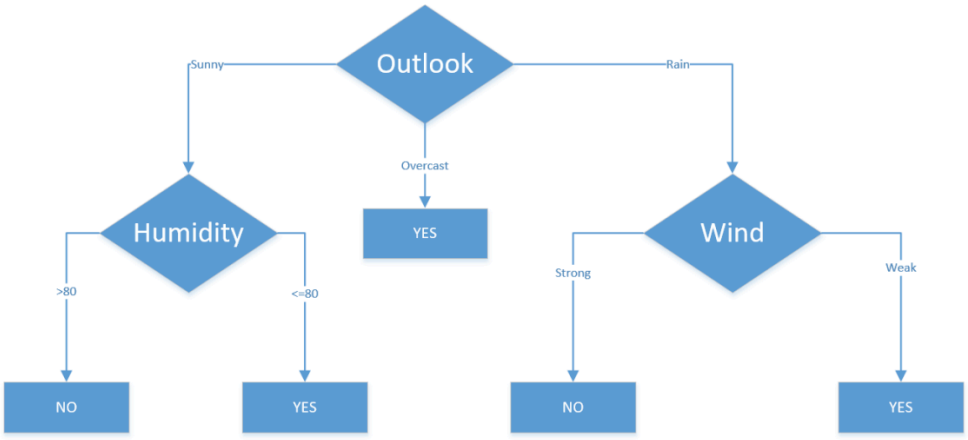
2.1.3 Zmiany względem oryginalnych zasad

Niniejsza implementacja gry karcianej różni się od pierwotnych zasad, a mianowicie została usunięta ostatnia zasada, ta zaznaczona na kolor czerwony.

W przeciwieństwie do oryginału, wersja ta pozwala na “przebijanie” karty +4 inną kartą +4. Spowodowane jest to faktem, iż alternatywna wersja jest znacznie popularniejsza niż ta oficjalna. Z tego powodu maksymalna liczba dobieranych kart przez gracza wzrasta od 20 w oryginalnej wersji, do 36 w alternatywnej.

3 Algorytm drzewa decyzyjnego C4.5

Drzewo decyzyjne C4.5 to zaawansowany algorytm stosowany w uczeniu maszynowym, oparty na wcześniejszym modelu ID3. Jego głównym celem jest klasyfikacja danych poprzez tworzenie drzewa decyzyjnego na podstawie wartości cech i ich wpływu na podział danych.



Rys 2. Wizualizacja drzewa decyzyjnego

Główna różnica między C4.5 a ID3 polega na sposobie obliczania przyrostu informacji. Algorytm C4.5 wykorzystuje wskaźnik znormalizowanego przyrostu informacji (ang. gain ratio), co pozwala na uniknięcie preferowania cech z dużą liczbą możliwych wartości. Dzięki temu podział danych jest bardziej precyzyjny i unika błędów wynikających z przeuczenia.

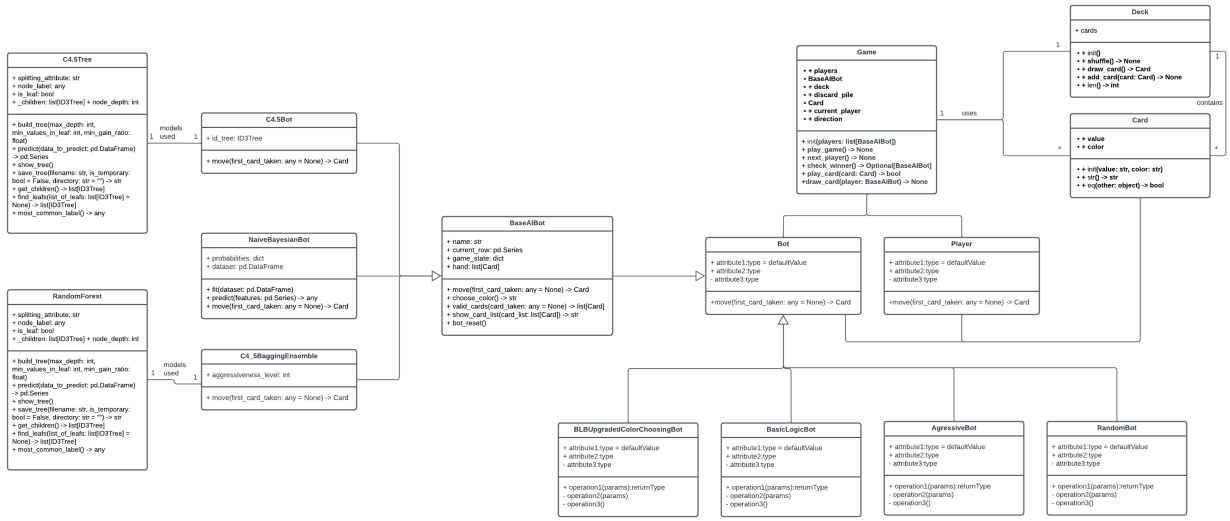
C4.5 radzi sobie również z brakującymi danymi w zbiorze treningowym, co czyni go bardziej uniwersalnym w praktycznych zastosowaniach. W przypadku brakujących wartości algorytm uwzględnia jedynie znane dane podczas obliczeń. Kolejną cechą wyróżniającą C4.5 jest jego zdolność do obsługi atrybutów numerycznych, poprzez konwersję ich na przedziały.

Działanie C4.5 opiera się na rekursywnym podziale danych. Algorytm wybiera cechę o największym wskaźniku gain ratio jako węzeł główny, a następnie powtarza ten proces dla podzbiorów utworzonych na podstawie wybranego atrybutu. Budowa drzewa kończy się, gdy dalszy podział nie wnosi nowych informacji lub gdy dane są wystarczająco jednorodne.

Dzięki swojej efektywności i zdolności do radzenia sobie z różnorodnymi typami danych.Algorytm C4.5 jest używany w wielu dziedzinach, takich jak klasyfikacja tekstu, analiza danych medycznych czy prognozowanie zachowań klientów.

4. Metodologia

4.1 Inicjalizacja środowiska



> Rys 3. Diagram klas dla projektu Uno

4.1.1 Podstawowe klasy gry

Środowisko do przeprowadzania gier jest tworzone poprzez 3 klasy **Card**, **Deck** oraz **Game**

- Klasa **Card** i klasy po niej dziedziczące służą do reprezrentacji oraz implementację zachowania kart w grze. Jej podklasy to:
  - **StopCard** - karta stopu
  - **ReverseCard** - karta Uno Reverse
  - **Plus2Card** - karta dobierania 2 kart
  - **Plus4Card** - karta dobierania 4 kart
  - **AllColors** - karta zmiany koloru
- Klasa **Deck** reprezentuje pełną kolekcję kart w grze. Zarządza dobieraniem, kładzeniem kart na stos, tasowaniem kard oraz przemieszczaniem kart ze stosu kart położonych do stosu kart do dobrania

- Klasa **Game** odpowiada za ogólną logikę gry w Uno. Jej zadaniem jest przeprowadzenie gry od początku do końca z upewnieniem sie aby wszystkie zasady tozgrywki były przestrzegane. Dodatkowo, zbiera ona informacje zarówno o zagranych ruchach jak i stanie gry. Na koniec gry zwraca te dane w postaci słownika.

4.1.2 Heurestyczne klasy graczy nie będące AI

Niewątpliwie do wytrenowania drzewa decyzyjnego potrzebne są dane. Dane te, w najlepszym wypadku powinny być tworzone na podstawie realnie przeprowadzonych gier, przez mistrzów tejże gry. Niestety istnieją dwa problemy z tak pozyskanymi danymi:

1. Brak danych w internecie. Niestety nigdzie w internecie nie ma udostępnionej żadnej bazy danych katalogującej tego typu rozgrywki.
2. Zbyt duża liczba kombinacji stanu gry, powodująca potrzebę większej bazy danych względem tego co rejestrowanie gier ludzkich jest w stanie wygenerować. Już sama przestrzeń zdarzeń elementarnych dla stanu początkowego gry wynosi:

   $$|\Omega| = \binom{108}{7} \cdot \binom{101}{7} \cdot 94 \cdot 93! \approx 5.214 \times 10^{166}$$

   gdzie:

   - $\binom{108}{7}$ - Liczba kombinacji 7 kart w ręce pierwszego gracza
   - $\binom{101}{7}$ - Liczba kombinacji 7 kart w ręce drugiego gracza
   - 94 - Liczba dla pierwszej karty położonej na stole
   - $93!$ - Liczba wszystkich permutacji kart w stosie kart do dobierania

   Liczba 5.214×10^{166} jest większa niż ilosc atomów w obserwowalnym wszechświecie 10^{80} podniesiona do kwadratu! Przypominam iż mówimy tu tylko i wyłacznie o samym stanie początkowym, nie wspominając już o wszystkich możliwośćiach przebiegu takiej gry.

Sprawia to, iż znalezienie odpowiedniego zbioru danych gier przeprowadzonych przez ludzi staje się niemożlie. Dlatego właśnie zaistniała potrzeba napisania heurestycznych botów które są w stanie rozegrać dużą ilość gier w relatywnie niedługim czasie.

Do tego celu zostały utworzone 3 klasy botów implementujących różne taktyki dla gry:

1. Klasa **RandomBot** - Implementuje on taktykę losowego wybierania kart możliwych do zagrania. Do zbioru tychże kart nie wchodzi karta do dobierania, ponieważ z oczywistych powodów dane od tego bota byłyby bezużyteczne, zważywszy na fakt, że przegrywałby w więcej niż 90% przypadków.
2. Klasa **BasicLogicBot** - Zasada tego bota jest najbardziej skomplikowana. Bot ten stara się podążać jedną z najpopularniejszych taktyk w grze.
    a. W przypadku w którym jest to możliwe, wybiera on karty do położenia w taki sposób aby kolor pasował temu którego ma najwięcej w ręce
    b. W momencie posiadania kart atakujących **Plus2Card**, **Plus4Card** oraz **StopCard** stara się on zachować je na wypadek kiedy sam zostanie zaatakowany.
    c. Kiedy żadne efekty nie są aktywne, stara się on zagrywać zwykłe karty lub **ReverseCard**
3. Klasa **AgressiveBot** - Jego taktyka polega na priorytetyzowaniu zagrywania kart atakujących w sytuacji w której jest to mozliwe.

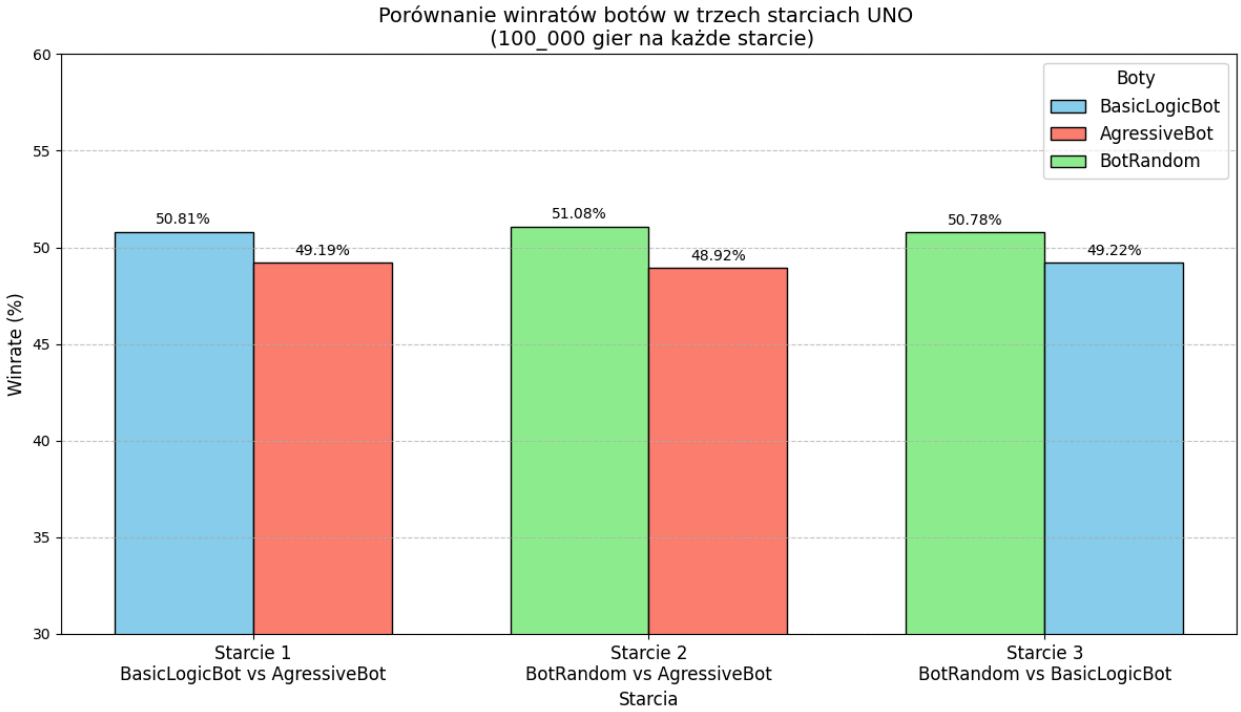
Wszystkie te boty opierają się na bardzo podstawowych logikach. Nie analizują dogłębnie sytuacji na stole, jak i historii zagranych ruchów. Jedyne atrybuty które je analizują to:

- jakie efekty na nie działają
- jakie karty są możliwe do zagrania w danej sytuacji

Dodatkowo, wszystkie te boty jeśli tylko mają możliwość zagrania karty, to to zrobią.

4.2 Rozgrywanie gier w celu zebrania danych

4.2.1 Statystyki podstawowych botów



> Rys 4. Starcia pomiędzy wszystkimi podstawowymi botami

Powyżej zaprezentowane są statystyki gier rozgrywanych między podstawowymi botami. Co zaskakujące, statystyki zwycięstw wcale nie odbiegają znacząco dla tych botów, pomimo faktu iż ich taktyki są tak radykalnie różne. Warto zwrócić uwagę na fakt, iż BotRandom zdaje się być najlepszy z nich wszystkich. Na tym etapie realizacji projektu nie byłoby nielogicznym stwierdzeniem uznanie tak małej różnicy jako dowód na to iż wpływ kart zagrywanych przez gracza jest minimalny, a dominujący efekt na zwycięstwo czy też przegraną danej jednostki ma losowość gry samej w sobie. Poleca się czytelnikowi zatrzymanie się nad tą sekcją i zastanowienie się jakich efektów należy się spodziewać dla zaimplementowanej sztucznej inteligencji.

4.2.2 Czas zbierania danych

Dzięki użyciu multiprocessingu, zbieranie danych do nauki drzewa decyzyjnego staje się bardzo proste. Rozgrywanie 100 tys. gier zajmuje nie dłużej niż parę minut na zwykłym laptopie.

4.2.3 Zebrane dane

W poniższej sekcji pozostaną wymienione kolekcjonowane atrybuty:

- **'num_red'** - liczba czerwonych kart w ręce
- **'num_green'** - liczba zielonych kart w ręce
- **'num_blue'** - liczba niebieskich kart w ręce
- **'num_yellow'** - liczba żółtych kart w ręce
- **'num_stop' -** liczba kart stopujących w ręce
- **'num_plus2' -** liczba kart “plus 2” w ręce
- **'num_plus4' -** liczba kart “plus 4” w ręce
- **'num_all_color' - l**iczba kart wielokolorowych w ręce
- **'top_card' -** karta na szczycie stosu
- **'num_cards_left_in_deck' - l**iczba kart pozostałych w decku
- **'round' - o**becna runda
- **'num_enemy_cards' - l**iczba kart w ręce drugiego gracza
- **'direction' - o**becny kierunek gry
- **'num_colors_in_hand' - o**gólna liczba kolorów w ręce
- **'num_cards_in_hand' - l**iczba kart w ręce
- **'is_game_over' -** czy gra się zakończyła w tej rundzie
- **'index_of_a_player' -** indeks obecnego gracza
- **'game_id' -** Id gry
- **'card_played' -** karta zagrana przez gracza

- **'did_win' -** czy gracz wygrał tę grę

Powyższe atrybuty zostały uważnie dobrane poprzez proces prób i błędów w celu maksymalizacji możliwości wygranej bota AI nie wydłużając przy okazji procesu uczenia. Nie wszystkie jednak atrybuty wyżej wymienione zostały użyte w procesie uczenia. Można zauważyć iż niektóre z nich nie mają znaczenia w kontekście maksymalizacji wygranych. Te dodatkowe atrybuty służą jako narzędzie eksploracji danych podczas analizy. Dane te zostają usunięte przed początkiem nauki bota.

5 Proces trenowania

5.1 Pobranie i przygotowanie środowiska

1. W celu pobrania projektu należy ściągnąć go przy pomocy platformy github z następującego linku:

   https://github.com/s28492/UnoGame.git

   git clone https://github.com/s28492/UnoGame

2. Po pobraniu, należy zainstalować wszystkie wymagane biblioteki, znajdujące się w

   https://github.com/s28492/UnoGame/blob/main/requirements.txt

   `pip install -r requirements.txt`

5.2 Trenowanie drzewa

1. Następnie za pomocą terminala będąc w głównej lokalizacji projektu uruchomić polecenie

   ```
   python3 -m Uno.main
   ```

2. W kolejnym kroku należy w odpalonym programie wybrać “1” to spowoduje że podstawowe boty zaczną grać przeciwko sobie, a następnie wpisać ile gier

mają przeciw sobie rogrywać. Zaleca się rozegrać 100_000 gier na starcie aby wygenerowane dane były wystarczające

a. Powyższy krok wygeneruje 3 pliki .csv. W celu złączenia ich należy uruchomić poniższą komendę.

```
python3 -m Uno.games_data.MergeCSV
```

Wygeneruje ona jeden plik w lokalizacji Uno/games_data/MergedCSV/ który to wykorzystamy do wytrenowania drzewa.

3. Następnym krokiem jest stworzenie drzewa decyzyjnego. W tym celu należy w terminalu wpisać:

```
python3 -m Uno.DecisionTrees.C4_5Tree --depth your_depth --filepath Uno/
```

gdzie:

- your_depth - głębokość drzewa jakie chce się uzyskać
- data_name.csv - nazwa pliku .csv na którym chcemy wytrenować drzewo

W chwili obecnej nie istnieje możliwość strojenia parametrów z pozycji terminala. Jednak nic nie stoi na przeszkodzie aby zrobić to na poziomie kodu.

▼ Poniżej przedstawiony jest przykładowy output drzewa, który będzie generowany podczas treningu:

```
   num_red  num_green  num_blue  num_yellow  num_stop  num_plus2  nu
0        3          1         1           1         0          0         1          1 ...     1
1        3          1         1           1         0          0         0          0 ...     3
2        3          1         1           0         0          0         0          0 ...     5
3        3          2         1           0         0          0         0          0 ...     7
4        3          2         0           0         0          0         0          0 ...     9

[5 rows x 18 columns]
```

```
0. None: num_plus4 → len: 100000
|   1. 0: num_all_color → len: 97267
|   |   2. 0: top_card → len: 94235
|   |   |   3. 0: num_plus2 → len: 2423
|   |   |   |   4. 0: is_game_over → len: 1852
|   |   |   |   4. 1: num_red → len: 475
|   |   |   3. 1: num_plus2 → len: 2355
|   |   |   |   4. 0: is_game_over → len: 1792
|   |   |   |   4. 1: num_red → len: 475
|   |   |   3. 2: num_plus2 → len: 2281
|   |   |   |   4. 0: is_game_over → len: 1716
|   |   |   |   4. 1: num_blue → len: 480
|   |   |   3. 3: num_plus2 → len: 2267
|   |   |   |   4. 0: is_game_over → len: 1681
|   |   |   |   4. 1: num_yellow → len: 511
|   |   |   3. 4: is_game_over → len: 1723
|   |   |   |   4. False: num_blue → len: 1690
|   |   |   3. 5: num_stop → len: 1726
|   |   |   |   4. 0: num_blue → len: 1153
|   |   |   |   4. 1: num_blue → len: 488
|   |   |   3. 6: num_stop → len: 1725
|   |   |   |   4. 0: num_red → len: 1188
|   |   |   |   4. 1: num_red → len: 443
|   |   |   3. 7: num_stop → len: 1701
|   |   |   |   4. 0: num_yellow → len: 1144
...

Tree successfully built in 7.66851806640625 seconds.
```

Wyjaśnienie outputu:

| | 2. 2: top_card → len: 385

- | | **2.** 2: top_card → len: 385 - Symbolizuje poziom głębokości drzewa

- │ │ 2. **2: top_card** → len: 385 - Symbolizuje wartość zakodowaną i zdekodowaną obecnie rozważanej gałęzi
- │ │ 2. 2: top_card → **len: 385** - Ilość elementów w danej gałęzi/liściu

Plik .pkl zostanie zapisany w poniższym folderze:

```
Uno/DecisionTrees/Models
```

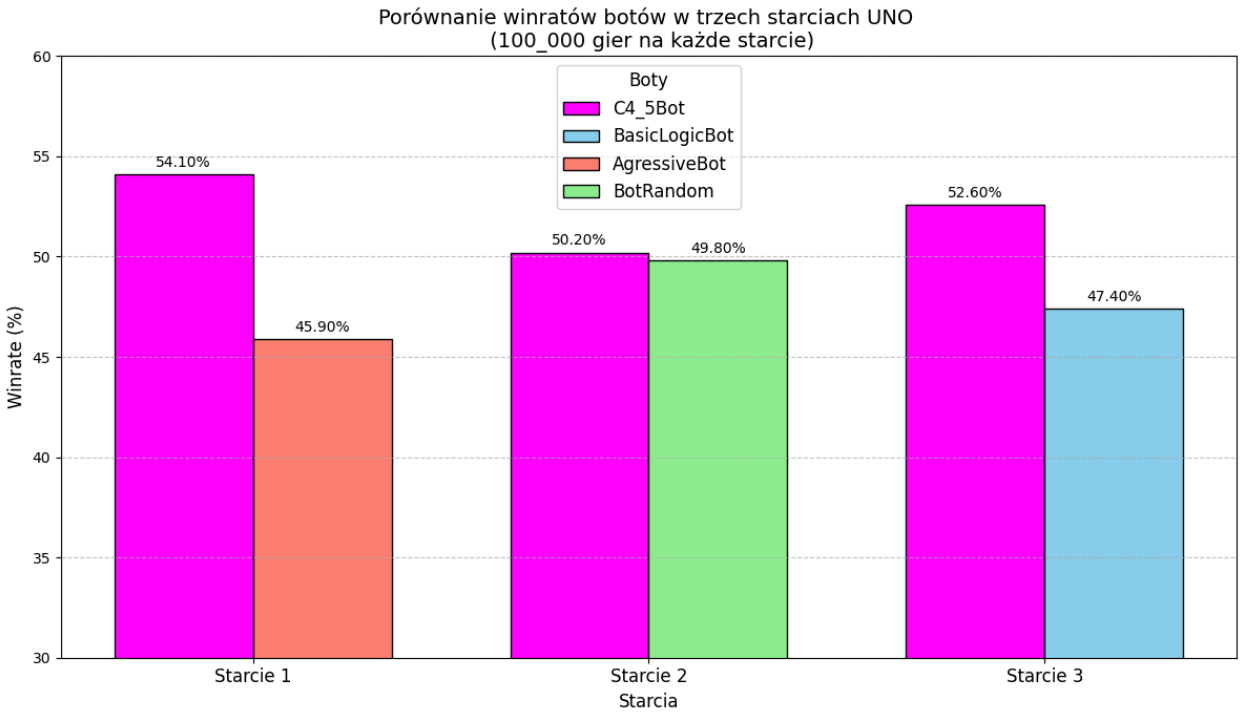
4. Po wykonaniu powyższych kroków mamy już gotowy pli .pkl z wytrenowanym drzewem. Teraz już pozostaje tylko przetestować nasze drzewo przeciwko botom. W tym celu uruchamiamy ponownie kod z kroku 1. w sekcji **“5.2 Trenowanie drzewa“** Jednak tym razem wybieramy drugą opcję i kiedy program nas poprosi to dostarczamy ścieżkę URL do drzewa.

Po wykonaniu powyższych kroków nowo wytrenowany bot rozegra gry przeciwko prostrszym botom, dzięki czemu zaistnieje możliwosć sprawdzenia tego, jak algorytm drzewa decyzyjnego C4_5 radzi sobie w grze Uno.

6 Wyniki i wnioski

6.1 Wyniki drzewa decyzyjnego w grze Uno

6.1.1 Porównanie proporcji wygranych

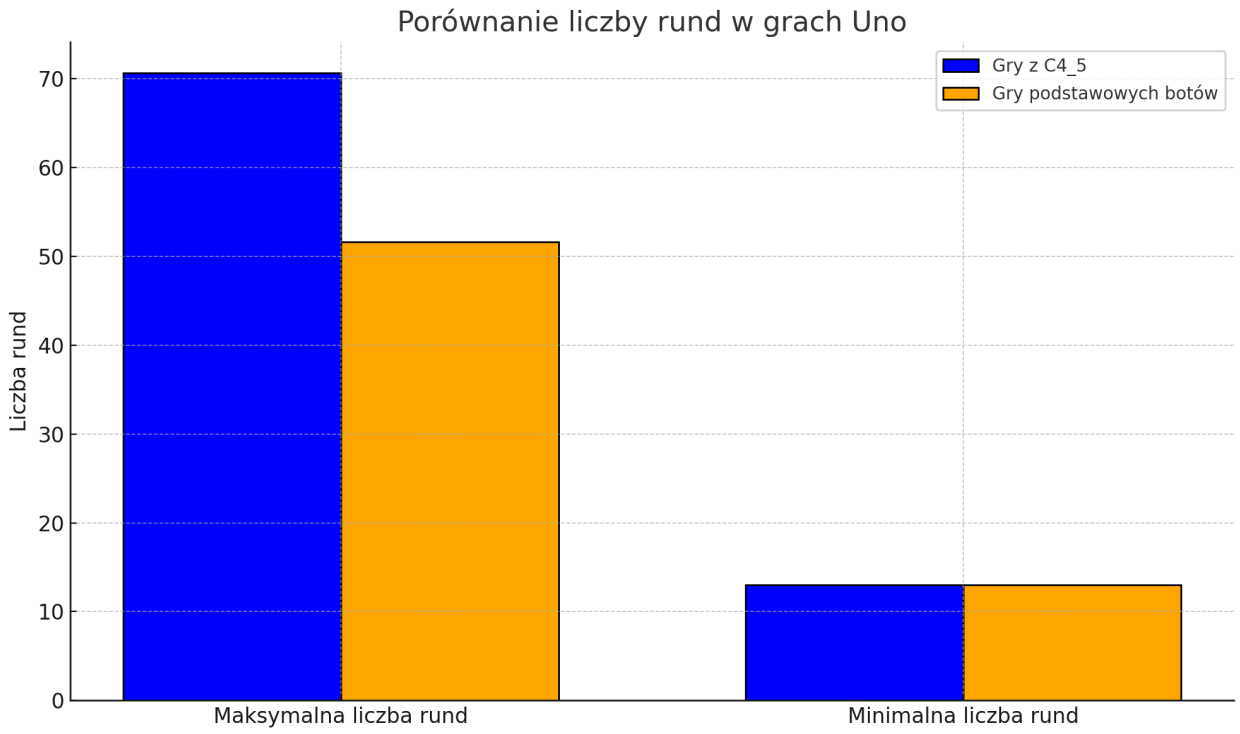


> Rys 5. Stosunek zwycięstw dla algorytmu C4.5 przeciwko pozostałym botom

1. **Porównanie AI z heurystycznymi botami:**
   Wyniki gier wskazują, że drzewo decyzyjne C4.5 przewyższało podstawowe boty heurystyczne w skuteczności wygrywania. W szczególności:
   - AI była w stanie bardziej precyzyjnie dobierać karty do aktualnej sytuacji na stole, co zmniejszało ryzyko zostania z dużą liczbą kart.
   - Zachowanie związane z planowaniem kolejnych ruchów (np. zachowanie kart specjalnych na późniejsze etapy gry) znacząco przyczyniło się do lepszych wyników.

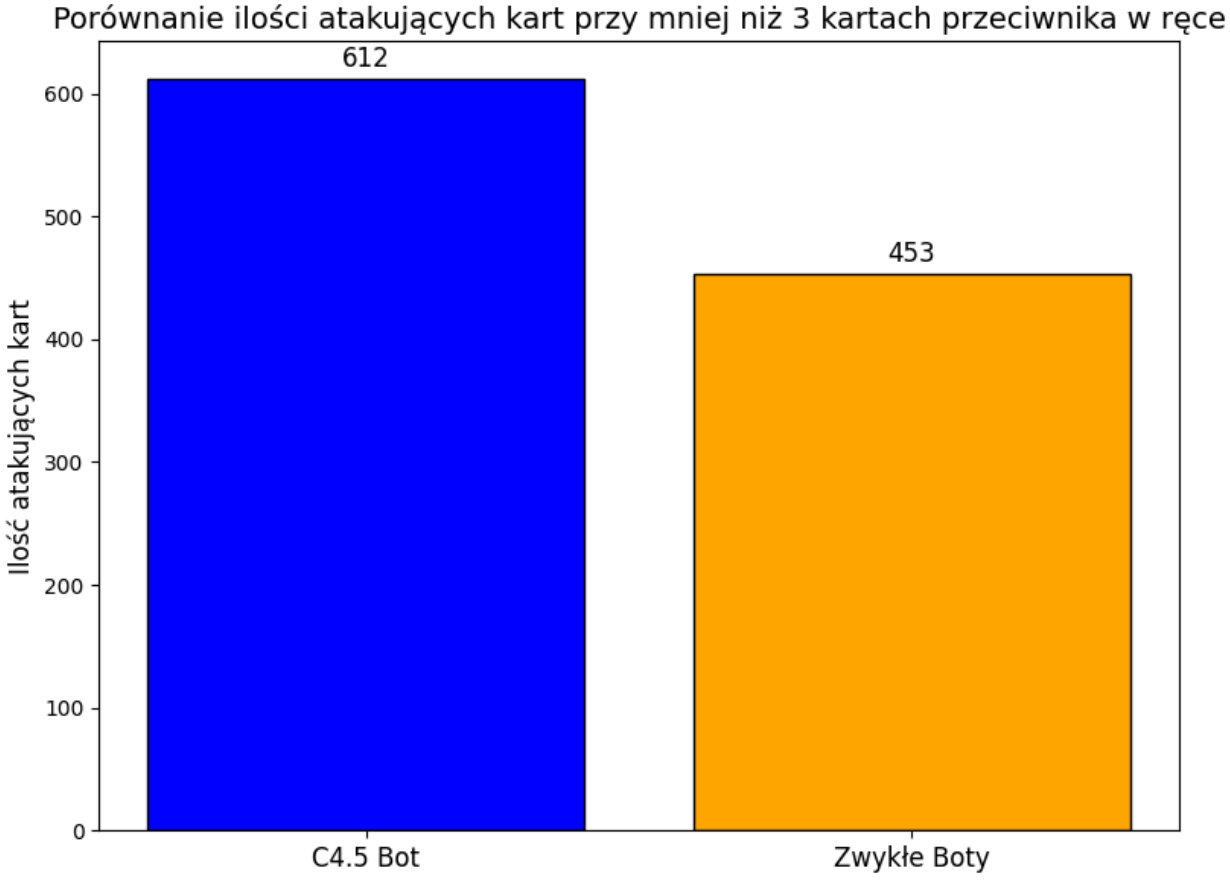
2. **Wpływ losowości na rozgrywkę:**
   Wysoka losowość gry Uno, w tym dobieranie kart z talii, znacząco wpłynęła na wyniki. Różnice w winratio między AI a heurystycznymi botami były widoczne, ale nie drastyczne, co podkreśla znaczenie czynnika losowego.
3. **Efektywność heurystycznych botów:**
   - **RandomBot:** Pomimo swojej losowej natury, osiągał wyniki zbliżone do bardziej zaawansowanych botów.
   - **BasicLogicBot:** Strategia oparta na utrzymywaniu przewagi kolorów w ręce okazała się skuteczna w dłuższej perspektywie, jednak brak zaawansowanej analizy sytuacji ograniczał jego możliwości.
   - **AgressiveBot:** Bot ten często podejmował decyzje zgodne z założeniem maksymalizacji presji na przeciwniku, jednak strategia ta okazała się być najmniej skuteczną spośród wszystkich botów.

6.1.2 Liczba tur między AI a podstawowymi botami



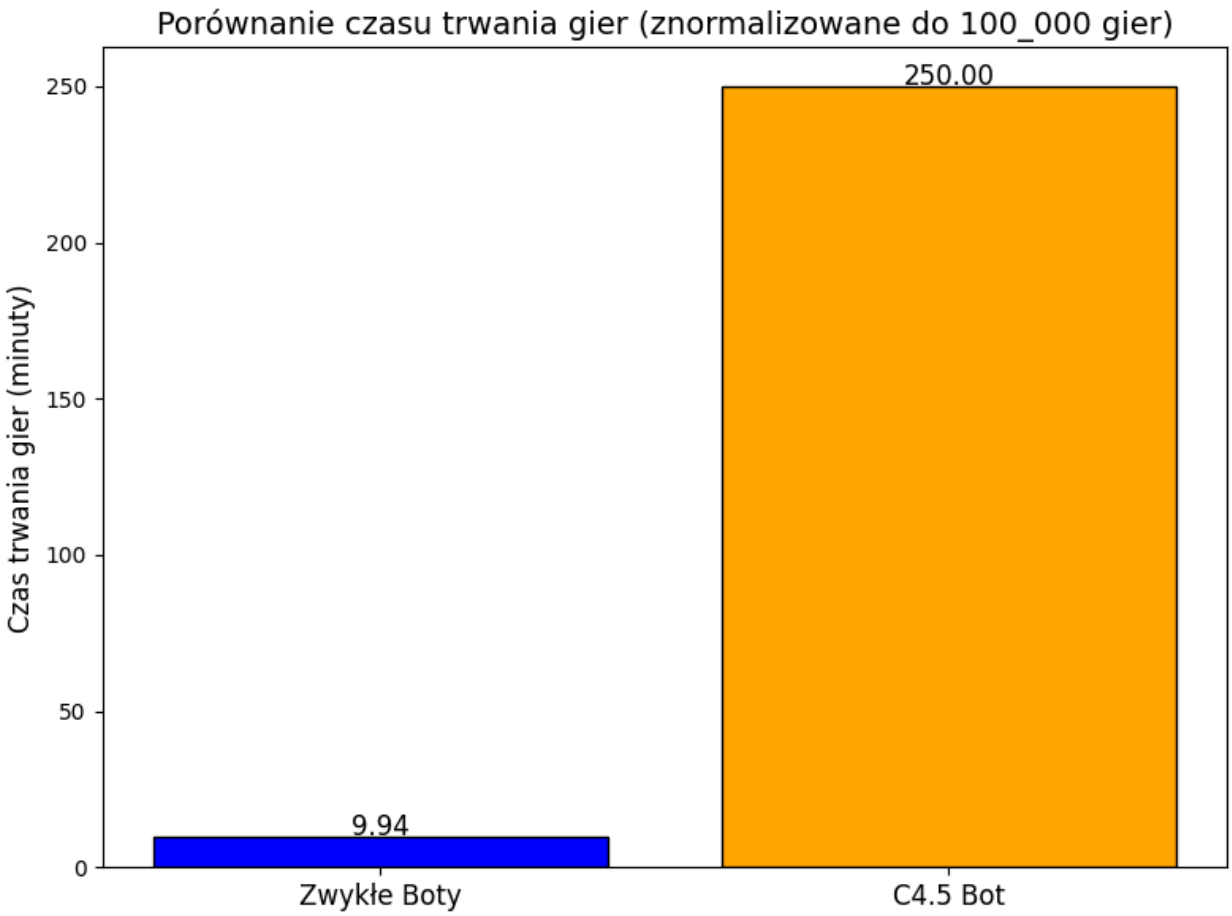
> Rys 6. Porównanie liczby rund algorytmu C4.5 względem zwykłych botów

Analizując liczbę rund w grach Uno, zauważamy, że gry z udziałem bota C4_5 cechują się wyższą maksymalną liczbą rund w porównaniu do gier rozgrywanych między podstawowymi botami. Wskazuje to, że bot C4_5 może prowadzić bardziej złożone rozgrywki, które wymagają dłuższego czasu na zakończenie. Powodowane jest to tym, iż bot C4_5 jest w stanie dopasować strategię gry pod sytuację przeciwnika, dzięki czemu jest w stanie zagrywać częściej karty atakujące kiedy przeciwnik ma mało kart w ręce, w celu minimalizacji jego szansy na wygraną. Co pokazuje poniższy wykres.



> Rys 7. Porównanie ilości atakujących kart przy mniej niż trzech kartach w ręce przeciwnika

6.1.3 Czas działania algorytmu



> Rys 8. Porównanie czasu rozgrywania gier algorytmu C4.5 względem zwykłych botów

Jak widać na powyższym wykresie, gry rozgrywane pomiędzy C4.5 a zwykłymi botami są około 25 razy wolniejsze od gier rozgrywanych przez zwykłe boty między sobą. Oznacza to, że **C4_5Bot** jest około 50 razy wolniejszy od swych podstawowych odpowiedników. Fakt ten nie jest zaskakujący, jako iż podstawowe algorytmy nie analizują dogłębnie stanu gry, podczas gdy Bot AI musi wpierw przejść przez drzewo decyzyjne, a następnie odfiltrować odpowiednie ruchy.

Monte Carlo Tree Search

Niedługo uzupełnię tę sekcję, aczkolwiek już teraz mogę powiedzieć że jest super. ~64% winrate przeciwko podstawowym botom!

Podsumowanie

Sztuczna inteligencja stworzona na potrzeby projektu wykazała, że algorytm C4.5 jest skuteczny w rozwiązywaniu problemów decyzyjnych w grze Uno. Dzięki głębokiej analizie danych oraz zdolności adaptacji strategii do sytuacji na stole, AI osiągnęła wyższy wskaźnik wygranych w porównaniu do podstawowych botów heurystycznych. Pomimo znacznego wpływu losowości, zastosowanie drzewa decyzyjnego pozwoliło na bardziej zaawansowane analizy, co przełożyło się na lepsze wyniki i bardziej strategiczne rozgrywki. Projekt potwierdza, że algorytmy tego typu mogą znaleźć zastosowanie w obszarach wymagających podejmowania decyzji w warunkach niepełnej informacji. Dużą zaletą drzew decyzyjnych jest to, iż należą one do dziedziny algorytmów typu “whie box” co oznacza że można przeanalizować ich działanie. Jednak należy pamiętać że algorytm ten jest stosunkowo wolny w porównaniu do prostych modeli stosujących mało zaawansowane heurestyki. W przypadku przyszłej implementacji warto rozważyć przyspieszenie wykonywania się algorytmu w momencie wyboru ruchu.