Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Blackjack

analiza strategii

Monika Tworek
Nr indeksu: 229776

Praca inżynierska napisana
pod kierunkiem
dr hab. Szymona Żeberskiego

Politechnika
Wrocławska

Wrocław 2018

Spis treści

1 Wstęp **1**

2 Analiza problemu **3**
2.1 Blackjack - zasady . . . 3
2.2 Strategie . . . 3
2.3 Czynniki losowe i wpływ na wynik . . . 4

3 Strategie **5**
3.1 Idealna . . . 5
3.2 Krupierska . . . 6
3.3 Ekspansyjna . . . 6
3.4 Never bust . . . 7
3.5 Prawdopodobna . . . 7
3.6 Filmowa (inaczej Hi-Low) . . . 7
3.7 Intuicyjna . . . 8
3.8 Reaguj na bank . . . 8
3.9 Podstawowa . . . 10
3.10 Przełam passę . . . 10
3.11 Przetrzymaj passę . . . 11
3.12 Zależna od szczęścia . . . 12
3.13 Pasująca . . . 13

4 Projekt aplikacji **15**
4.1 Gra . . . 15
4.2 Generowanie danych . . . 16
4.3 Złożoność obliczeniowa strategii idealnej . . . 17
4.4 Komunikacja z przeglądarką . . . 18
4.5 Interfejs graficzny . . . 19

5 Instalacja i uruchomienie aplikacji **23**
5.1 Wymagania techniczne i pierwsze uruchomienie . . . 23
5.2 Rozgrywka . . . 26

6 Analiza uzyskanych danych **29**
6.1 Próbka badawcza . . . 29
6.2 Podsumowanie . . . 29
6.3 Porównanie wyników na wykresach . . . 29
6.4 Idealna . . . 30
6.5 Krupierska . . . 30
6.6 Ekspansyjna . . . 30
6.7 Never bust . . . 30
6.8 Prawdopodobna . . . 30
6.9 Filmowa inaczej Hi-Low . . . 30
6.10 Intuicyjna . . . 30

6.11 Reaguj na bank . . . 31
6.12 Podstawowa . . . 31
6.13 Przełam passę . . . 31
6.14 Przetrzymaj passę . . . 31
6.15 Zależna od szczęścia . . . 31
6.16 Pasująca . . . 31
6.17 Przykładowe rozdanie . . . 31
6.18 Blackjacki . . . 33
6.19 Przewaga bez blackjacków . . . 33
6.20 Porażki poprzez przekroczeni 21 . . . 33

7 Podsumowanie **35**

Bibliografia **37**

A Zawartość płyty CD **39**

Wstęp

„Ciekawość to pierwszy stopień do piekła lub pracy inżynierskiej”

Prawdopodobnie najbardziej widowiskowe użycie matematyki (a zwłaszcza statystyki) można znaleźć w kasynie. Z dostępnych gier jedną z bardziej popularnych jest Blackjack ze względu na proste zasady. Niniejsza praca zajmuje się analizą strategii, które można stosować w tej grze karcianej oraz opisem powstałej w tym celu aplikacji. W trakcie jej działania można eksperymentalnie sprawdzić przewagę kasyna w najczęściej stosowanych strategiach.

Celem pracy jest:

- Zaprojektowanie i implementacja aplikacji, która będzie się składać z dwóch części:
  - Gra w Blackjacka, w której użytkownik mierzy się przeciwko krupierowi;
  - Moduł badający wyniki stosowania strategii.
- Analiza danych dostarczonych przez aplikację.

Istnieje wiele gotowych programów będących grą w Blackjacka. Żadna z nich nie pozwala na zautomatyzowane testowanie konkretnej strategii, np. za pomocą bota. Gry te są nastawione na zapewnienie rozrywki użytkownikowi, a nie przeprowadzanie analizy statystycznej. Szukając informacji na temat zasad gry można natrafić na artykuły przedstawiające popularne strategie, zachwalające ich skuteczność bez żadnych potwierdzeń popartych danymi, czy choćby zarysu ich analizy. Aplikacja pozwala na rozwiązanie powyższych problemów.

Praca składa się z 5 rozdziałów:
Pierwszy rozdział opisuje zarysowany problem oraz zasady gry.
W drugim rozdziale znajdują się strategie wraz z algorytmami.
W rozdziale trzecim przedstawiono opis programu posiłkując się diagramami UML oraz screenami z aplikacji.
Czwarty rozdział poświęcony jest instalacji i uruchomieniu programu.
Piąty rozdział zajmuje się analizą otrzymanych eksperymentalnie wyników.

Analiza problemu

2.1 Blackjack - zasady

Blackjack to kasynowa odmiana „oczka”, w której gracz mierząc się z krupierem stara się uzyskać wynik w kartach, który jest jak najbardziej zbliżony do 21, ale nie przekracza tej liczby. Każda karta warta jest konkretną liczbę punktów: figury są warte 10 punktów, as jest liczony za 1 albo 11 punktów (wybierana jest wartość korzystniejsza dla gracza), a karty numerowane mają tyle punktów jaką mają wartość liczbową. W talii nie występuje joker. Wynik gracza w rundzie to suma punktów kart, które ma na ręce. Gra rozpoczyna się od stawienia zakładu (w aplikacji wynosi on 10). Następnie krupier otrzymuje jedną zakrytą, a drugą odkrytą kartę, a gracz dwie odkryte.

W początkowym momencie rundy gracz może:

- ubezpieczyć się (insure) - jeżeli widoczną kartą krupiera jest as, gracz może postawić dodatkową stawkę, że krupier ma blackjacka i ten zakład wygrywa się 2 do 1;
- rozdwoić karty (split) - jeżeli karty gracza są tej samej wartości, użytkownik ma możliwość stworzenia dwóch niezależnych gier, wtedy dokładana jest dodatkowa stawka (10), a krupier rozdziela karty gracza na dwie ręce i dokłada do każdej ręki brakującą kartę;
- podwoić stawkę (double down) - gracz stawia dodatkową stawkę (10), otrzymuje dodatkową jedną kartę (trzecią na ręce), a runda zostaje rozwiązana.

Ponadto użytkownik może dobrać kartę (hit) lub przestać dobierać (stand). Ręka przestaje być aktywna po wykonaniu polecenia stand i double down, a runda jest rozwiązywana. Natomiast w przypadku przekroczeniu 21 punktów ręka przestaje być aktywna, gracz automatycznie przegrywa, a runda nie jest rozwiązywana. Rozwiązanie rundy polega na odsłonięciu odwróconej karty krupiera, a następnie krupier dobiera karty tak długo jak ma mniej niż 17 punktów. Po przekroczeniu 17 punktów przestaje dobierać bez względu na ilość punktów u gracza. Jeżeli w trakcie rundy zabraknie kart, to runda jest traktowana jako remis.

Jeżeli gracz ma Blackjacka (tzw. „oczko”)- dłoń wartości 21 przy tylko 2 kartach, to otrzymuje 2.5-krotną stawkę postawioną, dla zwycięstw innych niż blackjack otrzymuje dwukrotność stawki. Dla remisu zwracana jest stawka, czyli gracz wychodzi na 0. W przypadku porażki nie otrzymuje nic.

Nazwa “Blackjack” powstała od nazwy bonusu, który był wypłacany przez kasyna, kiedy gra po raz pierwszy pojawiła się w amerykańskich kasynach. Jeśli karta gracza składała się z asa pik i czarnego waleta (pik lub trefl) gracz otrzymywał dodatkową wygraną wynoszącą 10:1. Pomimo, że bonus w późniejszym etapie został usunięty to nazwa „blackjack” pozostała.

2.2 Strategie

Jest wiele strategii, które można podzielić na trzy kategorie ze względu na to od czego gracz uzależnia swoją decyzję:

- od wcześniejszych decyzji i kart, które zostały użyte,
- tylko od ilości punktów, które ma na ręce,
- od innych czynników, często nieprzewidywalnych.

Program eksperymentalnie sprawdza jakie są wyniki poszczególnych strategii na tej samej talii kart, która została wcześniej przetasowana. Analizie poddano tylko grę jednego użytkownika bez sprawdzania jak zmienia się wygrana użytkownika w zależności od decyzji innych graczy tego samego stołu, którzy w rzeczywistej rozgrywce mają duży wpływ na wynik końcowy. Jedyny czynnik losowy to inna kolejność kart, gdyż krupier działa deterministycznie według strategii takiej jak w kasynach.

Ze względu na bardzo dużą złożoność obliczeniową algorytmu do obliczania strategii idealnej, dla gier z więcej niż 1 talią aplikacja nie oblicza wyniku strategii idealnej.

2.3 Czynniki losowe i wpływ na wynik

Intuicja podpowiada, że część strategii nie sprawdza się zbyt dobrze. Na przykład zbyt mocne dążenie do 21 zwiększa szansę na przekroczenie tej wartości, czyli przegraną. Jeżeli użytkownik się chwilę zastanowi jest w stanie stwierdzić, że strategia krupierska raczej będzie dawać wynik ujemny, bo kiedy gracz przekroczy 21 punktów nie są sprawdzane karty krupiera, który wygrał rundę, więc krupier ma o połowę mniej rund, w których może przegrać.

Wiedząc jakie jest prawdopodobieństwo, że kolejna karta ma żądaną wartość, gracz może podjąć lepsze decyzje. Bez znajomości układu talii jest wręcz niemożliwe każdorazowe wygrywanie gier. Tylko wiedza jakie karty są następne pozwala na bezpieczne dobieranie, w przeciwnym wypadku ryzykujemy przekroczenie przez gracza wartości 21 punktów na ręce albo większej wartości ręki krupiera niż gracza.

Często sugeruje się również, by gracz wybierał stoły z mniejszą liczbą talii. Kiedy w grze jest większa liczba kart sytuacja gracza jest bardziej niestabilna - może się zdarzyć, że będzie wiele niskich kart z rzędu, które w nieprzewidywalnym momencie mogą zmienić się na serię kart wysokich. Ponadto wtedy ciężej jest zapamiętać ile niskich, a ile wysokich kart zostało już użytych.

W aplikacji do tasowania kart użyto wbudowanej funkcji `random.shuffle()` z biblioteki `Random` dostępnej w Pythonie, która korzysta z podstawowego algorytmu Mersenne Twister [6].

W pracy nie rozważono sytuacji kiedy jest więcej niż jeden gracz, bo nie mając wpływu na decyzję osób postronnych trzeba najpierw rozważyć ich strategie i dopasować do nich najbardziej optymalną dla użytkownika. Nawet po dopasowaniu, któryś z graczy może zaburzyć tok rozgrywki po naszej decyzji wpływając na naszą porażkę.

Strategie

3.1 Idealna

Znając układ potasowanej talii, deterministyczną strategię krupiera oraz sprawdzając każdą możliwą decyzję ze strony gracza jesteśmy w stanie znaleźć sekwencje ruchów prowadzących do najlepszego wyniku na koniec gry. Skuteczne znalezienie takiego wyniku może być zrealizowane za pomocą przeszukiwania z nawrotami, które działa następująco:

W pierwszej iteracji wykonywane są same pasy aż skończą się karty. Przed każdym ruchem odkładany jest na stos stan stołu wraz z decyzją „Stand". Uzyskany wynik jest zapisywany jako najlepszy do tej pory. Następnie jest sprawdzane jak będzie się zmieniał wynik końcowy w zależności od zmiany poprzednich „Stand" na „Hit" do momentu, gdy na stosie będą wyłącznie decyzje „Hit". Zapewnia to, że zostania sprawdzana każda sekwencja decyzji. Jeżeli w danym momencie w talii są jeszcze karty należy spauzować. Kiedy karty się skończyły porównywany jest aktualny wynik z najlepszym wynikiem do tej pory i aktualizowany. Następnie ściągany jest ze stosu stan stołu wraz z decyzją dopóki nie zostanie ściągnięty stan stołu z decyzją „Stand". Odkładamy na stos stan stołu z „Hit" i dobieramy kartę. Jeżeli w talii są jeszcze karty, to odkładamy na stos stan stołu z decyzją „Stand" i pauzujemy.

Pseudokod 3.1: Strategia Idealna

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
stack.append(table, "None")
while deck has cards do
    begin_game()
    if players hand is playing then
        stack.append(table, "Stand")
        stand()
besttable ← table
while deck has cards or stack has not only "Hit" do
    if deck hasn't cards then
        if table is better than besttable then
            besttable ← table
        table ← stack.pop()
        while last_choice is "Hit" do
            table ← stack.pop()
        while players hand isn't playing do
            begin_game()
        stack.append(table, "Hit")
        hit()
    else
        while players hand isn't playing do
            begin_game()
        stack.append(table, "Stand")
        stand()
```

Ze względu na dużą złożoność obliczeniową w pracy została zaimplementowana bardziej optymalna wersja z odcięciami. Więcej o tym znajduje się w podrozdziale 4.2.

Ponieważ na wynik końcowy składają się nie tylko zwycięstwa i porażki, ale również remisy i blackjacki. Można to zaobserwować na przykładowym rozdaniu 6.17.

W danej rundzie znając układ tali użytkownik może podjąć decyzje ile kart dobiera, ale nie od której karty rozpoczyna się runda. Poprzednie decyzje wpływają na miejsce rozpoczęcia następnych rund. To powoduje, że konieczne jest wzięcie pod uwagę każdej sekwencji decyzji, które mogą doprowadzić do innych początków kolejnych rund. W tej strategii nie dąży się do zwycięstwa rundy, ale całej gry. Są rozdania (takie jak w przykładzie) kiedy porażka w początkowych rundach daje więcej zwycięstw w późniejszym czasie gry.

Takie uwarunkowanie wyklucza algorytmy o złożoność liniowej, algorytmy zachłanne oraz lokalne optymalizacje algorytmu (dla podciągu talii). Poznanie wszystkich możliwych rozegrań sugeruje zastosowanie przeszukań z nawrotami o dużej złożoności obliczeniowej (analizę złożoności obliczeniowej wykonano w 4.3).

3.2 Krupierska

Jest to ta sama strategia według której gra krupier w kasynie. Należy ona do grupy strategii, w której gracz podejmuje decyzję uzależniając ją od wartości kart na ręce. Jeżeli ręka użytkownika ma wartość 16 punktów lub mniej należy wykonać polecenie Hit. W przeciwnym przypadku, czyli gdy gracz ma 17 oczek i więcej nie dobiera kart.

Pseudokod 3.2: Krupierska

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
while deck has cards do
    begin_game()
    while players hand is playing and players hand.value < 17 do
        hit()
    if players hand is playing then
        stand()
```

3.3 Ekspansyjna

Strategia ta dąży do osiągnięcia wartości jak najbliższej 21. Również należy do grupy strategii uzależniającej decyzję gracza od aktualnie posiadanej liczby punktów na ręce. Gracz dobiera kartę dopóki nie ma co najmniej 20 punktów. Jeżeli wartość ręki wynosi 20 pasuje. W przypadku „oczka” gra jest rozwiązywana.

Pseudokod 3.3: Ekspansyjna

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
while deck has cards do
    begin_game()
    while players hand is playing and players hand.value < 20 do
        hit()
    if players hand is playing then
        stand()
```

3.4 Never bust

Jest bardzo podobna do strategii krupierskiej, czyli również decyzję użytkownika uzależnia tylko od ilości punktów na ręce. Gracz przestaje dobierać karty gdy ma więcej niż 11 punktów. Jeżeli ma 11 punktów lub mniej ciągnie kartę. Jest to strategia, która nie pozwala na przekroczenie 21 punktów. Oznacza to, że krupier każdorazowo krupier musi grać, więc zwiększa się ilość wygranych ze względu na przekroczenie przez krupiera 21 punktów.

Pseudokod 3.4: Never bust

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
while deck has cards do
    begin_game()
    while players hand is playing and players hand.value < 11 do
        hit()
    if players hand is playing then
        stand()
```

3.5 Prawdopodobna

Ta strategia należy do strategii, które swoją decyzję uzależniają od wcześniejszych decyzji i kart, które zostały użyte. W tym wypadku ważne są użyte karty, które obniżają prawdopodobieństwo pojawienia się karty o wartości takiej jak wyciągnięta. Przed rozpoczęciem gry wyciągnięcie karty o konkretnej wartości jest tak samoprawdopodobne. Wiedząc, że w każdej talii są 4 takie karty jest liczone prawdopodobieństwo dla każdej możliwej wartości karty. Po każdej wyciągniętej karcie aktualizowane jest prawdopodobieństwo dla danej wartości liczbowej karty. W momencie, gdy po rozpoczęciu rundy gracz może ciągnąć lub pauzować, to sumowane jest prawdopodobieństwo wyciągnięcia karty, której wartość wraz z obecną sumą kart na ręce gracza nie przekroczy 21. Jeżeli prawdopodobieństwo jest większe niż 50%, to gracz wykonuje „Hit”, a w przeciwnym „Stand”.

Pseudokod 3.5: Prawdopodobna

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
Count_probability(NumberOfDecksOfCard)
while deck has cards do
    begin_game()
    Update_probability()
    while players hand is playing and probability(21 − players hand.value) > 50% do
        hit()
    if players hand is playing then
        stand()
```

3.6 Filmowa (inaczej Hi-Low)

Strategia najbardziej kojarzona ze względu na występowanie w wielu filmach związanych z grą w kasynie polega na liczeniu oczek. Należy do strategii, które decyzję o ciągnięciu uzależniają od wcześniejszych decyzji i użytych kart. Tutaj tak jak w prawdopodobnej ważne są użyte karty, które zmieniają stan stołu. Niskie karty (od 2 do 6) mają wartość +1. Średnie karty (od 7 do 9) są neutralne i mają wartość 0. Pozostałe, wysokie karty mają wartość -1. Przed rozpoczęciem rozdania stół ma wartość 0. Wraz ze wzrostem wyniku rośnie prawdopodobieństwo, że w talii pozostaje więcej wysokich kart i odwrotnie - jeżeli potrzebna jest

niska karta, a wynik jest ujemny, to oznacza, że jest wysokie prawdopodobieństwo karty z przedziału 2-6. Po każdej wyciągniętej karcie aktualizowany jest stan stołu. W momencie, gdy różnica między wartością ręki, a 21 wynosi więcej niż 10 oczek, a stan stołu jest dodatni, to gracz ciągnie kartę. Jeżeli różnica między 21, a ilością oczek na ręce użytkownika jest mniejsza od 11, ale większa od 8, a stan stołu jest w okolicy zera (-1, 0, 1), to użytkownik ciągnie kartę. Jeżeli wartości ręki gracza jest większa niż 13 (czyli różnica między 21, a ilością punktów gracza jest mniejsza od 8), a stan stołu jest ujemny, to gracz ciągnie kartę. W pozostałych wypadkach użytkownik pasuje.

Pseudokod 3.6: Hi-Low

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
while deck has cards do
    begin_game()
    Actual(state, croupiers hand)
    Actual(state, players hand)
    while players hand is playing do
        if 21 − players hand.value > 10 and state > 0 then
            hit()
        if 21 − players hand.value > 8 and state in [−1, 0, 1] then
            hit()
        if 21 − players hand.value > 0 and state < 0 then
            hit()
        else
            stand()
```

3.7 Intuicyjna

Jest to strategia starająca się symulować jak najbardziej grę opierającą się na intuicyjnym podejściu do ciągnięcia lub pasowania. Należy ona do strategii, które decyzję podejmują w zależności od innych, w tym wypadku nieprzewidywalnych czynników. Często intuicyjne podszepty są sugerowane otoczeniem, usłyszanym ostatnio zdaniem bądź zapachem, który aktualnie się unosi. Nie jest możliwe odtworzenie takich warunków, więc w celu przybliżenia różnych czynników, które potrafią się szybko zmienić, losowana jest liczba z przedziału od 0 do 100. Jeżeli wylosowany wynik jest niższy niż 50, to traktuje się to jako podszept duszy mówiącej, by pasować. W przypadku wylosowania liczby większej bądź równej 50 gracz ciągnie kartę.

Pseudokod 3.7: Intuicyjna

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
while deck has cards do
    begin_game()
    number ← Random(0,100) while players hand is playing and random > 50 do
        hit()
    if players hand is playing then
        stand()
```

3.8 Reaguj na bank

Ta strategia należy do trzeciej grupy wymienionej w podrozdziale 2.2 i uzależnia decyzję gracza od widocznej karty krupiera. Jeżeli ma ona wartość 2 lub 3, to gracz pasuje, gdy posiada co najmniej 13 punktów,

a ciągnie jeżeli wartość na ręce nie przekracza 13 oczek. Jeżeli karta krupiera jest z przedziału od 4 do 7, to gracz przestaje dobierać karty, gdy ma co najmniej 17 punktów, a w przypadku kiedy ma więcej niż 17, to pasuje. Dla tych kart krupiera strategia przypomina zasady strategii krupierskiej. W pozostałych przypadkach, czyli kiedy wartość widocznej karty krupiera jest większa niż 7 punktów, to gracz pasuje kiedy ma co najmniej 18 oczek. Kiedy ma mniej niż 18 punktów to ciągnie kartę.

Pseudokod 3.8: Reaguj na bank

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
while deck has cards do
    begin_game()
    if croupier hand.value in [2,3] then
        while players hand.value < 13 do
            hit()
    if croupier hand.value in [4,5,6,7] then
        while players hand.value < 17 do
            hit()
    if croupier hand.value > 7 then
        while players hand.value < 18 do
            hit()
    if players hand is playing then
        stand()
```

3.9 Podstawowa

Jest to strategia, która swoją decyzję uzależnia w pełni od wartości widocznej karty krupiera oraz wartości ręki gracza. Ponadto rozróżnia sytuację, gdy gracz rozpoczyna grę z parą lub gdy jedna z kart to as. Wszystkie zależności pokazuje poniższy rysunek.

Karty gracza	Karty krupiera									
	2	3	4	5	6	7	8	9	10	A
Punkty (bez par i asów)										
5,6,7,8	H	H	H	H	H	H	H	H	H	H
9	H	D	D	D	D	H	H	H	H	H
10	D	D	D	D	D	D	D	D	H	H
11	D	D	D	D	D	D	D	D	D	H
12	H	H	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
17+	S	S	S	S	S	S	S	S	S	S
Pary z asem										
A-2	H	H	H	D	D	H	H	H	H	H
A-3	H	H	H	D	D	H	H	H	H	H
A-4	H	H	D	D	D	H	H	H	H	H
A-5	H	H	D	D	D	H	H	H	H	H
A-6	H	D	D	D	D	H	H	H	H	H
A-7	S	D	D	D	D	S	S	H	H	H
A-8	S	S	S	S	S	S	S	S	S	S
A-9	S	S	S	S	S	S	S	S	S	S
Pary										
A-A	P	P	P	P	P	P	P	P	P	P
2-2	P	P	P	P	P	P	H	H	H	H
3-3	P	P	P	P	P	P	H	H	H	H
4-4	H	H	H	P	P	H	H	H	H	H
5-5	D	D	D	D	D	D	D	D	H	H
6-6	P	P	P	P	P	H	H	H	H	H
7-7	P	P	P	P	P	P	H	H	H	H
8-8	P	P	P	P	P	P	P	P	P	P
9-9	P	P	P	P	P	S	P	P	S	S
10-10	S	S	S	S	S	S	S	S	S	S

Legenda:

H	Hit - dobierz kartę
S	Stand - nie dobieraj kart
D	Double down - podwój stawkę
P	Split - rozdziel karty

Rysunek 3.1: Strategia podstawowa

3.10 Przełam passę

Strategia ta jest uzależniona od wcześniejszych decyzji gracza, a dokładniej jego wyników. Może być traktowana jako strategia „odbicia się od dna”. Na początku gracz kiedy ma mniej niż 15 punktów ciągnie

kartę. W momencie, gdy wartość jego ręki przekroczy 15, a on wciąż może grać, to pasuje. Jednak kiedy miał co najmniej 3 porażki pod rząd, to najmniejsza wartość ręki, która zmienia decyzję gracza z Hit na Stand wzrasta do 17 punktów. Jeżeli więc gracz miał powyżej 3 porażek pod rząd, to kiedy ma 16 oczek bądź mniej to ciągnie kolejną kartą, a gdy ma 17 punktów i więcej to pasuje. Jeżeli gracz miał co najmniej 5 porażek z rzędu, to pasuje dopiero kiedy ma 19 oczek lub więcej. Jeżeli wartość ręki ma poniżej tej liczby, to ciągnie kartę. Po każdym zwycięstwie licznik porażek jest zerowany.

Pseudokod 3.9: Przełam passę

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
state ← 0
while deck has cards do
    begin_game()
    if state < 3 then
        if players hand.value < 15 then
            hit()
        else
            stand()
    if statein[3, 4] then
        if players hand.value < 17 then
            hit()
        else
            stand()
    if state > 4 then
        if players hand.value < 19 then
            hit()
        else
            stand()
    if winner is Croupier then
        state ← state + 1
    else
        state ← 0
```

3.11 Przetrzymaj passę

Strategia ta jest bardzo podobna do Przełam passę i również opiera się na wcześniejszych decyzjach gracza, a w tym przypadku wygranych. W przypadku tej strategii jednak jest obniżany próg od kiedy gracz zaczyna pasować, a nie podwyższany. Na początku gracz przestaje dobierać jak ma co najmniej 17 punktów, czyli zaczyna się tak jak w przypadku strategii krupierskiej. W przypadku co najmniej 3 porażek z rzędu, to użytkownik przestaje dobierać jeżeli ma 14 oczek lub więcej. Jeżeli ma co najwyżej 13 punktów, to ciągnie kartę. Jeżeli gracz ma co najmniej 5 porażek, to pasuje, gdy ma powyżej 10 punktów. Kiedy ma mniej, to ciągnie kartę. Po każdym zwycięstwie zerowany jest licznik porażek.

Pseudokod 3.10: Przetrzymaj passę

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
state ← 0 while deck has cards do
    begin_game()
    if state < 3 then
        if players hand.value < 17 then
            hit()
        else
            stand()
    if statein[3, 4] then
        if players hand.value < 14 then
            hit()
        else
            stand()
    if state > 4 then
        if players hand.value < 11 then
            hit()
        else
            stand()
    if winner is Croupier then
        state ← state + 1
    else
        state ← 0
```

3.12 Zależna od szczęścia

Ta strategia przypomina strategie Przełam i Przetrzymaj passę. Na początku użytkownik przestaje ciągnąć karty jeżeli ma co najmniej 15 punktów. Gdy ma co najwyżej 14 oczek ciągnie kartę. Następnie licznik jest zwiększany z każdym zwycięstwem o 1 aż do 6, a z każdą porażką jest obniżany o 1 aż do -4. W zależności od stanu licznika zmienia się granica po której gracz przestaje ciągnąć karty. Wraz z każdym zwycięstwem jest zwiększana o 1 aż do 20, a z każdą porażką jest obniżana o 1 aż do 11 oczek na ręce gracza. Dobrze to obrazuje poniższa tabelka:

stan licznika	granica
-4	11
-3	12
-2	13
-1	14
0	15
1	16
2	17
3	18
4	19
5	20

Gdzie granica oznacza liczbę, która musi być większa od wartości ręki gracza, by ten mógł dalej ciągnąć kartę.

Pseudokod 3.11: Zależna od szczęścia

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
state ← 0
depend ← [−4 ← 11
 − 3 ← 12
 − 2 ← 13
 − 1 ← 14
 0 ← 15
 1 ← 16
 2 ← 17
 3 ← 18
 4 ← 19
 5 ← 20
 ]
while deck has cards do
    begin_game()
    if players hand.value < depend[state] then
        hit()
    else
        stand()
    if winner is Croupier then
        if state > −4 then
            state ← state − 1
    else
        if state < 6 then
            state ← state + 1
```

3.13 Pasująca

Jest to strategia, która zalicza się do strategii uzależnionych od innych, nieprzewidzianych czynników. W tej strategii gracz każdorazowo po rozpoczęciu gry pasuje bez względu na stan stołu, czy wartości ręki. W przypadku tej strategii oczekuje się, że będzie się miało więcej punktów niż krupier lub krupier podczas ciągnięcia kart przekroczy 21, co spowoduje wygraną gracza.

Pseudokod 3.12: Pasuja

```
Input: Stół table
Output: Wynik [win, draw, loss, blackjack, score]
while deck has cards do
    begin_game()
    if players hand is playing then
        stand()
```

Projekt aplikacji

4.1 Gra

Pierwszym krokiem do stworzenia aplikacji było umożliwienie użytkownikowi zwykłej gry. Tego typu zadania można łatwo zrealizować w języku obiektowym. Języki skryptowe pozwalają na szybkie prototypowanie i rozwój programów. Do stworzenia aplikacji użyto język skryptowy python, który jest obiektowy. Wykorzystanie obiektowości widać na załączonym niżej diagramie UML, który ukazuje zależność między klasami:

Rysunek 4.1: Diagram klas UML odpowiadających za grę w Blackjacka

Na tym etapie rozwoju aplikacji zaimplementowano zasady gry opisane w rozdziale 2. Funkcje dobierania karty oraz pasowania rozdzielono na pierwszą i drugą rękę gracza - `hit1()`, `hit2()`, `stand1()` i `stand2()`.

4.2 Generowanie danych

W celu uzyskania wyników poszczególnych strategii można wygenerować wyniki dla podanej liczby talii. Można zrobić to z poziomu ekranu głównego, za pomocą klawisza „Generuj”. Po skończeniu obliczeń, użytkownik zostaje przeniesiony do poniższego widoku:

Rysunek 4.2: Widok po wygenerowaniu danych

Po otwarciu podanego folderu (C:\User\Public) można znaleźć tam plik tekstowy o nazwie zgodnej z podaną przez aplikację strukturą (np. score_DeckOfCard1_2018-12-27.txt) o następującej zawartości:

```
score(name='Ekspansyjna', winnings=3, draw=0, loosing=5, money=-20, blackjack=0)
score(name='ReagujNaBank', winnings=4, draw=0, loosing=5, money=-10, blackjack=0)
score(name='HiLow', winnings=2, draw=0, loosing=8, money=-60, blackjack=0)
score(name='Intuicyjna', winnings=0, draw=3, loosing=8, money=-80, blackjack=0)
score(name='Krupierska', winnings=4, draw=0, loosing=5, money=-10, blackjack=0)
score(name='NeverBust', winnings=2, draw=1, loosing=8, money=-60, blackjack=0)
score(name='Prawdopodobna', winnings=2, draw=1, loosing=8, money=-60, blackjack=0)
score(name='PrzelamPasse', winnings=6, draw=0, loosing=3, money=30, blackjack=0)
score(name='PrzetrzymajPasse', winnings=4, draw=0, loosing=5, money=-10, blackjack=0)
score(name='ZaleznaOdSzczescia', winnings=5, draw=0, loosing=4, money=10, blackjack=0)
score(name='Podstawowa', winnings=5, draw=1, loosing=4, money=10, blackjack=0)
score(name='Pasujaca', winnings=2, draw=2, loosing=7, money=-50, blackjack=0)
score(name='Idealna', winnings=6, draw=1, loosing=2, money=40, blackjack=0)
```

Rysunek 4.3: Wygenerowane dane

W pliku zostały zapisane wyniki poszczególnych strategii z nazwami wraz z uwzględnieniem ile rund użytkownik by zwyciężył stosując się do danej strategii, ile razy przegrał, ile rund zakończyłoby się remisem, a ile razy padł blackjack.

Generowanie danych zostało dodane jako osobny moduł obok gry omówionej w poprzednim podrozdziale. W tym module zaimplementowano strategie, których pseudokod znajduje się w rozdziale 3. Każda ze strategii jest wywołana z kopią tej samej tablicy, czyli otrzymuje ten sam układ kart. Jednak ze względu na bardzo dużą złożoność obliczeniową zoptymalizowano strategię idealną dodając obostrzenia przed ruchami. Wiedząc ile kart zostało i ile jest jeszcze nie użytych asów w talii (zmienna `aces`) porównuje się hipotetycznie najlepszy od tego momentu wynik z wynikiem najlepszego stołu. Jeżeli jest on gorszy to ustawia się flagę `is_finished` na $True$ traktując daną sytuację jakby skończyły się karty i należy przejść do kolejnej gałęzi drzewa nawrotów.

Pseudokod 4.1: Strategia Idealna zoptymalizowana

```
Input: Tablica table
Output: Wynik [win, draw, loss, blackjack, score]
stack.append(table, "None")
while deck has cards do
    begin_game()
    if players hand is playing then
        stack.append(table, "Stand") stand()
besttable ← table
while deck has cards or not is_finished or stack has not only "Hit" do
    if deck hasn't cards or is_finished then
        if table is better than besttable then
            besttable ← table
        table ← stack.pop()
        while last_choice is "Hit" do
            table ← stack.pop()
        while players hand isn't playing do
            begin_game()
        if score + 1.5 * aces + (|decks.cards| − 2 * aces) * 0.25 < bestscore then
            table.is_finished ← True
        else
            stack.append(table, "Hit")
            hit()
    else
        while players hand isn't playing do
            begin_game()
        if score + 1.5 * aces + (|decks.cards| − 2 * aces) * 0.25 < bestsscore then
            is_finished ← True
        else
            stack.append(table, "Stand")
            stand()
```

Pomimo optymalizacji strategia idealna ma wciąż bardzo dużą złożoność obliczeniową. Z tego względu zarówno podczas generowania jak i uzyskania wyników strategii dla porównania gry użytkownika powyżej 1 talii strategia idealna nie jest obliczana.

4.3 Złożoność obliczeniowa strategii idealnej

Podjęcie decyzji użytkownika wiąże się z odłożeniem na stos informacji o stanie gry. Jest to najbardziej kosztowna operacja w czasie wykonywania algorytmu, dlatego względem niej jest liczona złożoność obliczeniowa.

Średnia wartość karty to 6,5385, czyli krupier ma średnio 13,0769 punktów na ręce przy rozpoczęciu rundy. Przeciętnie krupier dobiera od 0 do 2 kart. Korzystając z aplikacji można to dość szybko potwierdzić.

Podobna sytuacja jest po stronie użytkownika. Co daje około 6 kart, które ubywają co rundę przy podjęciu jednej z dwóch decyzji gracza. Jeśli użytkownik gra na M talii (czyli $N = 52 * M$ kart), to podejmuje w czasie gry $\frac{N}{6}$ decyzji, co daje $2^{\frac{N}{6}}$ możliwych rozegrań gry, z których wszystkie należy rozpatrzyć.W średnim przypadku otrzymujemy złożoność wykładniczą.

Dla każdej rundy ubywa z talii `k` kart, gdzie $40 \geqslant k \geqslant 4$. Co najmniej 4 karty zostają użyte ponieważ na rozpoczęcie rundy krupier otrzymuje 2 karty (które mogą mieć wartość wyższą niż 17) i gracz otrzymuje 2 karty. W najgorszym przypadku może zejść aż 40 kart. Dzieje się tak kiedy w grze jest co najmniej 10 talii kart. Wtedy gracz może otrzymać 10 asów, następnie dwójkę i 9 asów, a krupier otrzymuje 10 asów, dwójkę i 5 asów.

Aby oszacować złożoność obliczeniową w najlepszym przypadku należy wyliczyć najmniejszy współczynnik $r = \frac{d}{k}$, który jest stosunkiem liczby decyzji na liczbę użytych kart w czasie rundy. Niskie `r` oznacza małą liczbę decyzji w przebiegu całej gry. Najmniejszy taki współczynnik to $\frac{1}{19}$ dla co najmniej 4 talii kart, kiedy na jedną decyzję użytkownika krupier jest zmuszony pociągnąć aż 15 kart. Współczynnik ten jest zależny od liczby talii kart w grze:

liczba talii=M	liczba kart=N	współczynnik r	liczba decyzji na grę = r*N
1	52	0,083	4,333
2	104	0,067	6,933
3	156	0,059	9,177
4	208	0,053	10,947
5	260	0,053	13,684
6	312	0,053	16,421

Powyżej 4 talii współczynnik r jest już stały, ale zwiększa się liczba kart w talii. Ponieważ użytkownik ma 2 możliwe decyzje („Hit” i „Stand”), więc rozpatrując najlepszą możliwą sytuację, dla 1 talii jest $2^{4,33}$ możliwych sekwencji decyzji. W tym szacowaniu nie wzięto pod uwagę, czy użytkownik dobiera kartę, czy pasuje. Pasowanie powoduje, że z talii ubywa co najmniej 4 karty. Natomiast ciągnięcie zmniejsza talię tylko o 1 kartę (wyjątkiem jest sytuacja, gdy pociągnięta karta dopełnia sumę ręki do „oczka” i runda jest rozwiązywana lub przekracza 21 - wtedy krupier nie dobiera kart), co powoduje, że zwiększa się r (liczba decyzji na rundę). Dlatego dobieranie kart zwiększa liczbę decyzji na grę i zwiększa się liczba decyzji na grę, co zwiększa złożoność obliczeniową.

Powyższe oszacowanie jest mniejsze niż dla najmniejszej realnej złożoności obliczeniowej - nie uwzględniamy, że asy mogą się skończyć, dobranie karty zwiększa liczbę decyzji na grę, a krupier nie otrzymuje każdorazowo kart wyższych 3 (powyżej 1 talii nie otrzymuje kart wyższych niż 2). Złożoność obliczeniowa dla tej strategii jest co najmniej wykładnicza.

4.4 Komunikacja z przeglądarką

Ze względu na to, że interfejs jest obsługiwany przez Angulara, a serwer do obliczeń jest postawiony przy użyciu pythona należało rozwiązać problem związany z przesyłem zapytań z różnych źródeł.

Każda strona wyświetla w przeglądarce ma przypisany konkretny origin, czyli protokół, host i port. Jeżeli strona pobiera zasoby ze źródła o innym originie zachodzi podejrzenie cyber ataku. Aby złośliwa aplikacja nie spowodowała wycieku wrażliwych danych czy transferu z konta bankowego użytkownika stosowany jest w przeglądarkach mechanizm obronny Same-Origin Policy. SOP w swoich założeniach zapewnia, że dwie aplikacje o różnych originach nie mogą współdzielić żadnych danych. Wraz z rozwojem internetu i technologii powstały aplikacje, które korzystają z usług świadczonych przez różne serwery (różne originy). SOP uniemożliwia działanie takich programów w przeglądarce. Przykładem takiego programu jest interfejs w Angularze, który odpytuje serwer w pythonie mający inny origin.

Aby umożliwić działanie program tego typu opublikowano w 2014 roku wraz z rekomendacją W3C standard Cross-Origin Resource Sharing. Pozwola on na uzupełnienie protokołu HTTP o rozszerzenie nagłówków, które informują czy i w jaki sposób może nastąpić dostęp do zasobów. Korzystając z CORS warto dodać dodatkowe walidacje po stronie serwera, gdyż korzystanie z tego standardu tworzy lukę w zabezpieczeniach SOP.

W trakcie tworzenia tej aplikacji zastosowano rozwiązanie polegające na zainstalowaniu w przeglądarce

rozszerzenia, które wykorzystuje politykę CORS, jednak nawet przed uruchomieniem widać ostrzeżenie, że to narzędzie należy wyłączyć zaraz po zakończeniu użytkowania ze względów bezpieczeństwa.

4.5 Interfejs graficzny

Interfejs graficzny został stworzony w oparciu o framework Angular [1], który w domyślnej konfiguracji wykorzystuje dodatkowe zależności, m.in. język TypeScript [11] oraz bibliotekę RxJS [12] (ułatwiającą asynchroniczną obsługę eventów). Są to zewnętrzne zależności i ich kod źródłowy nie jest modyfikowany przez programistę. Dobrą praktyką jest nie umieszczanie zawartości bibliotek bezpośrednio w repozytorium. Zamiast tego wykorzystywany jest Node Package Manager [5], który dokłada zależności w procesie instalacji.

Angular działa według popularnego schematu tworzenia aplikacji webowych. Udostępnia on szablony widoków i komponentów, które po uzupełnieniu przez programistę przekładają się na od razu widoczne efekty.

Poza komponentami interferjs składa się jeszcze z serwisów, które są odpowiedzialne za komunikację z serwerem (pythonowym). Komunikacja odbywa się za pomocą zapytań XHR (XMLHttpRequest), które są obsługiwane przez przeglądarkę asynchronicznie. Dokonuje się tam obsługa odpowiedzi na zapytania. Reactive programming to paradygmat, który powstał, by ułatwić tego typu zadania. Jego implementacją jest RxJS, która jest promowana przez twórców Angulara.

Poniższy diagram UML pokazuje wyłącznie komponenty i serwisy stworzone na potrzeby aplikacji:

Rysunek 4.4: Diagram klas UML interfejsu

Głównym celem aplikacji było umożliwienie porównania wyniku gracza i poszczególnych strategii. Poza wynikiem końcowym użytkownik może dokonać również analizy wygranych (w tym ile jest blackjacków), remisów i porażek. Strategie są podawane w kolejności w jakiej występują w opisie, który można zobaczyć na rysunku 5.8. W przejrzysty sposób można zobaczyć, która strategia osiąga dla danej rozgrywki największą liczbę zwycięstw, blackjacków, czy najwyższy wynik końcowy. Znając sposób działania podobnych sobie strategii (np. NeverBust i Krupierskiej, które różnią się wyłącznie wartością ręki, po której przekroczeniu użytkownik pasuje) można zauważyć, która wersja jest lepsza dla danej kolejności kart.

Jedną z ważniejszych zasad przy tworzeniu frontendu jest dobór odpowiedniego koloru, czcionki, czy tła. Tło, który jest na każdym ekranie sugeruje użytkownikowi czym zajmuje się aplikacja nawiązując do zielonego stołu jaki jest używany w kasynach lub koloru rewersu kart. Aby nie przykrywać tła, a jednocześnie aby tekst był czytelny zastosowano półprzeźroczyste wypełnienie komponentów.

NAZWA	ZWYCIĘSTWA	REMISY	PORAŻKI	BLACKJACKI	WYNIK
Ekspansyjna	8	2	8	1	5
ReagujNaBank	8	3	7	1	15
HiLow	3	0	16	0	-130
Intuicyjna	5	1	14	1	-85
Krupierska	8	5	6	1	25
NeverBust	7	3	11	1	-35
Prawdopodobna	9	1	8	0	10
PrzelamPasse	6	3	10	1	-35
PrzetrzymajPasse	8	5	6	1	25
Podstawowa	10	0	9	2	20
ZaleznaOdSzczescia	7	2	9	1	-15
Pasujaca	5	1	16	1	-105

Rysunek 4.5: Porównanie wyników gracza i poszczególnych strategii

Wykorzystano również możliwość nadania innego koloru przyciskom i tłom komponentów po najechaniu na nie kursorem. Wybór kolorów pasujących do siebie jest niezwykle ważny. Przy złym doborze kolory zlewają się, są nieczytelne lub rozpraszają uwagę użytkownika. Aby uniknąć takich błędów stosuje się koło kolorów [3]. Pozwala ono na dobór kolorów monochromatycznych (zbliżających się na tym samym promieniu do środka koła), pokrewnych (sąsiadujące ze sobą) bądź dopełniających się (leżące naprzeciw siebie). Ponieważ celem jest wyróżnienie wybranej strategii zastosowano kolor dopełniający się z kolorem tła. Czarny kolor czcionki jest kolorem dopełniającym do każdego koloru tak samo jak biały, można więc bezpiecznie łączyć go z dowolnym tłem.

NAZWA	ZWYCIĘSTWA	REMISY	PORAŻKI	BLACKJACKI	WYNIK
Ekspansyjna	8	2	8	1	5
ReagujNaBank	8	3	7	1	15
HiLow	3	0	16	0	-130
Intuicyjna	5	1	14	1	-85
Krupierska	8	5	6	1	25
NeverBust	7	3	11	1	-35
Prawdopodobna	9	1	8	0	10
PrzelamPasse	6	3	10	1	-35
PrzetrzymajPasse	8	5	6	1	25
Podstawowa	10	0	9	2	20
ZaleznaOdSzczescia	7	2	9	1	-15
Pasujaca	5	1	16	1	-105

Rysunek 4.6: Ukazanie doboru kolorów pasujących do siebie

Instalacja i uruchomienie aplikacji

5.1 Wymagania techniczne i pierwsze uruchomienie

Praca była pisana na systemie Windows 10 z wykorzystaniem Pythona 3.6.2 [6] oraz Angulara 7.0.6 [1] (Node 8.11.2), a używana przeglądarka to chrome.

Ponieważ aplikacja została napisana za pomocą pythona wymagane jest, by użytkownik posiadał interpreter do Pythona 3.x.x. Najbezpieczniej jest pobrać ze strony autora [7]. W celu sprawdzenia, czy zainstalowano wystarczy w terminalu wpisać python. Po lewej stronie widać, że system nie rozpoznaje polecenia i należy go zainstalować:

Rysunek 5.1: Porównanie wyników terminali

Nie należy zapomnieć o dodaniu pythona do PATH - bez tego program nie działa poprawnie. Warto to zaznaczyć w instalatorze:

Rysunek 5.2: Instalator z zaznaczonym dodaniem Python do PATH

W programie użyto bibliotek, które wymagają dodatkowej instalacji - jsonscheme [9], flask [8] oraz namedlist [10]. Instaluje się je wpisując w terminal następujące polecenie:

```
$ pip install flask
$ pip install jsonscheme
$ pip install namedlist
```

Może się okazać, że należy zauktualizować pipa (poleceniem „python -m pip install –upgrade pip”).
W celu uruchomienia backendowego serwera należy w terminalu w folderze z Blackjack wywołać polecenie:

```
$ python blackjack.py
```

W tym momencie uruchomi się serwer, co zostanie wyświetlone na terminalu:

```
 * Serving Flask app "blackjack.server" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://localhost:5000/ (Press CTRL+C to quit)
```

Rysunek 5.3: Widok terminala po uruchomieniu serwera pythonowego

Następnie należy zainstalować środowisko do Angulara aby móc zobaczyć jak program działa w przeglądarce. W tym celu należy mieć zainstalowane następujące narzędzia: git [2], npm [5] i nodejs [4]. Czasami by system je „zauważył” po instalacji należy ponownie uruchomić komputer.
W celu zainstalowania Angulara należy wykonać w terminalu polecenie:

```
$ npm install -g @angular/cli
```

Następnie w folderze angular-client w terminalu wywołać polecenie:

```
$ npm install
$ npm rebuild node-sass
$ ng serve
```

Po uruchomieniu można zobaczyć, czy można już korzystać w przeglądarce z programu:

```
                          angular-client> ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

Date: 2018-12-23T18:10:51.455Z
Hash: ab3a798a8c0c20d0b1ea
Time: 20086ms
chunk {main} main.js, main.js.map (main) 83.7 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 223 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 32 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.31 MB [initial] [rendered]
i ｢wdm｣: Compiled successfully.
```

Rysunek 5.4: Widok terminala po poprawnej konfiguracji i uruchomieniu serwera angularowego

Kiedy użytkownik uruchomi pythonowy serwer, serwer angularowy i wejdzie na localhost:4200 w przeglądarce pojawi się strona startowa aplikacji:

Rysunek 5.5: Widok przeglądarki ze stroną startową aplikacji

Niestety ponieważ dane są na serwerze localhost:5000, a użytkowniko korzysta z portu 4200, to jest to sprzeczne ze standardem SOP. Ten problem został omówiony w rozdziale w 4.4. Aby móc wygenerować dane lub rozpocząć grę należy dodać rozszerzenie do przeglądarki, które doda do zapytań nagłówki zgodne ze standardem CORS. W przypadku przeglądarki chrome jest to rozszerzenie "Moesif Origin & CORS Changer"w wersji 0.2.11:

Rysunek 5.6: Włączone rozszerzenie wyłączające politykę CORS

5.2 Rozgrywka

Po wybraniu w oknie startowym liczby talii użytkownik może albo wygenerować dane albo rozpocząć rozgrywkę. W przypadku generowania danych użytkownik zostanie przeniesiony do innego okna wraz z informacja gdzie znajdują się wygenerowane dane i w jakim formacie. Po naciśnięciu przycisku „NOWA GRA” zostanie rozpoczęta automatycznie nowa rozgrywka. Przyciski po lewej stronie odpowiadają akcjom, które gracz może wykonać podczas normalnej gry w kasynie. Po zakończeniu rundy na środku ekranu pojawi się komunikat wraz z przyciskiem umożliwiającym rozpoczęcie następnej rundy. Natomiast w przypadku skończenia się kart w talii przycisk pod komunikatem przenosi użytkownika do ekranu z wynikami. Ekran z wynikami jest przedstawiony na rysunku 4.5. Tak jak zostało to już wyjaśnione w rozdziale 4.1 w grze rozdzielono „Hit” i „Stand” na osobne polecenia w zależności od ręki. Jak można zauważyć na rysunku poniżej obie ręce są podpisane:

Rysunek 5.7: Widok w trakcie gry użytkownika przeciwko krupierowi

Po zakończonej rozgrywce pojawia się okienko z wynikiem rundy oraz przycisk umożliwiający rozpoczęcie nowej rundy. Jeżeli skończą się karty, to wyświetla się okienko informujące o tym. Po kliknięciu w „OK” aplikacja pokazuje zestawienie wyników - gracza oraz strategii. Wszystkie potrzebne informacje o zasadach gry, ruchach oraz krótki opis strategii znajduje się w zasadach:

Rysunek 5.8: Potrzebne informacje i zasady gry

Analiza uzyskanych danych

6.1 Próbka badawcza

Dla 1 talii jest 52! różnych ułożeń kart. W przybliżeniu to $8.0658 * 10^{67}$. Jest to ogromna liczba - dla porównania wiek obserwowalnego wszechświata to „zaledwie” 10^{18} sekund. Jest więc niemożliwe, by dla 1 talii wyliczyć wynik wszystkich strategii dla każdego rozdania. W przypadku 2 talii ta liczba to $\frac{104!}{2^{52}}$, co daje ok. $2{,}3 * 10^{150}$, a przy 3 ($\frac{(52*3)!}{3!^{52}}$, czyli $2{,}5 * 10^{235}$) i 4 talii ($\frac{(52*4)!}{4!^{52}} \approx 4 * 10^{321}$) ciężko jest nawet znaleźć obrazowe przybliżenie wielkości tej liczby. W przypadku gry Blackjacka nie trzeba rozróżniać, czy otrzymana karta o wartości 7 jest koloru kier, karo, trefl, czy pik, więc liczba rozdań zdecydowanie się zmniejsza. Wtedy każda talia to nie 52 rozróżnialne karty, więc otrzymujemy $\frac{(13*4)!}{4!^{13}}$ kombinacji co daje nam „tylko” $9{,}2 * 10^{49}$. Ponieważ nie jesteśmy sprawdzić wszystkich możliwych rozdań zostało to sprawdzone dla XX gier z użyciem 1 talii, YY dla 2 talii oraz odpowiednio ZZ i AA dla 3 i 4 talii.

(Mądry komentarz dlaczego akurat tyle.)

6.2 Podsumowanie

Ogólna przewaga kasyna w ilości zwycięstw gracza do jego porażek waha się między 20% a 30%. W skrajnych przypadkach wynosi ona od 50% do nawet 70%. Jest część strategii, które mają większą liczbę zwycięstw niż porażek i wychodzą na plus - Przełam Passę i Podstawowa. Czasami zdarzają się ciekawe rozdania, gdy wszystkie strategie poza idealną wychodzą na minus lub gdy dla danego rozdania jakaś strategia daje ten sam wynik co idealna (w czasie zbierania danych wydarzyło się tak dla PrzełamPassę). Co ciekawe największą liczbę blackjacków miała strategia Intuicyjna. Czasami były rozdania, gdy tylko jedna strategia, która w ogólnym rozrachunku jest na dużym minusie była jedyną strategią (poza idealną) wychodzącą na plus po zakończeniu gry.
Można zauważyć, że zdecydowanie lepsze są strategie bardziej defensywne (Pasująca, NeverBust) lub opierająca się na prawdopodobieństwie wyciągnięcia odpowiedniej karty (Podstawowa, Prawdopodobna). Są strategie, które mają podobne wyniki co Intuicyjna (Ekspansyjna), co wskazuje na to, że ich zasady nie są dopasowane do Blackjacka.
Warto zwrócić uwagę na to, że po przekroczeniu przez gracza 21 punktów krupier nie musi grać. Gdyby ta zasada została zmieniona, to wiele porażek zamieniłoby się w remis, bo wartości ręki krupiera często by przekraczała 21 oczek. Jest to jeden z argumentów dlaczego kasynom opłaca się mieć tę prostą grę w swojej ofercie.

6.3 Porównanie wyników na wykresach

Tutaj są wykresy i odpowiedni komentarz do nich. Wykres dla 1 talii. Wykres dla 2 talii. Wykres dla 3 talii. Wykres dla 4 talii.

6.4 Idealna

W przypadku rozgrywek dla jednej talii strategia ta osiągała niemal brak przegranych, a wynik różnił się tylko ilością blackjacków dla danego rozdaniu. Wraz ze zwiększeniem ilości talli zmniejszyła się też skuteczność gry, co pokazuje, że często są rozdania, gdy nawet pomimo znania układu talii gracz nie jest w stanie całkowicie ograć kasyna. Można też zauważyć, że często opłaca się raz przegrać rundę, by potem móc wygrać 7 rund z rzędu niż starać się zwyciężać każdą rundę.

6.5 Krupierska

Pomimo stosowania tej samej strategii co krupier często wychodzi się przy tej strategii na minus. Pokazuje to, że 17 jest zbyt wysoką granicą dla bezpiecznej gry przeciwko kasynu.

(puścić dla 1 talii ile razy przegrywamy, bo krupier jest zły, a ile my przekroczyliśmy)

6.6 Ekspansyjna

Ta strategia uzyskała najgorszy wynik z wszystkich analizowanych. Jednocześnie jest jedną z najczęściej stosowanych strategii przez początkowych graczy, dzięki czemu kasyna wygrywają ogromne kwoty. W przypadku nieco ponad 300 rund statystycznie gracz traci około 1400 złotych. Jest to jednocześnie strategia wygrywająca najmniej rund poprzez blackjacka.

6.7 Never bust

Strategia ta bardzo podobna do strategii krupierskiej, ale mająca niższą granicę po której gracz pasuje wbrew pozorom osiąga lepsze wyniki. Nie są one drastycznie lepsze (różnica około 2%), ale pokazuje, że gdy gracz waha się między wyższym bądź niższym progiem do pasowania warto wybrać niższy.

6.8 Prawdopodobna

Znajduje się ona w czołówce trzech najlepszy strategii bez znajomości układu kart obok Podstawowej i Przełam Passę. Zarówno ta jak i Podstawowa strategia bazują na prawdopodobieństwu nie przekroczenia 21 jednak Podstawowa zachowuje pewien margines, co pozwala jej uzyskać lepsze wyniki. W tej strategii staramy się jak najbezpieczniej zbliżyć się do „oczka” bez uwzględnienia strategii krupiera.

6.9 Filmowa inaczej Hi-Low

Najbardziej widowiskowa strategia wbrew pozorom nie osiąga najlepszych wyników. Statystycznie wygrywa się co 3 rundę dzięki tej strategii, co w ogólnym rozrachunku wychodzi na minus. Ma jednak zdecydowaną przewagę dla gracza - jest najszybsza do zapamiętania i wdrożenia oraz pomaga ćwiczyć pamięć i podstawowe operacje matematyczne.

6.10 Intuicyjna

Ze względu na swój nieprzewidywalny charakter pokazuje, że pod względem matematycznym intuicyjne „Teraz pasuję” daje niemal najgorszy wynik ogólny. Gorsze wyniki daje tylko strategia Ekspansyjna. Warto jednak zauważyć pewną ciekawą zależność - mimo stosunkowo niewielkiej ilości wygranych, to w tej strategii

najczęściej pada blackjack. Nie tylko porównując, że gdy statycznie na 1600 zwycięstw pada nieco ponad 160 blackjacków (bez uwzględnienia strategii idealnej to około 150 na 1400), to przy tej strategi średnio na 4,6 zwycięstw pada 1 blackjack, co jest zadziwiającym wynikiem.

6.11 Reaguj na bank

Strategia ta ma jeden z wyższych wyników, a jednocześnie nie jest trudna do zapamiętania, co powoduje, że często pojawia się przy poruszaniu tematu strategii blackjackowych. Mimo dobrej proporcji liczby zwycięstw do porażek niestety przy tej strategii wychodzi się na minusie.

6.12 Podstawowa

Jest to jedna z nielicznych strategii, która pozwala na wychodzenie na plus jednak ma bardzo dużo skomplikowanych zależności i wymaga nauczenia się wielu warunków. Jako jedna z nielicznych wykorzystuje split w przypadku gry uczestnika.

6.13 Przełam passę

Jedna z niewielu strategii wychodząca na plus. Patrząc po wynikach pozostałych strategii można odnieść wrażenie, że ponieważ nie miała zbyt wielu porażek, to granica 15 punktów jest jedną z bardziej optymalnych by przestać ciągnąć kartę. Strategia wychodziła na plus też ze względu na dość dużą ilość blackjacków. (Po iluś porażkach często polegających na przekroczeniu 21 ciężej przekroczyć 21 i większa szansa na pokonanie krupiera. - kolokwialny ton)

6.14 Przetrzymaj passę

Mimo obniżania progu wynik jest ujemny, ale może być to kwestia mniejszej ilości dodatkowo punktowanych blackjacków. Przy jednej talii, gdzie średnia liczba gier to 9, może być ciężko uzyskać 5 porażek pod rząd stosując się do nie przekraczania ustalonego progu, który nie jest zbyt wysoki.

6.15 Zależna od szczęścia

Strategia ta jest niewiele poniżej zera jeżeli chodzi o wygrane pieniądze. Niestety wraz ze wzrostem liczby talii użytych w grze obniża się skuteczność tej strategii.

6.16 Pasująca

Można zauważyć, że kiedy na ręce jest 6 punktów, to ciągnięcie karty nie stwarza ryzyka przekroczenia wartości 21. Pomimo tego jest stosunkowo dużo zwycięstw na wszystkie rundy.

6.17 Przykładowe rozdanie

Poniżej znajduje się przykładowy, potasowany układ kart dla 1 talii:

Rysunek 6.1: Przykładowe rozdanie dla 1 talii

Można zauważyć, że jest to bardzo zróżnicowane rozdanie, co widać po wynikach strategii:

```
score(name='Ekspansyjna', winnings=3, draw=0, loosing=5, money=-20, blackjack=0)
score(name='ReagujNaBank', winnings=5, draw=0, loosing=3, money=20, blackjack=0)
score(name='HiLow', winnings=3, draw=1, loosing=5, money=-20, blackjack=0)
score(name='Intuicyjna', winnings=4, draw=0, loosing=6, money=-15.0, blackjack=1)
score(name='Krupierska', winnings=5, draw=0, loosing=3, money=20, blackjack=0)
score(name='NeverBust', winnings=1, draw=3, loosing=5, money=-40, blackjack=0)
score(name='Prawdopodobna', winnings=3, draw=3, loosing=3, money=0, blackjack=0)
score(name='PrzelamPasse', winnings=4, draw=1, loosing=4, money=0, blackjack=0)
score(name='PrzetrzymajPasse', winnings=5, draw=0, loosing=3, money=20, blackjack=0)
score(name='ZaleznaOdSzczescia', winnings=4, draw=0, loosing=5, money=-10, blackjack=0)
score(name='Podstawowa', winnings=2, draw=2, loosing=5, money=-30, blackjack=0)
score(name='Pasujaca', winnings=3, draw=0, loosing=8, money=-45.0, blackjack=1)
score(name='Idealna', winnings=7, draw=0, loosing=2, money=50, blackjack=0)
```

Rysunek 6.2: Zestawienie wyników

Dla tego układu aby osiągnąć najwyższy wynik należy najpierw spasować. Przegrywa się wówczas pierwszą rundę. Następnie należy ciągnąć kolejne 4 karty osiągając na ręce 21 punktów podczas, gdy krupier przekracza tę liczbę i w ten sposób wygrywa się tę rundę. Po rozpoczęciu trzeciej rundy należy wykonać polecenie „Hit” i sytuacja jest podobna do poprzedniej rundzie. Na ten moment są dwa zwycięstwa do jednej porażki bez remisu. Następną rundę pasując bez ciągnięcia należy przegrać. Kolejna runda po pociągnięciu karty to dokładnie 21 punktów i zwycięstwo nad krupierem. Następnie w kolejnych 4 rundach pasujemy bez ciągnięcia karty co daje nam 4 zwycięstwa (przekroczenie 21 przez krupiera za każdym razem). Po ostatnim ciągnięciu krupiera kończą się karty. Co dało nam końcowy wynik 50.

Większość strategii ukończyła grę na minusie. Dwóm strategiom udało się wyjść na zero, a tylko trzy (poza idealną) zarobiło na tej grze mając ten sam wynik 20. Tutaj tak jak w ogólnym podsumowaniu można zauważyć, że jedyna strategia, której udało się mieć blackjacka to idealna, ale i tak stosując tę strategię gracz jest stratny. Również zgodnie z założeniami uporczywe dążenie do 21 jest bardzo niebezpieczne. Warto za to nieco bardziej zwrócić uwagę na prawdopodobieństwo wyciągnięcia kart o danej wartości.

W przypadku blackjacka dobrze widać powiedzenie, że znajomość kart czasem zmienia decyzję gracza - nie dążymy do wygrania rundy, ale całej gry.

(może dołożyć cytat „Bądź gotów przegrać bitwę, żeby wygrać wojnę.” H. Jackson Brown, Jr.)

6.18 Blackjacki

Poniżej znajdują się porównania, która strategia ma największą szansą na blackjacka

6.19 Przewaga bez blackjacków

Poniżej znajduje się porównanie jak zmienia się ilość pieniędzy jeżeli nie bierzemy pod uwagę faktu, że blackjack jest punktowany dodatkowo.

6.20 Porażki poprzez przekroczeni 21

Na ile patrząc na ile rund w czasie talii średnio się tak przegrywa. Jak często wtedy by przegrał krupier?? -¿ Dlaczego często kasyno wychodzi na plus

Na koniec cytat Steinhousa ”Między duchem, a materią pośredniczy matematyka”

(trochę mnie niepokoi że rozdziały 6.4 - 6.16 to analiza kolejnych strategii, a na przykład 6.3 to zbiorcze porównanie wyników na wykresach - jakoś moc tych podrozdziałów jest nieadekwatna; być może wystarczyłby

rozdział 6.5 z podpunktami numerowanymi 6.5.1-6.5.13 (albo nienumerowanymi w ogóle) zbierający analizę strategii)

Podsumowanie

Bibliografia

[1] Angular technology. Web pages: https://cli.angular.io/.

[2] git technology. Web pages: https://git-scm.com/.

[3] Koło kolorów. Web pages: https://color.adobe.com/pl.

[4] nodejs enviromental. Web pages: https://nodejs.org/en/.

[5] npm enviromental. Web pages: https://www.npmjs.com/get-npm.

[6] Python documentation. Web pages: https://docs.python.org/3/library/.

[7] Python enviromental. Web pages: https://www.python.org/downloads/.

[8] Python flask library. Web pages: https://pypi.org/project/Flask/.

[9] Python jsoncheme library. Web pages: https://pypi.org/project/jsonschema/.

[10] Python namedlist library. Web pages: https://pypi.org/project/namedlist/.

[11] Typescript technology. Web pages: https://www.typescriptlang.org/.

[12] Typescript technology. Web pages: https://rxjs-dev.firebaseapp.com/.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, S. Clifford. *Wprowadzenie do algorytmów*. Wydawnictwo Naukowe PWN, year=2017.

[14] C. Drösser. *Matematyka daj się uwieść*. Wydawnictwo Naukowe PWN, year=2011.

Zawartość płyty CD

W tym rozdziale należy krótko omówić zawartość dołączonej płyty CD.