UNIWERSYTET WARSZAWSKI

Wydział Matematyki,
Informatyki i Mechaniki

Jakub Pawlewicz

TECHNIKI ALGORYTMICZNE ROZWIĄZYWANIA GIER NA PRZYKŁADZIE GRY W KOŚCI YAHTZEE

Rozprawa Doktorska

Promotor rozprawy
prof. dr hab. Krzysztof Diks
Instytut Informatyki

Sierpień 2008

Oświadczenie promotora rozprawy

Niniejsza rozprawa jest gotowa do oceny przez recenzentów.

Data Podpis promotora rozprawy

Oświadczenie autora rozprawy

Oświadczam, że niniejsza rozprawa została napisana przeze mnie samodzielnie.

Data Podpis autora rozprawy

Streszczenie

Yahtzee jest komercyjną, najpopularniejszą grą w kości na świecie. Yahtzee może być rozgrywana przez jedną, jak i wiele osób. W wersji jednoosobowej zbudowano komputerowe strategie optymalne dla maksymalnej wartości oczekiwanej wyniku oraz dla maksymalnego prawdopodobieństwa uzyskania określonego wyniku. W wersji wieloosobowej te strategie są dalekie od optymalności. W tym przypadku liczba stanów jest zbyt duża, aby było możliwe zbudowanie strategii optymalnej z pomocą dotychczasowych metod i technik.

W tej pracy po raz pierwszy dokonano dokładnej analizy wersji wieloosobowej gry Yahtzee. Zaproponowano implementacje strategii optymalnych dla gry jednoosobowej w sposób pozwalający na ich obliczanie zdecydowanie szybciej niż to było dotychczas. Precyzyjnie wyliczono rozmiar zasobów potrzebnych do stablicowania strategii optymalnej dla gry dwuosobowej oraz wykazano, że stablicowanie strategii optymalnej dla gry z większą liczbą osób jest poza zasięgiem współczesnych komputerów. Zaproponowano strategię heurystyczną dla gry wieloosobowej i za pomocą specjalnie skonstruowanych w tym celu eksperymentów wykazano, że w praktyce jest to strategia nieodróżnialna od strategii optymalnej.

Dokonano także eksperymentalnej analizy przewagi strategii optymalnej w stosunku do strategii stosowanych przez nieoptymalnych przeciwników jakimi są ludzie. Wyniki eksperymentów dowodzą, że gra Yahtzee jest grą „bardzo" losową i przewaga strategii optymalnej jest niewielka.

Słowa kluczowe: Yahtzee, rozwiązywanie gier, programowanie dynamiczne, przeszukiwanie drzewa gry, metody heurystyczne, strategia optymalna.

Klasyfikacja tematyczna ACM: I.2.8.

Abstract

Yahtzee is the most popular commercial dice game in the world. Yahtzee can be played either by one player or many players. The single player version was provided with optimal computer strategies for maximizing the expected average score and for maximizing the probability of gaining a particular score. However, when it comes to the multi-player version, these strategies are far from optimal. In this case the number of states is too high to develop an optimal strategy with the use of formerly known methods and techniques.

This work presents the first in-depth analysis of the multi-player version of Yahtzee. The proposed implementation of strategies optimal for the single player version allows for a much faster calculation than before. The resources necessary to memorize the optimal strategy for a two-player game have been precisely estimated. It has been shown that developing an optimal strategy for more players is not possible with the use of current technology. A heuristic strategy was suggested in case of the multi-player game and by means of experiments created especially for this purpose it has been proven that in practice this strategy is not distinguishable from the optimal strategy.

An experimental analysis of the actual advantage of the optimal strategy over non-optimal opponents like humans has also been conducted. The results show that Yahtzee is "highly" a game of chance and the advantage of the optimal strategy is not significant.

Keywords: Yahtzee, game solving, dynamic programming, game tree search, heuristic methods, perfect play.

ACM Classification: I.2.8.

Spis treści

Rozdział 1

Wprowadzenie

Od wieków ludzkość pasjonuje się grami intelektualnymi. Ten rodzaj rozrywki jest popularną formą spędzania czasu i gimnastykowania umysłu. Wiele gier jest bardzo popularnych w licznych krajach, a niektóre stają się nawet sportem narodowym, jak na przykład Go w krajach wschodnich, w szczególności w Korei. Celem gry jest wygrać z przeciwnikiem i powstaje pytanie jak to zrobić? Jak grać optymalnie? Jakie jest najlepsze posunięcie w danej sytuacji?

Tylko nieliczne gry umiemy rozwiązać przez pokazanie optymalnej strategii z użyciem metod matematycznych. Taką grą jest Nim. Zainteresowanych odsyłamy do najbogatszego źródła technik matematycznych [BCG04]. W ogólności, dla większości gier nie istnieją takie proste strategie optymalne. Dla danej gry, rozstrzygnięcie, która strategia jest najlepsza w danej sytuacji, zazwyczaj dokonywane było przez analizę gier ekspertów – najlepszych graczy świata z długoletnim doświadczeniem. Pojawienie się komputerów dało nowe możliwości analizowania gier.

Od początku istnienia komputerów rozważano stworzenie maszyny wraz programem komputerowym, która by potrafiła wygrywać w Szachy z ludźmi, a nawet pokonać mistrza świata. Na początku były to rozważania czysto filozoficzne, ale z czasem zaczęto konstruować coraz to skuteczniejsze algorytmy heurystyczne bazujące na „siłowym" przeszukiwaniu drzewa gry i stopniowo, wraz z rozwojem technologicznym, cel osiągnięto. W 1997 roku superkomputer skonstruowany w laboratorium IBM wygrał z ówczesnym mistrzem świata Gari Kasparowem [SP97]. Dzisiaj najsilniejsze programy komputerowe uruchamiane na zwykłych komputerach osobistych wygrywają z najlepszymi graczami. W 2006 program Deep Fritz wygrał 4–2 z mistrzem świata Władimirem Kramnikiem [Che06]. Dodajmy, że Deep Fritz wcale nie jest najsilniejszym programem. Aktualnie za najlepszy program uznawany jest program szachowy Rybka [RK08].

Przykład Szachów pokazuje, że postęp w informatyce, szczególnie w algorytmice, plus postęp technologiczny pozwalają tworzyć graczy komputerowych znacznie silniejszych niż najlepsi ludzie. W niektórych grach stworzenie programu komputerowego o sile poza zasięgiem ludzi jest znacznie prostsze. Na przykład w popularnej grze planszowej Reversi, Michael Buro stworzył w miarę prosty program o nazwie Logistello [Bur97], który bez trudu pokonał aktualnego mistrza świata. Możliwość stworzenia bardzo silnego gracza, bądź nawet wyroczni dla danej gry za pomocą programów komputerowych, jest wystarczająco silnym bodźcem, aby naukowcy poświęcali temu odpowiednio dużo uwagi.

Najbardziej spektakularnym odkryciem w ostatnich latach jest wyliczenie strategii optymalnej dla gry Warcaby [SBB+07]. Okazuje się, że przy optymalnej grze obu graczy wynikiem gry jest remis. Zespół, który zrealizował powyższe przedsięwzięcie, stosował znane techniki algorytmiczne, nie wprowadzając żadnych nowych metod. Projekt przede wszystkim polegał na wykorzystaniu mocy obliczeniowej dużej liczby komputerów w celu przeanalizowanie olbrzymiej liczby sytuacji w grze (liczba ta wynosi w przybliżeniu 10^{14}). Wynik ten ma duże znaczenie z teoretycznego punktu widzenia. W praktyce, program, który znacznie wykraczał poza możliwości ludzi istniał już od kilkunastu lat. Jest to również dzieło tego samego zespołu, a program nazywa się Chinook [BBB+08].

W rozwiązywaniu gier nie zawsze najważniejszą rolę pełnią zasoby sprzętowe. W 1993 roku Victor Allis rozwiązał grę Go–Moku (u nas popularnie znaną pod nazwą Kółko i Krzyżyk na Pięć) wprowadzając szereg zupełnie nowych technik przeszukiwania [AvdHH96]. Wskazano strategię wygrywającą dla gracza rozpoczynającego grę. W kolejnych latach jego techniki były doskonalone, a jedna z nich była również użyta przy rozwiązywaniu gry Warcaby.

Zarówno w przypadku gry Warcaby, jak i Go–Moku, podane rozwiązania są tak zwanymi *rozwiązaniami słabymi*, to znaczy takimi rozwiązaniami, w których wskazuje się strategię wygrywającą od stanu początkowego. Nie daje to jednak informacji, które posunięcie jest najlepsze w dowolnym stanie mogącym pojawić się w innej rozgrywce niż optymalna. Dla ludzi dużo bardziej pożądanymi są tak zwane *silne rozwiązania*. Są to takie rozwiązania, które dla każdej możliwej sytuacji potrafią wskazać najlepsze posunięcie oraz wartość gry – przegrana, remis, czy wygrana. Przykładem gry, która niedawno została silnie rozwiązana przez dwójkę holenderskich naukowców, jest Awari [RV02]. Liczba stanów w tej grze wynosi w przybliżeniu 10^{12}. Posiadanie takiej silnej wyroczni jest marzeniem każdego gracza. Natychmiastowa analiza z optymalną strategią jest pożądanym narzędziem w każdej grze.

Wszystkie dotychczasowe przykłady dotyczyły gier dwuosobowych, deterministycznych i z pełną informacją. Analiza staje się trudniejsza, jeśli gra zawiera elementy losowości, a szczególnie trudna, jeżeli gra jest dodatkowo z niepełną informacją. W grach losowych już nie wystarcza stwierdzenie, czy dana sytuacja jest wygrana, bądź przegrana. Tutaj wymaga się większej informacji, jak na przykład maksymalnej wartości oczekiwanej wyniku. Należy wspomnieć, że dla tego typu gier został dokonany olbrzymi postęp w ostatniej dekadzie. Mamy tu na myśli gry Brydż i Poker. Jeszcze pod koniec lat dziewięćdziesiątych nie istniały programy grające w Brydża, chociażby na poziomie amatorów. Sytuację tą drastycznie zmienił Matthew L. Ginsberg [Gin99], który stworzył program o nazwie GIB, w którym pojawiło się sporo całkiem nowych pomysłów. Dzisiejsze programy eksploatujące wprowadzone wtedy techniki grają na poziomie ekspertów. W latach 2005–2006, równorzędne pojedynki z czołówką holenderskich graczy [Hee06] stoczył program o nazwie Jack [KK06], a do analizy trudnych rozdań regularnie stosuje się najlepsze programy jako wyrocznię. W Pokerze kolejną pojawiającą się trudnością jest duża losowość i konieczność modelowania przeciwnika. Do konstrukcji programów, które nawiązują równorzędna walkę z ludźmi [Var08], stosuje się wiedzę z kilku dziedzin: ekonomii (równowaga Nasha), sieci neuronowych, rachunku prawdopodobieństwa [Joh07].

W tej pracy zrobimy kolejny krok w skróceniu listy gier dotychczas nierozwiązanych. Zmienimy status wiedzy o losowej grze z pełną informacją Yahtzee – pokażemy „prawie" silne rozwiązanie tej gry w wersji wieloosobowej stosując

szereg technik algorytmicznych oraz wprowadzając nowe metody heurystyczne. Gra *Yahtzee* jest międzynarodową, komercyjną grą w kości. Jest to najpopularniejsza i najlepiej sprzedająca się odmiana gry w kości. W Polsce spotykane są wersje pod nazwą *Poker* i *Generał*. Na świecie rocznie sprzedaje się 50 milionów zestawów do gry w Yahtzee. Szacuje się, że regularnie gra w nią ponad 100 milionów ludzi[1].

W grze może uczestniczyć dowolna liczba osób. W wersji jednoosobowej chodzi tylko o zdobycie możliwie największej liczby punktów, natomiast gdy w grze uczestniczą co najmniej dwie osoby, celem jest uzyskanie większej liczby punktów od przeciwników. Stosunkowo niewielka liczba stanów, duża popularność gry oraz coraz szybsze komputery zaowocowały pojawieniem się szeregu prac analizujących różne strategie w grze jednoosobowej. Równolegle w kilku miejscach przedstawiono strategię optymalną ([Ver99, Woo03, Gle06]). Jednakże zaproponowane techniki nie pozwalają na realizację strategii optymalnych, gdy w grze uczestniczy więcej niż jedna osoba. Oszacowania rozmiaru przestrzeni stanów gry jasno wskazują, że ze względu na zasoby sprzętowe nie będzie to możliwe w najbliższym czasie w wersji dla dwóch osób, a dla większej liczby osób jest to praktycznie niemożliwe. W rezultacie, dla wersji wieloosobowej do dzisiaj nie pojawiły się żadne strategie, które byłyby możliwie bliskie strategii optymalnej.

Główny wynik tej pracy to propozycja strategii bardzo bliskiej strategii optymalnej w grze wieloosobowej. W tym celu przeprowadzona zostanie dokładna analiza gry Yahtzee, a do budowy tej strategii zostanie wprowadzona całkowicie nowa, uniwersalna technika, która z pewnością może być stosowana do innych gier losowych z pełną informacją.

Na początku zaznajomimy się z grą Yahtzee. W rozdziale 2 zostaną przedstawione zasady gry oraz przykłady rozgrywek wraz z prostą analizą sytuacji.

W rozdziale 3 zajmiemy się szczegółowo wersją jednoosobową. Przeanalizujemy dokładnie strukturę gry. W porównaniu do dotychczasowych prac analizujących grę Yahtzee, dokonane zostaną nowe obserwacje pozwalające na znaczną redukcję rozmiaru grafu reprezentującego przestrzeń stanów. Zaproponujemy metodę obliczania strategii optymalnej dla maksymalizacji wartości oczekiwanej w czasie o rząd wielkości mniejszym, niż to było zrobione przez poprzedników [Ver99, Woo03, Gle06]. Następnie zajmiemy się strategią maksymalizującą prawdopodobieństwo osiągnięcia określonego wyniku. Ta strategia jest już trudniejsza do uzyskania. Sposób na jej wyliczanie został przedstawiony w [Cre02]. Jednakże w praktyce ta metoda wymaga zbyt dużo czasu do generowania odpowiedzi w dowolnym stanie gry. Tutaj pokażemy zupełnie nowe podejście. Wprowadzimy pojęcie dystrybucji i za ich pomocą stablicujemy strategię na maksymalne prawdopodobieństwo osiągnięcia określonego wyniku. Przedstawiona metoda pozwala na natychmiastowe podejmowanie decyzji, czyniąc ją przez to bardzo praktyczną.

W dotychczasowych pracach pojawiały się jedynie wzmianki na temat gry wieloosobowej. Przedstawiono bardzo proste strategie bazujące na rozwiązaniach dla wersji jednoosobowej (np. w [Ver99]). Nie została przeprowadzona żadna dokładna analiza przestrzeni stanów. W rozdziale 4 zajmiemy się wersją wieloosobową. Opiszemy czym jest strategia optymalna. Sprecyzujemy pojęcie celu rozgrywki wprowadzając termin punktów meczowych. Po raz pierwszy zostanie pokazana szczegółowa analiza niezbędnych zasobów potrzebnych do po-

[1] Źródło: [Ide04]

liczenia strategii optymalnej dla gry dwuosobowej, z uwzględnieniem wszelkich możliwych redukcji liczby stanów. Po wszystkich tych zabiegach okazuje się, że wygenerowanie strategii optymalnej jest w zasięgu dzisiejszych technologii.

Następnie zaprezentujemy najważniejszy wynik tej pracy — strategię heurystyczną dla gier wieloosobowych, która jest „prawie" optymalna. Heurystyka wykorzystuje wprowadzone wcześniej pojęcie dystrybucji z gry jednoosobowej i daje zaskakująco dobre wyniki. Pokazanie skuteczności tej strategii jest jednak zadaniem nietrywialnym. Specjalnie skonstruowane eksperymenty pozwalają na miarodajne stwierdzenie, że błąd w stosunku do strategii optymalnej okazuje się być na tyle mały, że jest praktycznie zaniedbywalny. Technika, która została użyta do konstrukcji strategii przybliżającej strategię optymalną z bardzo małym błędem, jest niezależna od gry i z pewnością może być stosowana do innych gier losowych z pełną informacją.

Yahtzee jest grą bardzo losową w tym sensie, że mimo możliwości podejmowania decyzji i tak ostateczny wynik w dużej mierze zależy od szczęścia. W związku z tym przewaga strategii „prawie" optymalnej w pojedynkach z ludźmi, którzy grają nieoptymalnie, jest ledwo zauważalna. Aby praktycznie stwierdzić ile tak na prawdę daje użycie strategii optymalnej (zakładając, że jesteśmy w stanie wygenerować taką strategię), została przeprowadzona pracochłonna analiza blisko 25 milionów gier ludzi na różnych poziomach zaawansowania.

Rozdział 2

Zasady gry Yahtzee

2.1 Zasady ogólne

W grze bierze udział dowolna liczba graczy (od jednego wzwyż). W praktyce nie przekracza ona czterech. Do gry potrzebne jest pięć sześciościennych kości oraz tabela z listą 13 kategorii. Każdy z graczy rzuca kośćmi tak, aby uzyskiwać różne kombinacje i stara się wypełnić tabelę możliwie największymi zapisami punktowymi.

Gra składa się z tylu rund ile jest kategorii (czyli 13 rund). Na początku gry wszystkie kategorie są puste. W danej rundzie każdy gracz może wykonać trzy rzuty. W drugim i trzecim rzucie można część kości zostawić i rzucać pozostałymi. Najdalej po trzecim rzucie gracz musi wybrać jedną kategorię i dokonać zapisu według reguł opisanych w punkcie 2.2. Wybrana kategoria powinna być pusta. W czasie całej rozgrywki każda kategoria będzie zapisana dokładnie raz.

Po ostatniej rundzie gra zostaje zakończona i ostateczna punktacja zostaje wyliczona na podstawie reguł opisanych w punkcie 2.3. Wygrywa gracz, który zdobędzie najwięcej punktów.

2.2 Kategorie

Kategorie dzielimy na dwie części: *sekcja górna* (tzw. „szkółka") oraz *sekcja dolna*. Każda kategoria odpowiada pewnemu układowi oczek na kościach. Jeżeli oczka na kościach spełniają wymogi danego układu, to zapisujemy liczbę punktów zgodnie z tabelą 2.1. Przy opisach układów przez *jednakowe kości* mamy na myśli kości z tą samą liczbą oczek. Jeżeli układ nie jest spełniony, to zapisywane jest zero.

2.2.1 Joker

W przypadku, gdy wyrzucone zostanie Yahtzee (pięć jednakowych kości), to możemy użyć tego układu jako *Jokera* zapisując go w:

- odpowiadającej temu układowi kategorii w sekcji górnej lub
- w dowolnej innej kategorii pod warunkiem, że odpowiadająca danemu układowi kategoria w sekcji górnej jest już zajęta.

Sekcja górna		
Kategoria	**Układ**	**Punkty**
Jedynki	dowolny	$1 \times$ liczba jedynek
Dwójki	dowolny	$2 \times$ liczba dwójek
Trójki	dowolny	$3 \times$ liczba trójek
Czwórki	dowolny	$4 \times$ liczba czwórek
Piątki	dowolny	$5 \times$ liczba piątek
Szóstki	dowolny	$6 \times$ liczba szóstek
Sekcja dolna		
Kategoria	**Układ**	**Punkty**
Trójka	$\geqslant 3$ jednakowe kości	suma oczek wszystkich kości
Kareta	$\geqslant 4$ jednakowe kości	suma oczek wszystkich kości
Ful	trójka + para $(*)$	25
Mały strit	cztery kolejne wartości kości	30
Duży strit	pięć kolejnych wartości kości	40
Yahtzee	5 jednakowych kości	50
Szansa	dowolny	suma oczek wszystkich kości

$(*)$ Liczby oczek na kościach z trójki i z pary muszą być różne.

Tabela 2.1: Zestawienie kategorii.

Do zapisania punktacji w sekcji dolnej stosuje się reguły wybranej kategorii, tak jakby dany układ był spełniony. Natomiast zasady punktacji w sekcji górnej pozostają niezmienione.

2.3 Punktacja

Ostateczna liczba punktów równa się sumie punktów za wszystkie kategorie plus ewentualne premie:

- za szkółkę,
- za Jokery.

2.3.1 Premia za szkółkę

W przypadku, gdy sumaryczna liczba punktów w sekcji górnej wyniesie co najmniej 63 punkty dodatkowo dostaje się 35 punktów. Zauważmy, że $63 = 3 \cdot (1 + 2 + 3 + 4 + 5 + 6)$. Oznacza to, że wystarczy dla każdej możliwej liczby oczek zebrać po trzy jednakowe kości. Niemniej, sposób, w jaki zostanie osiągnięty premiowany próg w sekcji górnej, jest dowolny.

2.3.2 Premia za Jokery

Za każdego użytego Jokera przysługuje 100 punktów, pod warunkiem, że przed zapisaniem Jokera w kategorii Yahtzee widnieje zapis 50. Premia za użycie Jokera nie jest przyznawana, jeżeli kategoria Yahtzee jest wolna lub zostało w niej zapisane zero. W szczególności premia nie przysługuje za samo zapisanie 50 do kategorii Yahtzee.

Premie za zapisane Jokery w sekcji górnej nie liczą się przy zbieraniu punktów na premię za szkółkę. Na przykład przy zapisie Jokera w Szóstkach dostaniemy w sumie 130 punktów, natomiast przy zliczaniu punktów na premię uwzględnimy tylko 30 punktów (czyli taką liczbę punktów, jaką byśmy zapisali bez premii za Jokera). W praktyce, aby uniknąć niejednoznaczności, do zapisywania premii za Jokery używa się specjalnej rubryki, w której zapisuje się liczbę takich premii, a dopiero na końcu przy podliczaniu punktacji dodaje się 100 razy liczba uzyskanych premii za Jokery.

2.4 Przykłady

Przykłady różnych zapisów. Najtrudniejszą częścią zasad jest przydzielenie punktów za zapisanie układu do wybranej kategorii. W tabeli 2.2 umieściliśmy szereg przykładów różnych nietypowych zapisów. Oprócz punktacji zaznaczone jest użycie Jokera.

Rzut	Kategoria	Punkty
⚀⚀⚁⚂⚄	Jedynki	0
⚀⚀⚁⚂⚄	Dwójki	4
⚀⚀⚁⚂⚄	Trójki	3
⚀⚀⚁⚂⚄	Trójka	0
⚀⚀⚁⚂⚄	Mały strit	30
⚀⚀⚁⚂⚄	Szansa	16
⚁⚁⚁⚂⚂	Trójki	9
⚁⚁⚁⚂⚂	Trójka	17
⚁⚁⚁⚂⚂	Kareta	0
⚁⚁⚁⚂⚂	Ful	25
⚁⚁⚁⚂⚂	Duży strit	0
⚂⚂⚂⚂⚂	Piątki	0 (Joker, jeżeli Szóstki są zapisane)
⚂⚂⚂⚂⚂	Szóstki	30 (Joker)
⚂⚂⚂⚂⚂	Kareta	30 (Joker, jeżeli Szóstki są zapisane)
⚂⚂⚂⚂⚂	Ful	$\begin{cases} 25 \text{ (Joker)} & \text{jeżeli Szóstki są zapisane} \\ 0 & \text{jeżeli Szóstki są wolne} \end{cases}$
⚂⚂⚂⚂⚂	Yahtzee	50

Tabela 2.2: Zapis przykładowych układów.

Przykładowa rozgrywka. Na koniec prześledźmy przykładową rozgrywkę jednego gracza. W tabeli 2.3 przedstawione jest wszystkie 13 rund. W kolumnie **Rzut** znajdują się układy kości po pierwszym, drugim, bądź trzecim rzucie w każdej z rund. Kolumna **Wybór** opisuje decyzję podjętą przez gracza po danym rzucie. Są to albo kości zatrzymane przed następnym rzutem, albo nazwa kategorii, w której gracz zdecydował się wykonać zapis. Szczegółowy komentarz do danej sytuacji umieszczony jest w kolumnie **Komentarz**.

Tabela 2.3: Przykładowa rozgrywka.

Rzut	**Wybór**	**Komentarz**
⚀⚀⚄⚃⚃	⚃⚃	Po pierwszym rzucie ⚀⚀⚄⚃⚃, najlepiej jest zbierać do Trójek i zatrzymujemy kości ⚃⚃.
⚄⚄⚂⚃⚃	⚄⚄	Rzucamy trzema wybranymi kośćmi dostając ⚄⚄⚂. Teraz bardziej opłaca zbierać się do Piątek, więc zatrzymujemy ⚄⚄.
⚄⚄⚂⚀⚄	Trójka	Po trzecim rzucie musimy wybrać kategorię do zapisu. W tym przypadku jedyne sensowne to Piątki i Trójka. W tak wczesnej fazie zapisanie 25 punktów w Trójce jest nieco bardziej opłacalne, gdyż na początku poluje się raczej na co najmniej cztery Piątki, bądź cztery Szóstki, żeby zapisać je w sekcji górnej. W ten sposób zwiększa się szanse na otrzymanie premii za szkółkę.
⚄⚀⚃⚂⚄	⚄⚄	Kolejną rundę rozpoczynamy od zbierania do Czwórek.
⚄⚂⚅⚃⚄	⚂⚅⚃⚄	Na trzech wybranych kościach po rzucie pojawia się ⚂⚃⚅. Teraz opłaca się rzucać do Dużego Strita, gdyż Mały Strit i tak mamy już zapewniony.
⚀⚂⚅⚃⚄	Mały Strit	Nie udało się rzucić ⚁, więc zapisujemy 30 w Małym Stricie.
⚅⚃⚀⚄⚁	Duży Strit	Już w pierwszym rzucie uzyskujemy Dużego Strita, tak więc nie ma potrzeby wykonywać dalszych rzutów i od razu zapisujemy 40 w Dużym Stricie.
⚄⚄⚀⚂⚅	⚄⚄	Zbieramy do Czwórek.
⚄⚄⚁⚄⚅	⚄⚄⚄	Zatrzymujemy kolejną ⚄ i wykonujemy ostatni rzut.
⚄⚄⚄⚄⚄	Yahtzee	Przy takim rzucie jedyna słuszna decyzja to zapisać 50 w Yahtzee.
⚂⚂⚂⚃⚅	⚂⚂⚂	Zatrzymujemy wszystkie ⚂.
⚂⚂⚂⚃⚁	⚂⚂⚂	Nie doszła żadna ⚂, więc próbujemy jeszcze raz dorzucić jakąś ⚂ z pozostałych dwóch kości.
⚂⚂⚂⚂⚁	Szóstki	Musimy zapis układ. W grę wchodzą tylko Szóstki i Kareta. Bardziej opłaca się zapisać 24 w Szóstkach, gdyż daje to nam sporą zaliczkę na premię za szkółkę.

Tabela 2.3: Przykładowa rozgrywka (ciąg dalszy).

Rzut	Wybór	Komentarz
⚄⚄⚄⚀⚁	⚄⚄	Zbieramy do Piątek.
⚀⚄⚄⚁⚃	⚄⚄	Jeszcze raz próbujemy wyrzucić jakąś ⚄.
⚄⚄⚄⚄⚄	Piątki	Rzuciliśmy kolejne Yahtzee, jednakże ta kategoria jest już zajęta. Możemy użyć Jokera, wtedy dostaniemy bonus 100 punktów. Zasady dotyczące Jokera mówią, że jeśli odpowiednia kategoria górna jest wolna, to Joker może być przyznany tylko przez zapisanie go w tej kategorii. Tutaj mamy taką sytuację. Zatem zapisujemy w Piątkach 25 punktów i dopisujemy sobie bonus 100 punktów. Zwróćmy uwagę, że zapis w innych kategoriach nie będzie traktowany jako Joker. Na przykład nie możemy zapisać Fula, tzn. jeśli zapisalibyśmy Fula, to musielibyśmy wpisać w tej kategorii 0 punktów.
⚅⚅⚅⚂⚁	⚅⚅⚅	Mimo, że Szóstki są już zapisane, opłaca zatrzymać się wszystkie ⚅, gdyż mamy szansę na wysoki wynik do Karety, bądź nawet szansę wyrzucenia kolejnego Yahtzee. Awaryjnie otrzymamy też wysoki wynik do zapisania w Szansie.
⚅⚅⚅⚅⚀	⚅⚅⚅⚅	Mamy już Karetę. Próbujemy jeszcze rzucić Yahtzee.
⚅⚅⚅⚅⚅	Ful	Znowu udało się rzucić Yahtzee. Możemy wykorzystać ten układ jako Joker. Tym razem odpowiednia kategoria w sekcji górnej (czyli Szóstki) jest już zapisana, zatem możemy użyć Jokera w dowolnej innej kategorii i dostać kolejne 100 punktów premii. Najbardziej opłaca się zapisać Karetę, ale tym razem zapiszemy Fula. Normalnie pięć jednakowych kości nie spełnia układu Fula, ale zasady Jokera mówią, że zapisujemy wtedy tyle punktów, jak gdyby układ był spełniony, więc zapisujmy 25 punktów i zapamiętujemy kolejne 100 punktów premii.
⚂⚂⚀⚀⚃	⚂⚂	Zbieramy do Trójek.
⚂⚂⚂⚅⚀	⚂⚂⚂	Zatrzymujemy kolejną ⚂.
⚂⚂⚂⚂⚀	Trójki	Doszła jeszcze jedna ⚂ i zapisujemy 12 w Trójkach. W sumie na premię za szkółkę mamy już 61 punktów (12 z Trójek, 25 z Piątek i 24 z Szóstek), więc brakuję nam już tylko 2 punktów.

Tabela 2.3: Przykładowa rozgrywka (ciąg dalszy).

Rzut	**Wybór**	**Komentarz**
⚅⚄⚃⚂⚀	⚅	Premię za szkółkę mamy już prawie zapewnioną, więc będziemy się starać zrobić Karetę za jak największą liczbę punktów, ewentualnie zapisując coś w Szansie, dlatego zostawiamy kość z największą liczbą oczek.
⚅⚄⚁⚃⚅	⚅⚅	Kontynuujemy nasz plan. W najgorszym razie będziemy zapisywać jakąś kategorię z sekcji górnej.
⚅⚂⚄⚀⚅	Jedynki	Suma kości nie jest zbyt wysoka, więc nie opłaca się zapisywać Szansy i lepiej zapisać 1 w Jedynkach.
⚃⚂⚃⚅⚃	⚃⚃	Zbieramy do Czwórek, a ewentualnie cały czas myślimy o Karecie.
⚃⚁⚃⚁⚃	⚃⚃	Niestety nie doszła żadna ⚃.
⚃⚀⚂⚂⚃	Czwórki	Jedyny sensowny wybór. W tym momencie mamy już w sumie 70 punktów w sekcji górnej, a więc dostaniemy 35 punktów za premię za szkółkę.
⚄⚄⚀⚄⚁	⚄⚄	Wciąż będziemy próbować zrobić Karetę.
⚄⚁⚄⚄⚁	⚄⚄⚄	Brakuje jeszcze jednej ⚄.
⚄⚁⚄⚄⚁	Dwójki	Po trzecim rzucie układ się nie zmienił. Punktów na Szansę jest mało, więc zapisujemy 4 w Dwójkach.
⚅⚄⚀⚅⚂	⚅⚅	Dwie ⚅ dobrze rokują, jak nie na Karetę, to na pokaźny wynik w Szansie.
⚅⚅⚃⚅⚂	⚅⚅⚅	Blisko Karety.
⚅⚅⚃⚅⚂	Szansa	Układ się nie zmienił i zapisujemy 25 w Szansie.
⚃⚂⚃⚂⚂	⚂⚂⚂	Jedyna wolna kategoria to Kareta i staramy się ją uzyskać.
⚀⚂⚅⚂⚂	⚂⚂⚂	Jeszcze jeden rzut.
⚄⚂⚄⚂⚂	Kareta	Nie udało się uzyskać czterech jednakowych kości. Musimy zapisać 0 w Karecie.

Ostateczna punktacja i stan zapisów w każdej kategorii po tej rozgrywce jest pokazany w tabeli 2.4.

Jedynki	1	
Dwójki	4	
Trójki	12	
Czwórki	8	
Piątki	25	(plus 100 za Jokera)
Szóstki	24	
Szkółka (suma punktów sekcji górnej)	74	(plus 35)
Trójka	25	
Kareta	0	
Ful	25	(plus 100 za Jokera)
Mały Strit	30	
Duży Strit	40	
Yahtzee	50	
Szansa	25	
Suma (razem z premiami)	504	

Tabela 2.4: Stan punktacji po zakończeniu przykładowej rozgrywki.

Rozdział 3

Gra jednoosobowa

Yahtzee może być grą jednoosobową. Taka wersja to pasjans, a celem jest zdobycie jak największej liczby punktów. Wyróżniamy dwie istotne strategie:

1. maksymalizacja wartości oczekiwanej wyniku,
2. maksymalizacja prawdopodobieństwa osiągnięcia określonego wyniku.

Pierwsza strategia na wartość oczekiwaną jest naturalnym podejściem, które stosuje się przy konstruowaniu gracza komputerowego. Popularność gry oraz stosunkowo mała liczba jej stanów przyciągnęła uwagę środowiska informatycznego. Niezależnie powstało wiele implementacji tej strategii. Najważniejsze z nich to te, których autorami są:

- Tom Verhoeff [Ver99],
- Phil Woodward [Woo03],
- James Glenn [Gle06].

Chronologicznie jako pierwsza pojawiła się implementacja Toma Verhoeffa. Pełne źródła w Turbo Pascalu dostępne są na jego stronie [Ver99]. W tej implementacji redukcja grafu gry umożliwia wydajne stablicowanie jej stanów i pozwala na wyliczenie strategii optymalnej w ciągu kilku minut.

Najczęściej cytowanym artykułem dotyczącym jednoosobowej gry Yahtzee jest praca Phila Woodwarda z „Chance" [Woo03]. Użyte algorytmy nie były zbyt efektywne, a wyliczenia trwały kilka dni. Najważniejszym przesłaniem tej pracy jednak było pokazanie wyższości komputera nad ludźmi. Nawet w tak losowej grze jak Yahtzee pojawiają się sytuacje nieoczywiste dla ludzi, a bezbłędny komputer jest w stanie wskazać najlepsze wybory.

Najwięcej badań gry Yahtzee dotychczas przeprowadził James Glenn [Gle06, Gle07]. Oprócz implementacji strategii optymalnej na maksymalizowanie wartości oczekiwanej, badał on siłę różnych nieoptymalnych strategii heurystycznych. W porównaniu do Verhoeffa zaobserwował on kolejne możliwości redukcji przestrzeni stanów gry. Niedawno w pracy [Gle07] podał on dalsze usprawnienia, które są podobne do zaproponowanych w tym rozdziale. Jednak Glenn nie wykorzystał do końca dokonanych obserwacji i zostawił sporo miejsca na kolejne optymalizacje.

W punkcie 3.1 przedstawimy kompletną analizę przestrzeni stanów gry i zależności między nimi. Przedstawiona w punkcie 3.2 implementacja strategii optymalnej na wartość oczekiwaną jest kilkukrotnie szybsza niż najlepsze dotychczas znane i pozwala na stablicowanie tej strategii w ułamku minuty. Tak dużą efektywność uzyskujemy przez dokonanie szeregu obserwacji, zastosowanie technik algorytmicznych i optymalizacji. Wszystkie te elementy zostaną tutaj szczegółowo omówione.

Druga strategia, maksymalizująca prawdopodobieństwo osiągnięcia pewnego ustalonego wyniku, jest już trudniejsza do realizacji i spotkała się ze znacznie mniejszym zainteresowaniem. Jedyne rozwiązanie jakie dotychczas się pojawiło, zostało zrealizowane przez Cremersa w jego pracy magisterskiej [Cre02] napisanej pod opieką Toma Verhoeffa. Przedstawiona tam technika jest jednak niepraktyczna. W czasie gry, na wyliczenie wartości potrzebnych do podjęcia decyzji potrzeba zdecydowanie zbyt wiele czasu.

W punkcie 3.3 przedstawimy zupełnie nowe podejście do stablicowania strategii na maksymalne prawdopodobieństwo osiągnięcia określonego wyniku. Zostanie wprowadzone pojęcie dystrybucji. Użycie dystrybucji pozwoli na natychmiastowe otrzymywanie najlepszych posunięć w dowolnej sytuacji, co nie było możliwe w [Cre02]. W dalszej części pracy dystrybucje okażą się użyteczne w wieloosobowej odmianie gry i pozwolą na konstruowanie prawie optymalnych strategii, co zostanie zaprezentowane w rozdziale 4.

3.1 Analiza przestrzeni stanów

Rozgrywkę opisujemy jako zbiór stanów i przejść między stanami. Stan określa sytuację w grze, czyli na przykład aktualny zapis i wyrzucone kości. Przejścia określają czynności wykonane przez gracza, takie jak rzut kośćmi, wybór kości do rzutu, czy wybór i zapis kategorii.

Każdy stan gry możemy opisać na dwóch poziomach szczegółowości. Na pierwszym poziomie opisujemy zapis gry. Określamy które kategorie są zapisane, ile punktów jest wpisanych w zużytych kategoriach oraz liczbę przyznanych premii za Jokery. Na drugim, bardziej szczegółowym poziomie, opisujemy sytuacje w danej rundzie. Określamy tutaj liczbę pozostałych rzutów, wartości wyrzuconych, bądź zatrzymanych kości. Podsumowując, stan gry określamy podając *opis zapisu* i *opis rundy*.

Stany gry podzielimy na dwa rodzaje. Stany należące do pierwszego rodzaju nazwiemy *stanami zapisu*. Stany te reprezentują sytuację pomiędzy kolejnymi rundami. Występują one zaraz po wykonaniu zapisu i przed pierwszym rzutem w następnej rundzie (o ile gra nie jest już zakończona). Do określenia stanu zapisu wystarczy sam opis zapisu. Stany należące do drugiego rodzaju nazwiemy *stanami rundy* i reprezentują one sytuacje w trakcie trwania rundy. Występują one, gdy rzuciliśmy kośćmi, bądź wybraliśmy kości do następnego rzutu.

Przejście z jednego stanu zapisu do drugiego stanu zapisu z jedną zapisaną kategorią więcej, wymaga przejścia przez stany rundy. W grafie przejść, rodzaje stanów przeplatają się warstwami. Jest 14 warstw stanów zapisu. Każda warstwa odpowiada liczbie zapisanych kategorii, od 0 do 13. Pomiędzy nimi jest 13 warstw stanów rundy. Rozbicie przestrzeni stanów na dwa rodzaje powoduje, że wszelakie analizy, wykonywane dla stanów, mogą być robione osobno dla każdego rodzaju. W celu policzenia wartości dla stanów rundy wystarczy znać

wartości dla odpowiednich stanów zapisu. Ponieważ stanów zapisu jest znacznie mniej, więc wystarczy zapisywać wartości tylko dla tych stanów, a wartości dla stanów rundy będzie można doliczać w miarę potrzeb na bieżąco. Liczba stanów zapisu będzie decydowała o potrzebnych zasobach pamięciowych.

3.1.1 Opis i stany zapisu

Pełny opis zapisu określa ile punktów zostało zapisanych w poszczególnych kategoriach oraz ile razy została przyznana premia za Jokera. Do jednoznacznego sformułowania strategii gry taka pełna informacja nie jest potrzebna. Wystarczy pamiętać sumaryczną liczbę zdobytych punktów (w tym premie) oraz *możliwości zapisowe*, czyli jakie są dalsze możliwości punktowania.

O możliwościach zapisowych decydują wolne kategorie oraz szanse na uzyskanie premii. Wolnymi kategoriami są wszystkie niezapisane kategorie. Szansę uzyskania premii za szkółkę określa liczba punktów zdobytych w sekcji górnej. Jeśli jest to liczba większa lub równa 63, oznacza to, że premia została już przyznana, zatem wystarczy pamiętać jedną liczbę ze zbioru $\{0, 1, \ldots, 63\}$. Premia za Jokera jest możliwa tylko wtedy, gdy w kategorii Yahtzee widnieje niezerowy zapis. Zatem, jeśli ta kategoria jest zapisana, to dodatkowo trzeba pamiętać, czy jest to zapis zerowy. Podsumowując, następujące informacje określają możliwości zapisowe:

- dla każdej kategorii oprócz Yahtzee: czy jest zapisana, czy też nie;
- dla kategorii Yahtzee: czy jest w niej zapis zerowy/niezerowy, czy też kategoria ta jest wolna;
- częściówka na premię za szkółkę: liczba ze zbioru $\{0, 1, \ldots, 63\}$.

Pierwsze dwie informacje nazwiemy *opisem kategorii*, a trzecią po prostu *częściówką*.

Opis kategorii

Na opis kategorii potrzebujemy, dla każdej kategorii oprócz Yahtzee, wartości binarnej, a dla Yahtzee trzech wartości. Wszystkich opisów kategorii jest więc $3 \cdot 2^{12}$. W implementacji opis kategorii jest reprezentowany liczbą 14–bitową, w której dwa najstarsze bity reprezentują jeden z trzech możliwych stanów zapisu w kategorii Yahtzee.

Przyjmiemy parę technicznych oznaczeń w celu przedstawiania opisów kategorii. Niech $K_g = \{1, 2, 3, 4, 5, 6\}$ oznacza zbiór dozwolonych kategorii górnych, gdzie każda cyfra oznacza odpowiadającą jej kategorię górną. Niech $K_d = \{\mathrm{T}, \mathrm{K}, \mathrm{F}, \mathrm{M}, \mathrm{D}, \mathrm{Y}, \mathrm{S}\}$ oznacza zbiór dozwolonych kategorii dolnych, gdzie odpowiednie literki biorą się z pierwszych liter nazw odpowiednich kategorii dolnych. Niech $K_w = K_g \cup K_d$ oznacza zbiór reprezentujący wszystkie kategorie, które są możliwe do wybrania do zapisu. W danym stanie gry do reprezentowania możliwych kategorii do zapisania będziemy używać podzbiorów K_w. Przez $\mathcal{K}_w$ oznaczmy wszystkie możliwe reprezentacje dostępnych wolnych kategorii:

$$\mathcal{K}_w = \{K \mid K \subseteq K_w\}.$$

Do opisu kategorii będziemy też używać zbiorów, które będą zawierały symbole poszczególnych kategorii. Jednak w przypadku kategorii Yahtzee potrzebujemy rozróżniać, czy to był zapis zerowy, czy też niezerowy. W tym celu zamiast jednego symbolu Y będziemy używać odpowiednio symbolów Y_0 i Y_{50}. Zatem opis kategorii będzie reprezentowany jako podzbiór zbioru $K_w \setminus \{\mathrm{Y}\}$ plus ewentualnie jeden z symboli Y_0 i Y_{50}. Przez $\mathcal{K}$ oznaczmy wszystkie możliwe reprezentacje opisu kategorii:

$$\mathcal{K} = \big\{K \mid K \subseteq K_w \cup \{\mathrm{Y}_0\} \setminus \{\mathrm{Y}\} \vee K \subseteq K_w \cup \{\mathrm{Y}_{50}\} \setminus \{\mathrm{Y}\}\big\}.$$

Dla wygody w opisie kategorii będziemy pomijać przecinki między elementami zbioru oraz klamry oznaczające zbiór.

Przykład 3.1. Jeśli mamy zapisane Trójki, Ful i Yahtzee na 0, to opisem kategorii jest $3\mathrm{FY}_0$.

W danym stanie zapisu, z opisem kategorii $K \in \mathcal{K}$, często przydatna będzie informacja o tym jakie kategorie są wolne. Niech Kategorie-Wolne$(K) \in \mathcal{K}_w$ oznacza taki zbiór kategorii wolnych.

Częściówki

Możliwych częściówek jest $64 = 2^6$. Wszystkich możliwości zapisowych jest więc: $3 \cdot 2^{18} = 786\,432$. Liczba ta może być znacznie zmniejszona, jeśli zauważymy dwie własności.

1. Tylko niektóre częściówki są osiągalne ze stanu początkowego, tzn. od sytuacji, w której wszystkie kategorie są wolne. Na przykład, jeśli jedyną zapisaną kategorią w sekcji górnej będą Dwójki, to możliwe częściówki są ze zbioru: $\{0, 2, 4, 6, 8, 10\}$.

2. Podobna własność zachodzi „od tyłu". Na przykład, jeśli jedyną wolną kategorią w sekcji górnej będą Czwórki, to nie ma znaczenia czy mamy 62, 61, 60, czy też 59 punktów na premię za szkółkę, gdyż i tak musimy rzucić jedną czwórkę, aby przekroczyć próg 63. Zatem można utożsamić ze sobą te częściówki, które dają dokładnie takie same możliwości uzyskania premii za szkółkę.

Opiszemy, w jaki sposób ograniczyć zbiór częściówek dla zadanego układu dostępnych kategorii. Zauważmy, że aby wyznaczyć istotne częściówki, interesuje nas tylko stan kategorii z sekcji górnej. Stan ten symbolicznie możemy oznaczyć zbiorem $S \subseteq \{1, 2, 3, 4, 5, 6\}$.

Po pierwsze wyznaczmy zbiór osiągalnych częściówek z S. Dla ścisłości pojęcie osiągalności częściówki przedstawione jest w definicji 3.2.

Definicja 3.2. Częściówka c jest *osiągalna* w stanie S wtedy i tylko wtedy, gdy istnieją $w_s \in \{0, 1, \ldots, 5\}$ dla $s \in S$ takie, że

$$\sum_{s \in S} w_s s = c.$$

To czy dana częściówka jest osiągalna można rozwiązać za pomocą programowania dynamicznego, tak jak się rozwiązuje dyskretny problem plecakowy (patrz [CLR97, Zadanie 17.2-2]). Wyliczanie osiągalnych częściówek przedstawione jest

CZĘŚCIÓWKI-OSIĄGALNE(S)
argumenty: zbiór kategorii w sekcji górnej $S \subseteq \{1, 2, 3, 4, 5, 6\}$
wynik: zbiór osiągalnych częściówek $C \subseteq \{0, 1, \ldots, 63\}$ ze stanu S

```
1: C ← {0}
2: foreach s ∈ S do
3:     C' ← C
4:     foreach c ∈ C' do
5:         for w ← 1 to 5 do
6:             C ← C ∪ {min(c + ws, 63)}
7: return C
```

Algorytm 3.1: Wyliczenie osiągalnych częściówek dla danego zbioru kategorii górnych.

w algorytmie 3.1. W tym algorytmie zakładamy, że liczba 63, czyli próg premii, symbolicznie oznacza dowolną częściówkę nie mniejszą od 63. Zbiór C wygenerowany przez CZĘŚCIÓWKI-OSIĄGALNE będziemy też reprezentować przez rosnący ciąg $c_1, \ldots, c_{|C|}$. Pozwoli to nam wygodnie przeglądać wartości zbioru od najmniejszej do największej lub na odwrót. W praktyce można stablicować wszystkie wyniki funkcji CZĘŚCIÓWKI-OSIĄGALNE, dla wszystkich możliwych argumentów, gdyż jest ich tylko 64.

Po drugie, możemy określić, które częściówki można utożsamić, tzn. które częściówki dają te same możliwości zapisowe. Dwie częściówki są równoważne, jeżeli sposoby osiągnięcia progu premii za szkółkę dla obu częściówek są takie same.

Definicja 3.3. Oznaczmy zbiór zapisanych kategorii przez S, zbiór wolnych kategorii przez $\overline{S} = \{1, \ldots, 6\} \setminus S$, oraz częściówki przez c_1 i c_2. Wówczas mówimy, że częściówki c_1 i c_2 są *równoważne* przy stanie S, jeżeli dla wszystkich możliwych wyborów $w_{\overline{s}} \in \{0, 1, \ldots, 5\}$ po $\overline{s} \in \overline{S}$, zachodzi

$$c_1 + \sum_{\overline{s} \in \overline{S}} w_{\overline{s}} \overline{s} \geqslant 63 \quad \text{wtedy i tylko wtedy, gdy} \quad c_2 + \sum_{\overline{s} \in \overline{S}} w_{\overline{s}} \overline{s} \geqslant 63.$$

Suma $\sum_{\overline{s} \in \overline{S}} w_{\overline{s}} \overline{s}$ jest tak na prawdę pewną częściówką $\overline{c}$ osiągalną w stanie $\overline{S}$, więc z definicji 3.3 bezpośrednio wynika proste kryterium na równoważność dwóch częściówek.

Fakt 3.4. *Częściówki $c_1 < c_2$ są równoważne w stanie S wtedy i tylko wtedy, gdy nie istnieje takie $\overline{c}$ osiągalne w stanie $\overline{S}$, że $c_1 + \overline{c} < 63 \leqslant c_2 + \overline{c}$.*

Z powyższego faktu wynika też następująca własność.

Fakt 3.5. *Dane są częściówki $c_1 < c_2 < c_3$. Jeżeli c_1 jest równoważna c_3, to c_2 jest równoważna c_1 i c_3.*

Niech $NRC_S[c]$, dla $c \in \{0, \ldots, 63\}$, oznacza *najmniejszą równoważną osiągalną częściówkę* w stanie S. Na podstawie faktów 3.4 i 3.5 możemy wygenerować tę tablice na przykład tak, jak jest to przedstawione w algorytmie 3.2. Tablica ta będzie przydatna do szybkiego znajdowania częściówek równoważnych.

GENERUJ-NRC(S)
argumenty: zbiór kategorii w sekcji górnej S
wynik: wypełniona tablica $NRC_S[c]$ dla c osiągalnych w stanie S

```
1: C ← CZĘŚCIÓWKI-OSIĄGALNE(S)
2: C̄ ← CZĘŚCIÓWKI-OSIĄGALNE(S̄)
3: ī ← |C̄|
4: for i ← 1 to |C| do
5:     NRC_S[c_i] ← c_i
6:     while c̄_ī > 0 ∧ c_i + c̄_{ī−1} ⩾ 63 do
7:         ī ← ī − 1
8:     if c_i > 0 ∧ (c_i + c̄_ī < 63 ∨ c_{i−1} + c̄_ī ⩾ 63) then
9:         NRC_S[c_i] ← NRC_S[c_{i−1}]
```

Algorytm 3.2: Generowanie tablicy najmniejszych równoważnych częściówek.

Podsumujmy rozważania o częściówkach. Wszystkich możliwych układów kategorii górnych i częściówek jest $2^6 \cdot 64 = 4096$. Jeśli uwzględnimy tylko częściówki osiągalne, to liczba ta zmniejszy się do 2794. Sklejając częściówki równoważne liczba istotnych częściówek dalej zmniejszy się do 1640, co daje zysk blisko 60%. Liczba wszystkich możliwości zapisowych zmniejszy się z 786 432 do $1640 \cdot 3 \cdot 2^6 = 314\,880$.

Dodatkowo, w algorytmach zamieszczonych w tej pracy możemy utożsamiać częściówkę 63 z częściówką 0. Wiąże się to z tym, że jeśli częściówka 63 jest osiągalna w stanie S, to z częściówki 0 w stanie S nie możemy już osiągnąć progu premii za szkółkę, a zatem w obu przypadkach premia już nie będzie przyznawana w przyszłości (w pierwszym przypadku jest już po prostu przyznana). Jest tak dlatego, że 63 nie może być osiągalna jednocześnie w stanie S i $\overline{S}$, gdyż $5 \cdot (1 + \ldots + 6) = 105 < 126 = 2 \cdot 63$. Algorytm 3.2 można zmodyfikować tak, aby uwzględniał równoważność częściówki 63 i 0 poprzez dodanie na końcu pętli **for** takich wierszy:

```
10:     if c_i = 63 then
11:         NRC_S[c_i] ← 0
```

Układów kategorii górnych, w których jest osiągalne 63 jest dokładnie 22, a zatem liczba istotnych częściówek może być dalej zmniejszona do 1618, a liczba możliwości zapisowych do $1618 \cdot 3 \cdot 2^6 = 310\,656$.

Reprezentacja

Opis zapisu jest trójką (K, c, p), w której

- $K \in \mathcal{K}$ reprezentuje opis kategorii,
- $c \in \{0, 1, \ldots, 63\}$ reprezentuję częściówkę,
- $p \in \{0, 1, 2, \ldots\}$ reprezentuję łączną liczbę zdobytych punktów (w tym premie).

Tę reprezentację będziemy używać do określania stanów zapisu. Para (K, c) reprezentuje możliwości zapisowe w stanie zapisu (K, c, p). Zaznaczmy, że nie wszystkie stany są osiągalne. Oznaczając przez $S = K \cap K_g$ zbiór zapisanych kategorii górnych, wiemy, że musi być $c \in$ CZĘŚCIÓWKI-OSIĄGALNE(S). Ponadto maksymalna możliwa liczba punktów do zdobycia wynosi 1575, zatem

$p \leqslant 1575$. Również nie wszystkie punktacje są osiągalne. Na przykład wynik 1574 nie jest możliwy.

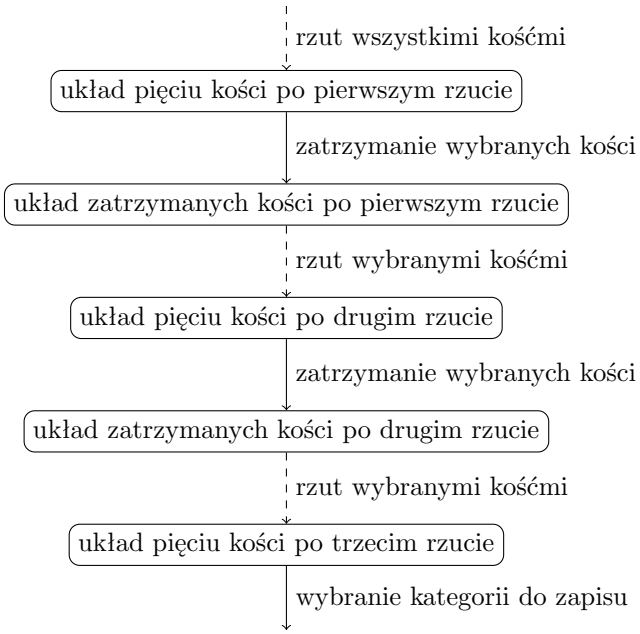
3.1.2 Opis i stany rundy

Podczas jednej rundy gracz wykonuje szereg czynności, w skład których wchodzą:

- rzucanie kośćmi,
- wybieranie kości do rzutu,
- wybieranie kategorii do zapisu.

Rzucanie jest zdarzeniem losowym, natomiast wybierania wymagają od gracza podejmowania decyzji. Jeśli przyjmiemy, że przy wybieraniu kości do rzutu jest też możliwość zatrzymania wszystkich kości, tzn. nie rzucania żadną kością, to możemy też przyjąć, że zapis jest dozwolony dopiero po wykonaniu trzeciego rzutu. Takie założenie ułatwi dalszą analizę. W stosunku do oryginalnych zasad jest to pewne odstępstwo, ale w pełni zgodne z regułami gry, gdyż zawsze możliwość zapisu po wcześniejszych rzutach można symulować poprzez rzuty bez wybranych kości, co nie zmienia układu kości.

Na rysunku 3.1 przedstawiono szczegółowo przebieg pojedynczej rundy.



Rysunek 3.1: Schemat przebiegu pojedynczej rundy.

Czynności są reprezentowane przez strzałki. Strzałki przerywane oznaczają zdarzenia losowe. Strzałki ciągłe oznaczają wybory gracza. Ponadto na rysunku zaznaczone są także stany, które powstają w trakcie rundy. Stany są dwojakiego rodzaju:

- układ kości po rzucie,
- układ zatrzymanych/wybranych kości.

Przeanalizujmy przestrzeń możliwych układów dla każdego rodzaju.

Układy kości po rzucie

Każda z pięciu kości może przyjąć sześć możliwych wartości. W najprostszym podejściu układ kości po rzucie reprezentowany jest przez ciąg $R = r_1, \ldots, r_5 \in \{1, \ldots, 6\}^5$. Liczba takich ciągów to $6^5 = 7776$. Prawdopodobieństwo wyrzucenia układu R wynosi zatem dokładnie $1/7776$.

Kości nie są rozróżnialne, więc kolejność elementów ciągu R nie ma znaczenia. Liczba kombinacji 5–elementowych zbioru 6–elementowego wynosi:

$$\binom{6+5-1}{5} = 252.$$

Rzut możemy reprezentować jako ciąg niemalejący, ale dla wygody rzut R będziemy reprezentować jako multizbiór. Elementami (być może wielokrotnymi) są liczby $1, \ldots, 6$, a rozmiar zbioru R wynosi 5.

Przykład 3.6. Reprezentacją rzutu ⚀⚀⚁⚃⚅ jest multizbiór $\{1, 1, 2, 3, 5\}$. Jedynka w takim multizbiorze ma krotność dwa.

Będziemy oznaczać, że $r \in R$, jeżeli istnieje kość w rzucie R, na której jest r oczek. Zbiór wszystkich 252 rzutów oznaczamy przez $\mathcal{R}$. Przy tych oznaczeniach, prawdopodobieństwo wyrzucenia $R \in \mathcal{R}$ jest równe liczbie różnych ciągów $r_1, \ldots, r_5$ takich, że $\{r_1, \ldots, r_5\} = R$, podzielone przez 6^5.

Przykład 3.7. Prawdopodobieństwo wyrzucenia rzutu $\{1, 1, 2, 3, 5\}$ wynosi:

$$\Pr(\{1, 1, 2, 3, 5\}) = \frac{1}{7776} \cdot \binom{5}{2} \cdot 3 \cdot 2 \cdot 1 = \frac{1}{7776} \cdot 60 = \frac{5}{648}.$$

Układy zatrzymanych kości

Każda kość może być albo zatrzymana i wtedy może przyjąć jedną z sześciu możliwych wartości, albo może być wybrana do rzutu. Zatrzymanie kości Z, podobnie jak rzuty, będziemy reprezentować przez multizbiór. Elementami Z są liczby $1, \ldots, 6$, a rozmiar zbioru Z wynosi co najwyżej 5.

Przykład 3.8. Reprezentacją zatrzymania kości ⚀⚂⚂ jest multizbiór $\{1, 3, 3\}$ o mocy 3.

Liczba wszystkich możliwych zatrzymań kości wynosi:

$$\binom{7+5-1}{5} = 462.$$

Będziemy pisali, że $z \in Z$, jeżeli istnieje kość w zatrzymaniu Z, która ma z oczek. Zbiór wszystkich 462 zatrzymań oznaczamy przez $\mathcal{Z}$.

Prawdopodobieństwo otrzymania rzutu R dla zatrzymania Z oznaczamy przez $\Pr(R|Z)$.

Przykład 3.9. Prawdopodobieństwo wyrzucenia $\{1, 3, 3, 3, 4\}$ dla zatrzymania $\{1, 3, 3\}$ wynosi tyle co prawdopodobieństwo, że dwoma kośćmi wyrzucimy ⚂⚄, czyli

$$\Pr(\{1, 3, 3, 3, 4\}|\{1, 3, 3\}) = \frac{1}{6^2} \cdot 2 = \frac{1}{18}.$$

Liczba stanów

Policzmy liczbę stanów rundy. Stan rundy wyznaczony jest przez:

- opis zapisu,
- liczbę wykonanych rzutów,
- typ stanu: rzut lub zatrzymanie,
- układ kości po rzucie, bądź układ zatrzymanych kości.

Opis zapisu musi posiadać co najmniej jedną wolną kategorię. Mamy 3 rzuty i 2 zatrzymania podczas jednej rundy. Możliwych rzutów jest 252, a zatrzymań 462, zatem dla danego opisu zapisu jest dokładnie $3 \cdot 252 + 2 \cdot 462 = 1680$ stanów rundy.

Zapis układu kości

Zmianę stanu zapisu można dokonać jedynie w wyniku zapisania danego układu kości w wybranej wolnej kategorii. Liczbę zapisanych punktów, punktów na częściówkę oraz premie regulują zasady gry. W dalszej części przyjmujemy, że mamy do dyspozycji funkcję $\textsc{Zapis}\big((K, c, p), R, k\big)$, która dla danego stanu zapisu (K, c, p), rzutu R i wybranej wolnej kategorii k zwraca nowy stan zapisu (K', c', p'). W punktacji p' uwzględnia się wszystkie przysługujące premie. Wartość argumentu p nie ma wpływu na liczbę zdobytych punktów, co można wyrazić za pomocą następującego faktu.

Fakt 3.10. *Niech* $(K', c', p') = \textsc{Zapis}\big((K, c, 0), R, k\big)$*, wtedy*

$$\textsc{Zapis}\big((K, c, p), R, k\big) = (K', c', p' + p).$$

3.2 Maksymalizacja wartości oczekiwanej wyniku

W tym punkcie zajmiemy się wyliczaniem wartości oczekiwanej wyniku dla wszystkich możliwych sytuacji podczas gry.

Zastosujemy programowanie dynamiczne. Tablicowanie wartości oczekiwanej będziemy robić na dwóch poziomach. Na poziomie zapisu będziemy wyliczać wartości dla stanów zapisu. Na poziomie rundy będziemy wyliczać wartości dla stanów rundy.

3.2.1 Poziom zapisu

Dla danego stanu zapisu (K, c, p) oznaczmy przez $E_{K,c,p}$ maksymalną wartość oczekiwaną wyniku. Na podstawie faktu 3.10 zachodzi równość

$$E_{K,c,p} = p + E_{K,c,0}. \tag{3.1}$$

Wynika to z tego, że liczba zdobytych punktów nie ma wpływu na możliwości dalszego punktowania. Do wyznaczenia wartości oczekiwanych dla stanów zapisu wystarczy, że znajdziemy wartości oczekiwane dla wszystkich możliwości zapisowych.

Będziemy budować tablicę E indeksowaną parami (K, c) po $K \in \mathcal{K}$ i $c \in \{0, 1, \ldots, 63\}$ tak, aby $E[K, c] = E_{K,c,0}$. W tym celu stosujemy podejście dynamiczne (algorytm 3.3). Najpierw inicjujemy wartość oczekiwaną na zero dla opi-

E-Poziom-Zapisu
wynik: wartości oczekiwane E dla wszystkich opisów kategorii K i częściówek osiągalnych c
1: **for** $c \leftarrow 0$ **to** 63 **do**
2: $E[\{123456\text{TKFMDY}_0\text{S}\}, c] \leftarrow 0$
3: $E[\{123456\text{TKFMDY}_{50}\text{S}\}, c] \leftarrow 0$
4: **for** $z \leftarrow 12$ **downto** 0 **do**
5: **foreach** $K \in \mathcal{K}, |K| = z$ **do**
6: E-Poziom-Zapisu-Częściówki(K)

E-Poziom-Zapisu-Częściówki(K)
1: $S \leftarrow K \cap K_g$
2: $C \leftarrow$ Częściówki-Osiągalne(S)
3: **foreach** $c \in C$ **do**
4: $E[K, c] \leftarrow$ E-Poziom-Rundy(K, c)

Algorytm 3.3: Wyliczanie wartości oczekiwanej dla stanów zapisu.

sów kategorii, które nie mają żadnych kategorii wolnych (wiersze 1–3). Następnie wartość oczekiwaną będziemy liczyć dla wszystkich opisów kategorii i osiągalnych częściówek, dla opisów kategorii o coraz większej liczbie kategorii wolnych (wiersze 4–6).

W wywołaniu E-Poziom-Zapisu-Częściówki(K) przechodzimy wszystkie częściówki osiągalne. Tam dla danego opisu kategorii K i częściówki c wartość oczekiwaną wyliczamy wywołując E-Poziom-Rundy(K, c). Funkcja E-Poziom-Rundy symuluje wszystkie czynności jakie są wykonywane podczas jednej rundy przy dochodzeniu do stanów zapisu, w których jest zapisana jedna kategoria więcej. Ta funkcja jest opisana w następnym punkcie. Zwróćmy uwagę, że w E-Poziom-Zapisu-Częściówki nie musimy liczyć wielokrotnie częściówek równoważnych. Możemy to zoptymalizować używając tablicy najmniejszych równoważnych częściówek NRC (algorytm 3.4).

E-Poziom-Zapisu-Częściówki(K)
1: $S \leftarrow K \cap K_g$
2: $C \leftarrow$ Częściówki-Osiągalne(S)
3: **for** $i \leftarrow 1$ **to** $|C|$ **do**
4: **if** $NRC_S[c_i] = c_i$ **then**
5: $E[K, c_i] \leftarrow$ E-Poziom-Rundy(K, c_i)
6: **else**
7: $E[K, c_i] \leftarrow E[K, NRC_S[c_i]]$

Algorytm 3.4: Wyliczanie wartości oczekiwanej dla stanów zapisu — optymalizacja przetwarzania częściówek.

3.2.2 Poziom rundy

Przyjrzyjmy się jakie obliczenia muszą zostać wykonane, aby wyliczyć maksymalną wartość oczekiwaną możliwą do osiągnięcia w stanie zapisu $(K, c, 0)$. Oznaczmy szukaną wartość przez *wynik*.

Pierwszą czynnością jest rzucenie wszystkimi kośćmi. Możliwa do osiągnięcia wartość oczekiwana zależna jest od tego co wyrzuciliśmy. Oznaczmy przez $rzut_1[R]$, $R \in \mathcal{R}$, maksymalną wartość oczekiwaną jaką można osiągnąć po wyrzuceniu w pierwszym rzucie R. Rzut jest zdarzeniem losowym i wynikowa wartość oczekiwana jest średnią ważoną, czyli

$$wynik = \sum_{R \in \mathcal{R}} \Pr(R) \cdot rzut_1[R]. \tag{3.2}$$

Kolejną czynnością jest zatrzymanie kości. Oznaczmy przez $zatrzymanie_1[Z]$, $Z \in \mathcal{Z}$, maksymalną wartość oczekiwaną jaką można osiągnąć po pierwszym zatrzymaniu kości Z. Dla danego rzutu R zatrzymanie Z jest możliwe jeśli $Z \subseteq R$. Wybór zatrzymanych kości ma maksymalizować wartość oczekiwaną, a zatem

$$rzut_1[R] = \max_{Z \subseteq R} zatrzymanie_1[Z]. \tag{3.3}$$

Następnie jest drugi rzut kośćmi, tym razem tylko niezatrzymanymi kośćmi. Oznaczmy przez $rzut_2[R]$, $R \in \mathcal{R}$, maksymalną wartość oczekiwaną jaką można osiągnąć po wyrzuceniu drugiego rzutu R. Zaznaczmy, że rzut R jest możliwy przy zatrzymaniu Z tylko wtedy, gdy $\Pr(R|Z) > 0$. Podobnie jak przy pierwszej czynności jest to zdarzenie losowe, a zatem, aby wyliczyć wartość oczekiwaną należy wziąć średnią ważoną:

$$zatrzymanie_1[Z] = \sum_{R \in \mathcal{R}} \Pr(R|Z) \cdot rzut_2[R]. \tag{3.4}$$

Następnie jest drugie zatrzymanie i trzeci rzut. Maksymalne wartości oczekiwane oznaczamy jak wyżej, odpowiednio przez $zatrzymanie_2[Z]$ i $rzut_3[R]$. Zależności są analogiczne do (3.3) i (3.4):

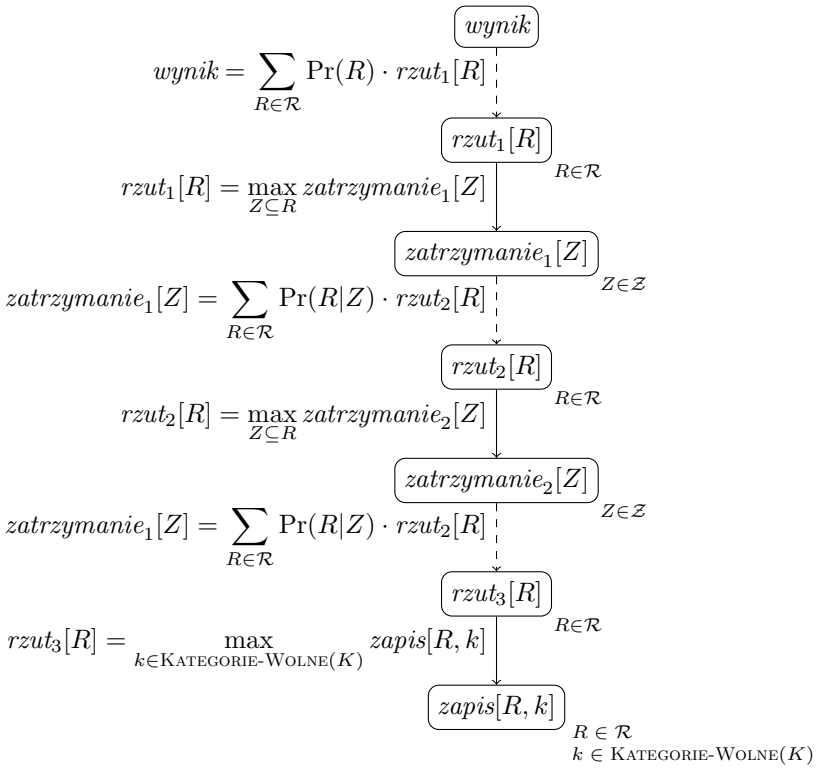
$$rzut_2[R] = \max_{Z \subseteq R} zatrzymanie_2[Z]. \tag{3.5}$$

$$zatrzymanie_2[Z] = \sum_{R \in \mathcal{R}} \Pr(R|Z) \cdot rzut_3[R]. \tag{3.6}$$

Ostatnią czynnością po trzecim rzucie jest wybór kategorii do zapisu. Oznaczając przez $zapis[R, k]$, $R \in \mathcal{R}$ i $k \in$ Kategorie-Wolne(K), maksymalną wartość oczekiwaną jaką możemy uzyskać zapisując rzut R w kategorii k, ostatnią zależnością jest:

$$rzut_3[R] = \max_{k \in \text{Kategorie-Wolne}(K)} zapis[R, k]. \tag{3.7}$$

Schemat wszystkich zależności przedstawiony jest na rysunku 3.2. Przy założeniu, że mamy dane wszystkie wartości $zapis[R, k]$, możemy wyliczyć pozostałe wartości stosując programowanie dynamiczne, czyli licząc od końca. Pseudokod przedstawia procedura Propaguj-Wartości-Stanów-Rundy w algorytmie 3.5. W wierszach 1–2 wyznaczamy $rzut_3$ na podstawie wzoru (3.7). Dalej



Rysunek 3.2: Zależności między wartościami w pojedynczej kolejce.

wyznaczamy $zatrzymanie_i$ z $rzut_{i+1}$ (wiersz 4) i $rzut_i$ z $zatrzymanie_i$ (wiersz 5) kolejno dla $i = 2$ i $i = 1$. Stosujemy tu odpowiednio funkcje Waż-Rzuty-Dla-Zatrzymań i Maksymalizuj-Zatrzymania-Dla-Rzutów, dla których podamy efektywną implementację. Na końcu Waż-Rzuty-Dla-Wszystkich-Kości wyznacza *wynik* z $rzut_1$ (wiersz 6).

Najprostszym sposobem implementacji Waż-Rzuty-Dla-Zatrzymań jest dla każdego zatrzymania $Z \in \mathcal{Z}$ użycie równań (3.4) i (3.6). Suma może być zoptymalizowana w ten sposób, że R przebiega tylko te rzuty, dla których $\Pr(R|Z) > 0$. Można sobie zawczasu przygotować odpowiednie zbiory indeksów i prawdopodobieństwa dla każdego Z. Wyznaczanie tablicy *zatrzymanie* na podstawie tablicy *rzut* można zrobić jednak znacznie efektywniej jeśli podejdziemy do tego całościowo i użyjemy kolejny już raz programowania dynamicznego.

Przyjrzyjmy się jakim dokładnie wzorem można przedstawić $zatrzymanie[Z]$ na podstawie tablicy *rzut*. Jakie są możliwe rzuty dla zatrzymania Z? Niech $w = |Z|$ i $Z = \{z_1, \ldots, z_w\}$. Wtedy rzucamy $5 - w$ kośćmi (może ich być też zero) i w wyniku otrzymujemy $z_{w+1}, \ldots, z_5$, co daje rzut $R = \{z_1, \ldots, z_5\}$, gdzie $z_{w+1}, \ldots, z_5 \in \{1, \ldots, 6\}$. Prawdopodobieństwo wyrzucenia ciągu $z_{w+1}, \ldots, z_5$ wynosi dokładnie $\frac{1}{6^{5-w}}$. Zatem otrzymujemy:

$$zatrzymanie[\{z_1, \ldots, z_w\}] = \frac{1}{6^{5-w}} \cdot \sum_{z_{w+1}, \ldots, z_5 \in \{1, \ldots, 6\}} rzut[\{z_1, \ldots, z_5\}], \quad (3.8)$$

skąd wynikają następujące wzory na liczenie tablicy *zatrzymanie*.

PROPAGUJ-WARTOŚCI-STANÓW-RUNDY(K)
argumenty: opis kategorii K i tablica *zapis*
wynik: wypełnione tablice $rzut_i$, $zatrzymanie_i$ oraz zmienna *wynik*
1: **foreach** $R \in \mathcal{R}$ **do**
2: $\quad rzut_3[R] \leftarrow \max\{zapis[R,k] \mid k \in$ KATEGORIE-WOLNE(K)$\}$
3: **for** $i \leftarrow 2$ **downto** 1 **do**
4: $\quad zatrzymanie_i \leftarrow$ WAŻ-RZUTY-DLA-ZATRZYMAŃ($rzut_{i+1}$)
5: $\quad rzut_i \leftarrow$ MAKSYMALIZUJ-ZATRZYMANIA-DLA-RZUTÓW($zatrzymanie_i$)
6: $wynik \leftarrow$ WAŻ-RZUTY-DLA-WSZYSTKICH-KOŚCI($rzut_1$)

Algorytm 3.5: Propagowanie wartości dla stanów rundy.

Fakt 3.11. *Niech $Z \in \mathcal{Z}$ i $w = |Z|$. Jeśli $w = 5$, to*

$$zatrzymanie[Z] = rzut[R], \tag{3.9}$$

w przeciwnym razie zachodzi:

$$zatrzymanie[Z] = \frac{1}{6} \cdot \sum_{z=1}^{6} zatrzymanie[Z \cup \{z\}]. \tag{3.10}$$

Dowód. Jeśli $w = 5$, to nie rzucamy żadnymi kośćmi i (3.9) wynika bezpośrednio z (3.8). Niech $w < 5$ i $Z = \{z_1, \ldots, z_w\}$, wtedy

$$\begin{aligned} zatrzymanie[\{z_1,\ldots,z_w\}] &= \frac{1}{6^{5-w}} \cdot \sum_{z_{w+1},\ldots,z_5 \in \{1,\ldots,6\}} rzut[\{z_1,\ldots,z_5\}] \\ &= \frac{1}{6} \cdot \sum_{z_{w+1}=1}^{6} \cdot \left(\frac{1}{6^{5-(w+1)}} \sum_{z_{w+2},\ldots,z_5 \in \{1,\ldots,6\}} rzut[\{z_1,\ldots,z_5\}] \right) \\ &= \frac{1}{6} \cdot \sum_{z_{w+1}=1}^{6} zatrzymanie[\{z_1,\ldots,z_w,z_{w+1}\}], \end{aligned}$$

skąd dostajemy (3.10). □

Realizacja funkcji WAŻ-RZUTY-DLA-ZATRZYMAŃ na podstawie faktu 3.11 przedstawiona jest w algorytmie 3.6. Różnica pomiędzy najprostszą metodą

WAŻ-RZUTY-DLA-ZATRZYMAŃ($rzut$)
1: **foreach** $R \in \mathcal{R}$ **do**
2: $\quad zatrzymanie[R] \leftarrow rzut[R]$
3: **for** $w \leftarrow 4$ **downto** 0 **do**
4: $\quad$ **foreach** $Z \in \mathcal{Z}, |Z| = w$ **do**
5: $\quad\quad zatrzymanie[Z] \leftarrow \frac{1}{6} \cdot \sum_{z=1}^{6} zatrzymanie[Z \cup \{z\}]$
6: **return** *zatrzymanie*

Algorytm 3.6: Wyznaczanie wartości zatrzymań na podstawie wartości dla rzutów.

polegającą na użyciu równań (3.4) i (3.6), a przedstawioną metodą, jest taka, że dla danego zatrzymania Z w prostszej metodzie suma ma rozmiar taki, jak

liczba możliwych rzutów dla danego zatrzymania, a w algorytmie 3.6 rozmiar sumy jest zawsze 6. Widać, że użycie programowania dynamicznego pozwoliło nam na implementację szybszą o rząd wielkości.

Zajmijmy się teraz implementacją funkcji Maksymalizuj-Zatrzymania-Dla-Rzutów. Najprościej jest użyć równań (3.3) i (3.5). Dla każdego rzutu R liczone jest maksimum po wszystkich zatrzymaniach $Z \subseteq R$. Liczenie tablicy *rzut* na podstawie tablicy *zatrzymanie* można zrobić jednak efektywniej. Rozszerzamy tablicę *rzut* tak, aby zbiór indeksów wynosił $\mathcal{Z}$ zamiast $\mathcal{R}$. Niech *rzut*$[Z]$ dla $Z \in \mathcal{Z}$ wynosi $\max_{Z' \subseteq Z}$ *zatrzymanie*$[Z']$. Aby wyznaczyć taką tablicę *rzut* możemy zastosować programowanie dynamiczne. Wystarczy zauważyć, że *rzut*$[Z]$ umiemy wyznaczać jak tylko znamy wartości *rzut* dla zatrzymań zawierających tylko jedną kość mniej. W wyniku otrzymujemy algorytm 3.7.

Maksymalizuj-Zatrzymania-Dla-Rzutów(*zatrzymanie*)
1: **for** $w \leftarrow 0$ **to** 5 **do**
2: **foreach** $Z \in \mathcal{Z}, |Z| = w$ **do**
3: $rzut[Z] \leftarrow \max\big(\{zatrzymanie[Z]\} \cup \{rzut[Z \setminus \{z\}] \mid z \in Z\}\big)$
4: obetnij tablicę *rzut* tak, że zbiór indeksów zmniejszy się z $\mathcal{Z}$ do $\mathcal{R}$
5: **return** *rzut*

Algorytm 3.7: Wartości rzutów.

Ostatnią funkcją jest Waż-Rzuty-Dla-Wszystkich-Kości. Jest to nic innego jak policzenie wartości dla zatrzymania pustego, na podstawie wszystkich rzutów. Można do tego celu stosować funkcję Waż-Rzuty-Dla-Zatrzymań, ale w tym wypadku jest to zbyt pracochłonne ze względu na zbędne liczenie wartości również dla pozostałych zatrzymań. Najprostszą implementacją jest użycie wzoru (3.2) (algorytm 3.8).

Waż-Rzuty-Dla-Wszystkich-Kości(*rzut*)
1: **return** $\sum_{R \in \mathcal{R}} \Pr(R) \cdot rzut[R]$

Algorytm 3.8: Wartości rzutów.

Mamy funkcję, która propaguje wartości stanów rundy na podstawie tablicy *zapis*. Teraz jedyne czego jeszcze potrzeba to inicjacja tej tablicy. Wartości *zapis*$[R, k]$ wyliczamy na podstawie zasad gry i wyznaczonych wcześniej wartości oczekiwanych dla stanów zapisu z większą liczbą zapisanych kategorii. Następnie używamy procedury Propaguj-Wartości-Stanów-Rundy(K). W ten sposób realizujemy liczenie wartości oczekiwanej na poziomie rundy, co jest przedstawione w algorytmie 3.9.

3.2.3 Uwagi implementacyjne i wyniki

Przedstawiony algorytm tablicowania wartości $E[K, c]$ był testowany z różną dokładnością liczb zmiennoprzecinkowych i wykazuje się dużą stabilność numeryczną. To znaczy obliczenia z większą dokładnością potwierdzały wyniki uzyskane z mniejszą dokładnością.

Do uzyskiwania informacji o maksymalnej wartości oczekiwanej w dowolnym momencie gry wystarczy, że będziemy pamiętać tylko tablicę E. Jeśli będziemy potrzebowali wartości oczekiwanej na przykład dla danego rzutu pod-

E-Poziom-Rundy(K, c)
argumenty: opis kategorii K i częściówka c
wynik: wartość oczekiwana wyniku uzyskanego w stanie zapisu $(K, c, 0)$

```
1: foreach R ∈ 𝓡 ∧ k ∈ Kategorie-Wolne(K) do
2:     (K', c', p') ← Zapis((K, c, 0), R, k)
3:     zapis[R, k] ← p' + E[K', c']
4: Propaguj-Wartości-Stanów-Rundy(K)
5: return wynik
```

Algorytm 3.9: Wyliczanie wartości oczekiwanej na poziomie rundy.

czas rundy, wtedy za pomocą funkcji E-Poziom-Rundy uzyskujemy praktycznie natychmiast szukane wartości. Co więcej podczas normalnej gry, w której chcemy maksymalizować wartość oczekiwaną wyniku, dla danej rundy funkcję E-Poziom-Rundy możemy wywołać tylko raz, a w czasie całej gry tylko 13 razy. Strategia na maksymalną wartość oczekiwaną jest zaimplementowana w algorytmie 3.10.

Strategia-E$(K, c, rzuty, R)$
argumenty: opis kategorii K, częściówka c, liczba wykonanych rzutów *rzuty* oraz aktualny rzut R
wynik: posunięcie maksymalizujące wartość oczekiwaną wyniku w stanie zapisu $(K, c, 0)$ — zatrzymanie kości $Z \subseteq R$, bądź zapisywana kategoria $k \in$ Kategorie-Wolne(K)

```
1: foreach R ∈ 𝓡 ∧ k ∈ Kategorie-Wolne(K) do
2:     (K', c', p') ← Zapis((K, c, 0), R, k)
3:     zapis[R, k] ← p' + E[K', c']
4: Propaguj-Wartości-Stanów-Rundy(K)
5: if rzuty < 3 then
6:     return Z ⊆ R maximizing zatrzymanie_rzuty[Z]
7: else
8:     return k ∈ Kategorie-Wolne(K) maximizing zapis[R, k]
```

Algorytm 3.10: Strategia na maksymalna wartość oczekiwaną.

Przedstawione algorytmy tablicowania są bardzo efektywne. Stablicowanie wartości oczekiwanych przez [Ver99] zajmuje niecałe 10 minut. Na podobnej maszynie (tj. Pentium 4, 2.8 GHz), przy użyciu naszego programu, tablicowanie trwało niecałe 35 sekund. Wartość oczekiwana gry wynosi 254.589374.

3.3 Maksymalizacja prawdopodobieństwa osiągnięcia określonego wyniku

Strategia maksymalizowania wartości oczekiwanej jest optymalną strategią, jeżeli chcemy średnio zdobywać jak najwięcej punktów. Możemy sobie stawiać inny cel. Otóż możemy chcieć pobić najlepsze wyniki uzyskane dotychczas w grze jednoosobowej. Ustalamy sobie wynik jaki minimalnie chcielibyśmy osiągnąć i chcemy zmaksymalizować swoje szanse osiągnięcia tego wyniku. W tym punkcie zajmiemy się wyliczeniem maksymalnych prawdopodobieństw osiągnięcia

każdego z wyników dla każdego stanu.

W przeciwieństwie do liczenia wartości oczekiwanej, do obliczenia największego prawdopodobieństwa określonego wyniku dla danego stanu nie wystarczy pamiętać jednej liczby. Załóżmy, że chcemy wyliczyć maksymalne prawdopodobieństwo osiągnięcia wyniku x ze stanu (K, c, p). W tym celu potrzebujemy maksymalne prawdopodobieństwa osiągnięcia wyniku x ze wszystkich stanów do których możemy dojść przez wykonanie pojedynczego zapisu, tj. stanów (K', c', p') takich, że $(K', c', p') = \textsc{Zapis}\big((K, c, p), R, k\big)$ po wszystkich rzutach $R \in \mathcal{R}$ i kategoriach wolnych $k \in \textsc{Kategorie-Wolne}(K)$. Dla danego opisu kategorii K' i częściówki c' na ogół można osiągnąć kilka różnych punktacji p'. Dla każdej z nich musimy wyliczać prawdopodobieństwo osiągnięcia wyniku x osobno. W przypadku wartości oczekiwanej wystarczyło wyliczać wartość dla stanów $(K, c, 0)$ ze względu na wzór (3.1). Takiej własności jednak nie mają maksymalne prawdopodobieństwa osiągnięcia wyniku x. Tym razem trzeba na pewno wyliczyć wartości dla wszystkich osiągalnych stanów zapisu (K, c, p). Takie podejście było stosowane w [Cre02]. Tutaj będziemy robić to bardziej ogólnie przez wprowadzenie pojęcia dystrybucji.

Mianowicie, dla każdej możliwości zapisowej (K, c) będziemy liczyć maksymalne prawdopodobieństwa uzyskania wszystkich możliwych wyników osiągalnych z tego stanu. Tę informację można reprezentować funkcją $P : \mathbb{Z} \mapsto [0, 1]$, która dla danego $x \in \mathbb{Z}$ zwraca maksymalne prawdopodobieństwo $P(x)$ osiągnięcia wyniku co najmniej x. Taka funkcja przypomina dystrybucję rozkładu prawdopodobieństwa. Funkcja ta ma tę własność, że im większe x tym mniejsze $P(x)$. W związku z tym, na potrzeby tej pracy, przyjmiemy następującą definicję dystrybucji.

Definicja 3.12. *Dystrybucją* nazywamy nierosnącą funkcję $P : \mathbb{Z} \mapsto [0, 1]$, tzn. taką, że dla każdego $x_1, x_2 \in \mathbb{Z}$ i $x_1 \leqslant x_2$ zachodzi $P(x_1) \geqslant P(x_2)$.

Ponieważ w zastosowaniach do Yahtzee, dla prawie wszystkich x dystrybucje przyjmują wartość 1 lub 0, więc na pamiętanie ich potrzeba pamięci o skończonym rozmiarze. Dla danej dystrybucji P, jeżeli $x_0 \in \mathbb{Z}$ jest najmniejsze takie, że $P(x_0) < 1$, oraz $x_1 \in \mathbb{Z}$ jest największe takie, że $P(x_1) > 0$, to do pamiętania P wystarczy przechowywać x_0 oraz wektor wartości $[P(x_0), P(x_0 + 1), \ldots, P(x_1)]$. x_0 będziemy nazywać punktem początkowym. Będziemy utożsamiać P z parą punkt/wektor: $P \equiv \big(x_0, [P(x_0), \ldots, P(x_1)]\big)$.

Podobnie jak dla wartości oczekiwanej, do wyliczenia maksymalnych prawdopodobieństw zastosujemy programowanie dynamiczne na dwóch poziomach. Obliczenia okazują się być identyczne jak w przypadku tablicowania wartości oczekiwanych. Różnią się tylko tym, że wartości oczekiwane zastępujemy dystrybucjami. Jedyne co należy zrobić to inicjację stanów końcowych z użyciem dystrybucji oraz zdefiniować operacje na dystrybucjach.

3.3.1 Poziom zapisu

Niech $P_{K,c,p}$ będzie dystrybucją dla stanu zapisu (K, c, p), tzn. prawdopodobieństwo osiągnięcia wyniku x ze stanu (K, c, p) wynosi $P_{K,c,p}(x)$. Liczba zdobytych punktów p nie ma wypływu na dalsze możliwości punktowania (Fakt 3.10), a jedynie mówi, jak dużo punktów już mamy. Zachodzi równość:

$$P_{K,c,p}(x + p) = P_{K,c,0}(x). \tag{3.11}$$

Definicja 3.13. Niech P będzie dystrybucją oraz niech $p \in \mathbb{Z}$. Przez *dystrybucję przesuniętą* $P + p$ będziemy oznaczać taką dystrybucję, że dla $x \in \mathbb{Z}$ zachodzi:

$$(P + p)(x + p) = P(x).$$

Z pomocą definicji 3.13 wzór (3.11) możemy wyrazić za pomocą następującego faktu:

Fakt 3.14. *Dla stanu zapisu* (K, c, p) *zachodzi:*

$$P_{K,c,p} = P_{K,c,0} + p.$$

Z faktu 3.14 wynika, że wystarczy stablicować $P_{K,c,0}$ dla wszystkich możliwości zapisowych (K, c). Będziemy zatem budować tablicę P indeksowaną parami (K, c) po $K \in \mathcal{K}$ i $c \in \{0, 1, \ldots, 63\}$ taką, że $P[K, c] = P_{K,c,0}$.

Wyliczanie dystrybucji na poziomie zapisu różni się od wyliczania wartości oczekiwanych (algorytm 3.3) tylko inicjacją dla stanów końcowych. Oznaczając przez θ_y dystrybucję

$$\theta_y(x) = \begin{cases} 1 & \text{dla } x \leqslant y, \\ 0 & \text{dla } x > y, \end{cases}$$

wyliczanie P na poziomie zapisu przedstawione jest w algorytmie 3.11. Tym razem przechodzenie częściówek podane jest od razu w wersji zoptymalizowanej.

P-Poziom-Zapisu
wyjście: dystrybucje P osiągalnych wyników dla wszystkich opisów kategorii K i częściówek osiągalnych c

```
1: for c ← 0 to 63 do
2:     P[123456TKFMDY_0S, c] ← θ_0
3:     P[123456TKFMDY_50S, c] ← θ_0
4: for z ← 12 downto 0 do
5:     foreach K ∈ 𝒦, |K| = z do
6:         P-Poziom-Zapisu-Częściówki(K)
```

P-Poziom-Zapisu-Częściówki(K)

```
1: S ← K ∩ K_g
2: C ← Częściówki-Osiągalne(S)
3: for i ← 1 to |C| do
4:     if NRC_S[c_i] = c_i then
5:         P[K, c_i] ← P-Poziom-Rundy(K, c_i)
6:     else
7:         P[K, c_i] ← P[K, NRC_S[c_i]]
```

Algorytm 3.11: Wyliczanie dystrybucji osiągalnych wyników na poziomie zapisu.

3.3.2 Poziom rundy

Tutaj analiza przebiega bardzo podobnie jak przy wyliczaniu wartości oczekiwanej, z tą różnicą, że stosujemy dystrybucje. Wprowadzamy analogiczne oznaczenia:

- *wynik* — szukana dystrybucja w danym stanie zapisu $(K, c, 0)$, czyli wartość $\mathit{wynik}(x)$ oznacza prawdopodobieństwo uzyskania wyniku co najmniej x w tym stanie,
- $\mathit{rzut}_i[R]$, $i = 1, 2, 3$, $R \in \mathcal{R}$ — dystrybucja po wykonaniu i-tego rzutu R,
- $\mathit{zatrzymanie}_i[Z]$, $i = 1, 2$, $Z \in \mathcal{Z}$ — dystrybucja po i-tym zatrzymaniu Z,
- $\mathit{zapis}[R, k]$, $R \in \mathcal{R}$, $k \in$ Kategorie-Wolne(K) — dystrybucja po zapisaniu rzutu R w kategorii k.

Podczas rundy występują dwa rodzaje zdarzeń: losowe lub wybór gracza. Przyjrzyjmy się ogólnie jak traktować oba rodzaje zdarzeń za pomocą dystrybucji.

W przypadku zdarzenia losowego, załóżmy, że zbiór możliwych zdarzeń to A. Ponadto załóżmy, że dla każdego $a \in A$ mamy już wyliczoną dystrybucję wyników P_a oraz, że prawdopodobieństwo zdarzenia a wynosi $\Pr(a)$. Jaka jest dystrybucja wynikowa P? Otóż prawdopodobieństwo uzyskania wyniku co najmniej x zgodnie z prawami rachunku prawdopodobieństwa wyraża się wzorem:

$$P(x) = \sum_{a \in A} \Pr(a) \cdot P_a(x), \tag{3.12}$$

skąd wywodzi się definicja na dystrybucję ważoną.

Definicja 3.15. Dystrybucję P taką, że dla każdego $x \in \mathbb{Z}$ zachodzi (3.12), nazywamy *dystrybucją ważoną* dystrybucji P_a po $a \in A$ i zapisujemy:

$$P = \sum_{a \in A} \Pr(a) \cdot P_a.$$

W przypadku wyboru gracza, załóżmy, że zbiór możliwych wyborów to B. Ponadto załóżmy, że dla każdego $b \in B$ mamy już wyliczoną dystrybucję wyników P_b. Jaka jest w tym wypadku dystrybucja wynikowa P? Otóż, jeśli chcemy osiągnąć wynik x z maksymalnym prawdopodobieństwem, to powinniśmy wybrać takie b, dla którego $P_b(x)$ jest największe, czyli

$$P(x) = \max_{b \in B} P_b(x), \tag{3.13}$$

skąd definicja dystrybucji maksimum:

Definicja 3.16. Dystrybucję P taką, że dla każdego $x \in \mathbb{Z}$ zachodzi (3.13), nazywamy *dystrybucją maksimum* dystrybucji P_b po $b \in B$ i zapisujemy:

$$P = \max_{b \in B} P_b.$$

Używając oznaczeń z definicji 3.15 i 3.16 na dystrybucję ważoną i maksimum, po analogicznej analizie rundy jak przy wartości oczekiwanej, otrzymamy ten sam schemat zależności między poszczególnymi wartościami, przedstawiony na rysunku 3.2. W rezultacie otrzymujemy algorytm analogiczny do algorytmu 3.9 liczący dystrybucje prawdopodobieństw. Stosowna procedura przedstawiona jest jako algorytm 3.12. Zauważmy, że w wierszu 3 użyliśmy dystrybucji przesuniętej (definicja 3.13). Ponadto w Propaguj-Wartości-Stanów-Rundy(K) pracujemy nie na wartościach oczekiwanych, a na dystrybucjach, tzn. w miejscach, gdzie ważymy wartości lub bierzemy maksimum, to stosujemy odpowiednio ważenie lub maksimum z dystrybucji.

P-Poziom-Rundy(K, c)
argumenty: opis kategorii K i częściówka c
wynik: dystrybucja osiągalnych wyników w stanie $(K, c, 0)$
1: **foreach** $R \in \mathcal{R} \wedge k \in$ Kategorie-Wolne(K) **do**
2: $\quad (K', c', p') \leftarrow$ Zapis$\big((K, c, 0), R, k\big)$
3: $\quad zapis[R, k] \leftarrow P[K', c'] + p'$
4: Propaguj-Wartości-Stanów-Rundy(K)
5: **return** *wynik*

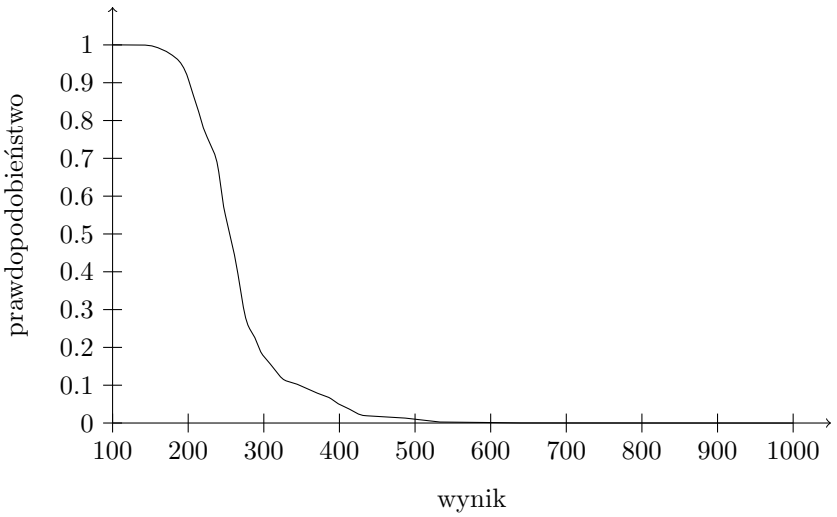
Algorytm 3.12: Wyliczanie dystrybucji osiągalnych wyników na poziomie rundy.

3.3.3 Uwagi implementacyjne i wyniki

Ponieważ największy możliwy wynik wynosi 1575 (13 kolejnych Yahtzee), więc w najgorszym przypadku trzeba pamiętać 1575 wartości dla jednej dystrybucji. W praktyce wiele prawdopodobieństw jest bardzo bliskie 1 lub 0. W implementacji dodatkowo „obcinaliśmy" dystrybucję, gdy prawdopodobieństwo różniło się od 1 lub 0 o nie więcej niż 2^{-32}, tj. przyjmowaliśmy, że w tym przypadku prawdopodobieństwo wynosiło odpowiednio 1 lub 0.

Wyliczenie dystrybucji dla wszystkich możliwości zapisowych zajęło niecałe 6 godzin na Pentium 4 z zegarem 2.8 GHz. Zajmują one na dysku 1.3 GB. Wyliczenie dystrybucji dla jednego stanu zapisu za pomocą funkcji P-Poziom-Rundy zajmuje od kilku milisekund do 150 milisekund, w zależności od liczby wolnych kategorii. W praktyce oznacza to, że przy strategii na maksymalizowanie prawdopodobieństwa osiągnięcia określonego wyniku jesteśmy w stanie natychmiast podejmować decyzje. Podczas gry, dla każdej rundy, wyliczamy dystrybucje dla wszystkich stanów rundy w czasie mniejszym niż 150 ms, a następnie po każdym rzucie podejmujemy decyzję korzystając z gotowych wartości.

Dla stanu początkowego $(\emptyset, 0, 0)$, czyli dla stanu, w którym wszystkie kategorie są wolne, jest zero punktów na częściówkę i całkowity wynik wynosi 0, dystrybucja przedstawiona jest na rysunku 3.3. Najmniejszy i największy wynik,



Rysunek 3.3: Dystrybucja dla początku gry.

który nie został obcięty to odpowiednio 80 i 1062. Największy wynik, jaki można osiągnąć z prawdopodobieństwem co najmniej $\frac{1}{2}$, to 254 i $P_{\emptyset,0,0}(254) \approx 0.508$. Zauważmy, że jeśli popatrzymy na tą dystrybucję jak na rozkład prawdopodobieństwa i weźmiemy wartość oczekiwaną, to wyniesie ona 264.449, a to jest więcej niż 254.589 — wartość uzyskana dla strategii optymalnej na maksymalną wartość oczekiwaną. Widzimy więc, że ta dystrybucja nie może reprezentować rozkładu prawdopodobieństwa żadnej strategii. Jest ona jedynie użyteczna do określania tego, jakie jest maksymalne prawdopodobieństwo osiągnięcia pewnego ustalonego wyniku.

W danej sytuacji na ogół nie ma jednego najlepszego posunięcia. To znaczy, aby zmaksymalizować prawdopodobieństwo wygranej dla różnych wyników może być potrzeba wykonania różnych posunięć.

Przykład 3.17. Weźmy sytuację z rzeczywistej gry. Niech stanem zapisu będzie $(2456\mathrm{TMDY}_0\mathrm{S}, 51, 170)$. Wykonaliśmy już ostatni trzeci rzut i mamy rzut ⚀⚁⚁⚃⚃. W tabeli 3.1 przedstawione są prawdopodobieństwa osiągnięcia wyników 200, 207, 242 i 251 oraz maksymalna wartość oczekiwana dla wszystkich możliwych posunięć. Widać, że każde z posunięć może być optymalne zależnie od tego jaki jest cel strategii.

	E	200	207	242	251
zapisz Jedynki	**206.064**	0.704	0.392	0.146	**0.030**
zapisz Trójki	204.382	**0.787**	0.367	0.000	0.000
zapisz Karetę	205.293	0.711	0.479	**0.185**	0.000
zapisz Fula	200.615	0.517	**0.484**	0.078	0.002

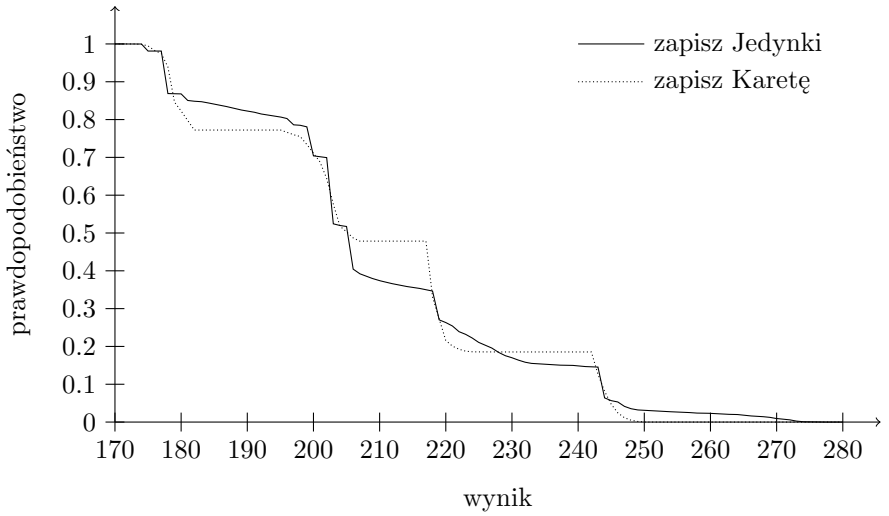
Tabela 3.1: Wartość oczekiwana oraz prawdopodobieństwa osiągnięcia niektórych wyników dla poszczególnych posunięć po trzecim rzucie ⚀⚁⚁⚃⚃ z opisem zapisu $(2456\mathrm{TMDY}_0\mathrm{S}, 51, 170)$.

Dystrybucje mogą też się przeplatać, tzn. w celu maksymalizacji prawdopodobieństwa osiągnięcia coraz większych wyników może być optymalne raz jedno posunięcie, a raz drugie.

Przykład 3.18. Dla sytuacji z przykładu 3.17 takimi dystrybucjami są dystrybucje dla posunięć zapisz Jedynki i zapisz Karetę (rysunek 3.4).

Strategia na maksymalizowanie prawdopodobieństwa osiągnięcia określonego wyniku została tak naprawdę stworzona w celu bicia rekordowych wyników. Ze względu na bonus w wysokości 100 punktów za kolejne Yahtzee, rekordy osiągane są po prostu przez wyrzucenie kilku Yahtzee. Chcąc pobić rekord w jakimś serwisie internetowym, to na początku próbujemy od razu wyrzucić Yahtzee raz, dwa lub więcej razy. Jeżeli nie uda się, to przerywamy grę i rozpoczynamy od nowa. Przyjrzyjmy się jakie są szanse osiągnięcia rekordowych wyników po wyrzuceniu jednego, bądź dwóch Yahtzee na początku. W tabeli 3.2 przedstawione są prawdopodobieństwa osiągnięcia niektórych wyników z trzech różnych sytuacji:

- z początku gry — stan przed rozpoczęciem gry, opis zapisu: $(\emptyset, 0, 0)$,
- po jednym Yahtzee — stan po zapisaniu w pierwszej rundzie 50 punktów w kategorii Yahtzee, opis zapisu: $(\mathrm{Y}_{50}, 0, 50)$,



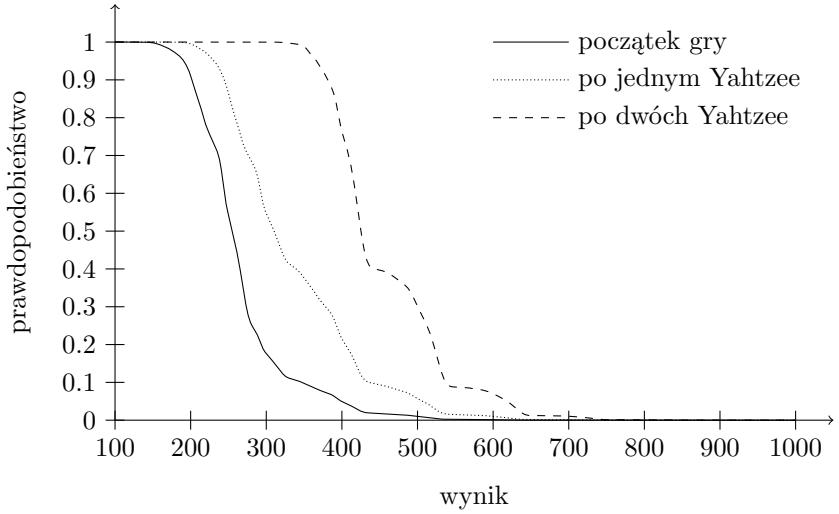
Rysunek 3.4: Dystrybucje dla wybranych posunięć po trzecim rzucie ⚀⚁⚁⚂⚄ z opisem zapisu $(2456\mathrm{TMDY}_0\mathrm{S}, 51, 170)$.

- po dwóch Yahtzee — stan po wyrzuceniu w drugiej rundzie ⚂⚂⚂⚂⚂ i zapisaniu 30 punktów w Szóstkach plus 100 punktów za użycie Jokera, opis zapisu: $(6\mathrm{Y}_{50}, 30, 180)$.

	500	600	700	800	1000
z początku gry	0.010	$1.4 \cdot 10^{-3}$	$1.4 \cdot 10^{-4}$	$1.0 \cdot 10^{-5}$	$1.9 \cdot 10^{-8}$
po jednym Yahtzee	0.057	0.010	$1.2 \cdot 10^{-3}$	$1.0 \cdot 10^{-4}$	$2.6 \cdot 10^{-7}$
po dwóch Yahtzee	0.30	0.070	0.010	$1.0 \cdot 10^{-3}$	$3.5 \cdot 10^{-6}$

Tabela 3.2: Prawdopodobieństwa osiągnięcia niektórych wyników na początku gry, po jednym Yahtzee oraz po dwóch Yahtzee.

Z początku gry trudno jest osiągnąć nawet 500 punktów. Na taki wynik potrzebujemy około 100 rozgrywek. Rekordy w serwisach oscylują w okolicy 800 punktów. Na osiągnięcie takiego wyniku od początku potrzebowalibyśmy około 100 000 prób. Widać, że lepiej jest wystartować od rzucenia z początku Yahtzee. Wtedy, żeby zaatakować 800 punktów, potrzebujemy około 10 000 prób. Po drugim Yahtzee potrzebujemy już tylko około 1 000 podejść. Na rysunku 3.5 przedstawione są dystrybucje dla poszczególnych sytuacji.



Rysunek 3.5: Dystrybucje dla początku gry, po jednym Yahtzee oraz po dwóch Yahtzee.

Rozdział 4

Gra wieloosobowa

W grze wieloosobowej celem jest zdobycie większej liczby punktów od przeciwników, zatem trzeba brać pod uwagę ich aktualną sytuację. Tutaj strategia na maksymalną wartość oczekiwaną nie jest już optymalna. Jest tak dlatego, ponieważ gdy w pewnym momencie tracimy już dosyć dużo do przeciwników, to należy podejmować o wiele większe ryzyko. Odwrotnie, gdy mamy dużą przewagę nad przeciwnikami opłaca się zapisać pewne małe punkty, niż grać na to, aby średnio mieć ich jak najwięcej. W dalszej części w przykładzie 4.4 podamy rozgrywkę, w której granie na maksymalną wartość oczekiwaną w grze dwuosobowej znacznie zmniejsza szanse wygrania z przeciwnikiem.

Problem strategii optymalnej w grze wieloosobowej Yahtzee okazuje się być na tyle trudnym, że w literaturze możemy znaleźć jedynie krótkie wzmianki. Sama definicja strategii optymalnej nie jest oczywista. O ile dla dwóch graczy możemy powiedzieć, że chodzi o maksymalizację prawdopodobieństwa zdobycia większej liczby punktów od przeciwnika (choć i to nie jest dobra definicja ze względu na możliwość remisu), to dla większej liczby graczy, cel gry nie jest jasny. W punkcie 4.1.1 podamy pewien naturalny cel rozgrywki wprowadzając pojęcie punktów meczowych.

W związku z tym, że strategia na maksymalną wartość oczekiwaną wyniku nie jest właściwa w grze wieloosobowej, jest potrzeba skonstruowania strategii, która sprawdziłaby się w takich warunkach znacznie lepiej. Tom Verhoeff [Ver99] wskazuje jedynie, że możliwym podejściem jest aproksymacja. James Glenn [Gle06] pisze, że technika propagacji wstecz, podobnie jak przy wyliczaniu wartości oczekiwanej, może być użyta do stablicowania strategii optymalnej w grze dwuosobowej. Proste szacowanie prowadzi go do nieosiągalnej technicznie liczby stanów: 2^{48}. Stwierdza, że wyliczenie strategii optymalnej nie jest możliwe obecnie znanymi technikami. W swojej ostatniej pracy [Gle07] pisze jedynie o możliwości zastosowania strategii heurystycznych wraz z programowaniem genetycznym.

W tym rozdziale przełamiemy dotychczasowy stan badań wieloosobowej wersji gry w kości Yahtzee. W punkcie 4.1.2 pokażemy implementację strategii optymalnej dla gry dwuosobowej. Następnie w punkcie 4.1.3 dokonamy dokładnej analizy liczby stanów w grze dwuosobowej. Pokażemy dalsze możliwości redukcji przestrzeni stanów i w rezultacie okaże się, że stablicowanie strategii optymalnej w grze dwuosobowej jest w zasięgu dzisiejszych technologii, wbrew temu co dotychczas twierdzono. Mimo wszystko wymaga to sporych zasobów, do których

nie mamy dostępu, a w grze z udziałem trzech lub więcej osób, stablicowanie strategii optymalnej staje się praktycznie niemożliwe. W punktach 4.2 i 4.3 pokażemy innowacyjne strategie heurystyczne oparte na pojęciu dystrybucji wprowadzonym w rozdziale 3 i przeszukiwaniu w głąb. Jedna z otrzymanych strategii doskonale przybliża strategię optymalną. Jej „prawie" optymalność wykażemy za pomocą specjalnie zaprojektowanych eksperymentów w punkcie 4.4.1.

Na koniec zastanowimy się jakie to ma znaczenie w grach z ludźmi. Phil Woodward [Woo03] pisze:

> Jestem przekonany, że mój program pokona każdego człowieka przy odpowiednio długiej rozgrywce, tzn. ze statystycznego punktu wiedzenia, ale nie jestem w stanie podać ile dokładnie partii jest potrzebnych.

Mowa oczywiście o strategii na maksymalizację wartości oczekiwanej w grze jednoosobowej. W grze z ludźmi ta strategia może być wystarczająca, aby mieć przewagę. W punkcie 4.4.4 przeprowadzimy analizę blisko 25 milionów gier ludzi z serwisu Kurnik [Fut07]. Analiza ta pozwoli nam stwierdzić ile tak naprawdę można zyskać w grze z ludźmi, w tak losowej grze jak Yahtzee, stosując strategię „prawie" optymalną. Ponadto okazuje się, że najlepsi gracze w grze dwuosobowej są w stanie wygrywać z komputerem stosującym strategie na wartość oczekiwaną, wbrew temu co twierdził Woodward.

4.1 Strategia optymalna

Może być kilka kryteriów, które określają cel strategii. Na przykład można maksymalizować prawdopodobieństwo wygrania z przeciwnikiem, tzn. prawdopodobieństwo tego, że nasz wynik będzie większy niż każdego z przeciwników. My przyjmiemy inny, bardziej naturalny cel rozgrywki. Będziemy minimalizować wartość oczekiwaną osiągniętego miejsca, czyli będziemy chcieli zająć średnio jak najwyższe miejsce. Dla dwóch graczy oznacza to „prawie" to samo co maksymalizowanie prawdopodobieństwa wygranej. Dlaczego tylko „prawie" to samo, a nie dokładnie to samo? Otóż dlatego, że są też możliwe remisy. Remis jest wtedy, gdy gracze kończą grę z tym samym wynikiem.

4.1.1 Punkty meczowe

Jak określać miejsce, gdy jest więcej graczy? Na przykład, gdy dwóch graczy będzie miało taki sam wynik, to mówi się, że zajęli oni miejsca drugie i trzecie ex aequo. Z punktu widzenia optymalizacji średniego miejsca lepiej by było powiedzieć, że ci gracze zajęli miejsce dwa i pół. Aby usystematyzować pojęcie miejsca wprowadzimy pojęcie punktów meczowych.

Definicja 4.1. Załóżmy, że mamy daną rozgrywkę n graczy z wynikami końcowymi $w_1, \ldots, w_n$. Liczba *punktów meczowych* za tą rozgrywkę przydzielana jest poszczególnym graczom w następujący sposób. Dla każdej pary graczy, gracz który ma większy wynik dostaje jeden punkt meczowy, a jeśli obaj gracze mają ten sam wynik, to obaj dostają po pół punktu meczowego. Innymi słowy liczba

punktów meczowych zdobytych przez gracza i, $1 \leqslant i \leqslant n$, wynosi:

$$\sum_{j \neq i} \begin{cases} 1 & \text{dla } w_i > w_j \\ \frac{1}{2} & \text{dla } w_i = w_j \\ 0 & \text{dla } w_i < w_j \end{cases}$$

W przypadku n graczy 0 punktów meczowych oznacza miejsce ostatnie, 1 punkt oznacza miejsce przedostatnie, a $n-1$ punktów oznacza miejsce pierwsze. Oczywiście są też możliwe wartości ułamkowe. Zamiast mówić o miejscach, w dalszej części będziemy mówić o punktach meczowych. Strategia optymalna, którą się zajmiemy, polega na maksymalizowaniu wartości oczekiwanej punktów meczowych.

Alternatywne punktacje. Możliwe są też dowolne inne przydziały punktów za zajęte miejsca. W ogólności, jeżeli w grze uczestniczy n graczy, to za i-te miejsce możemy przydzielać p_i punktów, przy czym ciąg $p_1, \ldots, p_n$ powinien być ciągiem nierosnącym. W przypadku miejsc ex aequo możemy definiować dalsze kryteria przydzielania liczby punktów. Na przykład może to być średnia arytmetyczna zdobytych punktów za poszczególne miejsca.

Przykład 4.2. W grze, w której uczestniczą cztery osoby, zajęcie pierwszego, czy drugiego miejsca może być bardziej premiowane. Ciąg p_i jaki można przyjąć to: $6, 3, 1, 0$, tzn. za pierwsze miejsce dostaje się 6 punktów, za drugie 3, za trzecie 1, a za czwarte 0. W przypadku remisów, jeśli bierzemy średnią arytmetyczną, mamy następujące możliwości. Za pierwsze i drugie ex aequo gracze dostaną $\frac{1}{2}(6+3) = 4\frac{1}{2}$ punktu, a na przykład za miejsca drugie, trzecie i czwarte ex aequo dostaną $\frac{1}{3}(3+1+0) = 1\frac{1}{3}$ punktu.

4.1.2 Gra dwuosobowa

Gra w kości jest grą losową z pełną informacją. W przypadku dwóch osób jest grą o sumie zerowej. Wtedy maksymalizowanie naszych punktów meczowych polega jednocześnie na minimalizowaniu punktów meczowych przeciwnika. Zatem, gdy są dwie osoby, strategia optymalna istnieje i można to w prosty sposób wykazać konstruując ją w sposób indukcyjny „od tyłu". Jest to indukcja po sumie wolnych kategorii u obu graczy.

Wynik gry w stanie końcowym, gdy nie ma kategorii wolnych u żadnego gracza, określa definicja 4.1. W założeniu indukcyjnym przyjmujemy, że skonstruowaliśmy strategię optymalną (dla obu graczy) dla stanów o mniejszej liczbie kategorii wolnych lub przy danym opisie zapisu dla późniejszych stanów rundy (np. o większej liczbie wykonanych rzutów), a co za tym idzie potrafimy także określić wartość oczekiwaną punktów meczowych przy optymalnej rozgrywce obu graczy. Przy zdarzeniu losowym nie podejmujemy żadnej decyzji, a wartością stanu jest średnia ważona stanów po tym zdarzeniu. W stanie, w którym gracz podejmuje decyzję, wybieramy posunięcie, które maksymalizuje naszą wartość oczekiwaną punktów meczowych, co jednocześnie minimalizuje wartość oczekiwaną punków meczowych przeciwnika.

Przekształćmy powyższe rozumowanie w algorytm. Najpierw zastanówmy się jakie są możliwe stany w grze dwuosobowej i co należy w nich pamiętać. Ponumerujmy graczy. Pierwszy niech będzie graczem, który właśnie wykonuje

swoje posunięcie (czyli my), a drugi niech będzie graczem, który czeka na swoją kolej (przeciwnik). W każdym z możliwych stanów będziemy liczyć maksymalną oczekiwaną liczbę punktów meczowych, jaką może zdobyć pierwszy gracz. Liczba ta jest liczbą rzeczywistą z przedziału $[0, 1]$. Jeśli liczba ta wynosi x, to z punktu widzenia gracza drugiego, jego maksymalna oczekiwana liczba punktów meczowych wynosi $1 - x$. Wynika to z tego, że punkty meczowe sumują się zawsze do jedynki.

Załóżmy, że stany zapisu pierwszego i drugiego gracza wynoszą odpowiednio (K_1, c_1, p_1) i (K_2, c_2, p_2). Taki stan zapisu obu graczy oznaczamy parą $\big((K_1, c_1, p_1), (K_2, c_2, p_2)\big)$. Przyjrzyjmy się jakie są ograniczenia dla tego stanu. Liczba zapisanych kategorii w K_1 i K_2 zależy od siebie, gdyż gracze cyklicznie wykonują swoje posunięcia. Jeśli pierwszy gracz inicjuje rundę, to $|K_1| = |K_2|$, a jeśli ją kończy, to $|K_1| + 1 = |K_2|$. Ponadto częściówki c_1 i c_2 muszą być osiągalne odpowiednio dla opisów kategorii K_1 i K_2.

Chcemy policzyć maksymalną oczekiwaną liczbę punktów meczowych dla pierwszego gracza w stanie $\big((K_1, c_1, p_1), (K_2, c_2, p_2)\big)$. Zgodnie z faktem 3.10, aktualne wyniki graczy p_1 i p_2 nie wpływają na dalsze możliwości punktowania. Wartości p_1 i p_2 potrzebne są do określenia wyników poszczególnych graczy na końcu rozgrywki, a co za tym idzie punktów meczowych. Do tego, tak naprawdę jest nam potrzebna jedynie różnica wyników, gdyż maksymalna oczekiwana liczba punktów meczowych w danym stanie jest taka sama jak w stanie $\big((K_1, c_1, 0), (K_2, c_2, p_2 - p_1)\big)$. Ustalmy możliwości zapisowe (K_1, c_1) i (K_2, c_2). Taki stan możliwości zapisowych oznaczamy przez $\big((K_1, c_1), (K_2, c_2)\big)$. Szukana maksymalna oczekiwana liczba punktów meczowych będzie tym większa im mniejsza jest różnica $p_2 - p_1$ i tym mniejsza im większa jest ta różnica. Zatem funkcja, która dla różnicy $p_2 - p_1$ zwraca maksymalną oczekiwaną liczbę punktów meczowych jest dystrybucją (definicja 3.12). Tak więc możemy użyć dystrybucji do pamiętania wartości stanów, co w praktyce pozwala posłużyć się algorytmami z punktu 3.3.

Użycie dystrybucji pozwala na znaczne ograniczenie liczby danych, które należy pamiętać, gdyż na przykład w stanach, w których jest dużo kategorii zapisanych, aktualne wyniki graczy p_1 i p_2 mogą znacznie się różnić, a ich różnica co do wartości bezwzględnej może przekraczać nawet tysiąc. Jednakże jeśli różnica $p_2 - p_1$ jest odpowiednio duża, bądź odpowiednia mała, to zwycięzca gry jest już znany, a wtedy wartość dystrybucji wynosi 1 lub 0 i w ogóle nie będzie ona pamiętana.

Tablicowanie ponownie robimy używając programowania dynamicznego na dwóch poziomach. Będziemy budować tablicę[1] *EMP* (wartości oczekiwanych punktów meczowych) indeksowaną czwórkami (K_1, c_1, K_2, c_2) po $K_1, K_2 \in \mathcal{K}$ i $c_1, c_2 \in \{0, 1, \ldots, 63\}$ taką, że $EMP[K_1, c_1, K_2, c_2]$ zawiera dystrybucję, a wartość $EMP[K_1, c_1, K_2, c_2](p_2 - p_1)$ oznacza maksymalną oczekiwaną liczbę punktów meczowych osiągalną w stanie $\big((K_1, c_1, 0), (K_2, c_2, p_2 - p_1)\big)$ przez pierwszego gracza.

Poziom zapisu

Wyliczanie oczekiwanych punktów meczowych w grze dwuosobowej na poziomie zapisu przedstawione jest w algorytmie 4.1. Stany końcowe w wierszu 5

[1] W praktyce *EMP* jest mapą, gdyż wiele indeksów w ogóle się nie pojawia.

EMP-Poziom-Zapisu
wynik: dystrybucje *EMP* oczekiwanych punktów meczowych dla wszystkich możliwości zapisowych obu graczy

1: **foreach**
2: $c_1, c_2 \in \{0, 1, \ldots, 63\}$,
3: $K_1, K_2 \in \big\{\{123456\text{TKFMDY}_0\text{S}\}, \{123456\text{TKFMDY}_{50}\text{S}\}\big\}$
4: **do**
5: $EMP[K_1, c_1, K_2, c_2] \leftarrow \frac{1}{2}(\theta_{-1} + \theta_0)$
6: **for** $z \leftarrow 25$ **downto** 0 **do**
7: **foreach** $K_1, K_2 \in \mathcal{K}, |K_1| + |K_2| = z, 0 \leqslant |K_2| - |K_1| \leqslant 1$ **do**
8: EMP-Poziom-Zapisu-Częściówki(K_1, K_2)

EMP-Poziom-Zapisu-Częściówki(K_1, K_2)

1: $S_1 \leftarrow K_1 \cap K_g$
2: $S_2 \leftarrow K_2 \cap K_g$
3: $C_1 \leftarrow$ Częściówki-Osiągalne(S_1)
4: $C_2 \leftarrow$ Częściówki-Osiągalne(S_2)
5: **for** $i \leftarrow 1$ **to** $|C_1|$ **do**
6: **for** $j \leftarrow 1$ **to** $|C_2|$ **do**
7: **if** $NRC_{S_1}[c_1^i] < c_1^i$ **then**
8: $EMP[K_1, c_1^i, K_2, c_2^j] \leftarrow EMP[K_1, NRC_{S_1}[c_1^i], K_2, c_2^j]$
9: **elseif** $NRC_{S_2}[c_2^j] < c_2^j$ **then**
10: $EMP[K_1, c_1^i, K_2, c_2^j] \leftarrow EMP[K_1, c_1^i, K_2, NRC_{S_2}[c_2^j]]$
11: **else**
12: $EMP[K_1, c_1^i, K_2, c_2^j] \leftarrow$ EMP-Poziom-Rundy(K_1, c_1^i, K_2, c_2^j)

Algorytm 4.1: Wyliczanie oczekiwanej liczby punktów meczowych w strategii optymalnej dla dwóch graczy na poziomie zapisu.

procedury EMP-Poziom-Zapisu inicjowane są dystrybucją $\frac{1}{2}(\theta_{-1} + \theta_0)$, która wyraża się wzorem:

$$\frac{1}{2}(\theta_{-1} + \theta_0)(x) = \begin{cases} 0 & \text{dla } x > 0, \\ \frac{1}{2} & \text{dla } x = 0, \\ 1 & \text{dla } x < 0, \end{cases}$$

Innymi słowy jest to 0, gdy $p_1 < p_2$, $\frac{1}{2}$, gdy $p_1 = p_2$ i 1, gdy $p_1 > p_2$, czyli zgodnie z definicją punktów meczowych. Zwróćmy również uwagę, że optymalizacja częściówek jest bardziej skomplikowana niż w wersji jednoosobowej (procedura EMP-Poziom-Zapisu-Częściówki).

Poziom rundy

Przy wyliczaniu wartości stanu dla ustalonych możliwościach zapisowych graczy, należy przejść wszystkie stany rundy pierwszego gracza. Podobnie jak przy maksymalizacji wartości oczekiwanej i przy maksymalizacji prawdopodobieństwa określonego wyniku (punkty 3.2 i 3.3) możemy przyjąć analogiczne oznaczenia na dystrybucje poszczególnych stanów rundy:

- *wynik* — szukana dystrybucja dla możliwości zapisowych graczy (K_1, c_1) i (K_2, c_2), czyli wartość $wynik(p_2 - p_1)$ oznacza maksymalną wartość osze-

kiwaną punktów meczowych gracza pierwszego, gdzie p_1 i p_2 oznaczają aktualny wynik każdego z graczy,

- $rzut_i[R]$, $i = 1, 2, 3$, $R \in \mathcal{R}$ — dystrybucja po wykonaniu i-tego rzutu R,
- $zatrzymanie_i[Z]$, $i = 1, 2$, $Z \in \mathcal{Z}$ — dystrybucja po i-tym zatrzymaniu Z,
- $zapis[R, k]$, $R \in \mathcal{R}$, $k \in$ Kategorie-Wolne(K_1) — dystrybucja po zapisaniu rzutu R w kategorii k.

Aby je wyznaczyć możemy użyć procedury Propaguj-Wartości-Stanów-Rundy i jedynie należy zainicjować tablicę *zapis*. Do tego można użyć gotowych dystrybucji dla stanów, w których pierwszy gracz ma zapisaną jedną kategorię więcej. Takie dystrybucje mamy już policzone, ale mamy je z punktu widzenia gracza drugiego. Jeżeli dystrybucja P_2 określa oczekiwaną liczbę punktów meczowych z punktu widzenia gracza drugiego dla różnicy wyników $p_1 - p_2$, to chcielibyśmy z tego wyciągnąć dystrybucję P_1 dla gracza pierwszego, która określa oczekiwaną liczbę punktów meczowych dla różnicy wyników $p_2 - p_1$. Przyjmując $x = p_2 - p_1$ możemy $P_1(p_2 - p_1)$ wyrazić wzorem:

$$P_1(x) = 1 - P_2(-x), \tag{4.1}$$

skąd wywodzi się definicja na dystrybucję odwróconą.

Definicja 4.3. Dystrybucję P_1 nazywamy *dystrybucją odwróconą* dystrybucji P_2, jeżeli dla każdego $x \in \mathbb{Z}$ zachodzi (4.1) i oznaczamy

$$P_1 = \text{Odwróć}(P_2).$$

W praktyce procedurę Odwróć implementujemy w ten sposób, że wektor wartości $[P(x_0), \ldots, P(x_1)]$ i punkt początkowy x_0 dystrybucji P zastępujemy wektorem $[1 - P(x_1), \ldots, 1 - P(x_0)]$ i punktem początkowym $-x_1$.

Mając powyższą operację możemy już podać kod procedury EMP-Poziom-Rundy liczącą wartości dla poszczególnych stanów rundy (algorytm 4.2).

EMP-Poziom-Rundy(K_1, c_1, K_2, c_2)
argumenty: opis możliwości zapisowych graczy (K_1, c_1) i (K_2, c_2)
wynik: dystrybucja maksymalnych wartości oczekiwanych punktów meczowych

1: **foreach** $R \in \mathcal{R} \wedge k \in$ Kategorie-Wolne(K_1) **do**
2: $\quad (K_1', c_1', p') \leftarrow \text{Zapis}\big((K_1, c_1, 0), R, k\big)$
3: $\quad zapis[R, k] \leftarrow \text{Odwróć}(EMP[K_2, c_2, K_1', c_1']) + p'$
4: Propaguj-Wartości-Stanów-Rundy(K_1)
5: **return** *wynik*

Algorytm 4.2: Wyliczanie oczekiwanej liczby punktów meczowych w strategii optymalnej dla dwóch graczy na poziomie rundy.

4.1.3 Analiza złożoności gry dwuosobowej

W poprzednim punkcie pokazaliśmy algorytm na wyliczenie strategii optymalnej dla wszystkich stanów w grze dwuosobowej. Zastanówmy się, czy w praktyce jest to wykonalne, tzn. jaka jest potrzebna przestrzeń dyskowa oraz ile czasu może to zająć.

Na dysku musimy zapamiętać jedynie wartości oczekiwane punktów meczowych dla wszystkich par możliwości zapisowych graczy (K_1, c_1) i (K_2, c_2) oraz dla wszystkich możliwych różnic wyników $p_2 - p_1$. Skupimy się teraz na wyznaczeniu ile z tych wartości będzie potrzeba zapisać. Wpierw ograniczymy zbiór par możliwości zapisowych. Nie wszystkie stany mogą wystąpić. Można przyjąć następujące ograniczenia.

1. Pierwszy gracz musi mieć zapisane tyle samo kategorii co drugi lub jedną mniej: $0 \leqslant |K_2| - |K_1| \leqslant 1$.
2. Częściówki c_1 i c_2 powinny być osiągalne odpowiednio dla opisu kategorii K_1 i K_2.
3. Możemy utożsamić częściówki nierozróżnialne, tzn. brać pod uwagę tylko takie częściówki c, dla których $NRC_S[c] = c$ dla odpowiedniego S.

Tabela 4.1 przedstawia zestawienie liczby stanów dla poszczególnych $|K_1|$ i $|K_2|$. Dla ustalonych mocy $|K_1|$ i $|K_2|$, liczymy liczbę możliwości zapisowych (K_1, c_1)

$\lvert K_1\rvert$	$\lvert K_2\rvert$	
13	13	4
12	13	170
12	12	7 225
11	12	108 970
11	11	1 643 524
10	11	10 179 080
10	10	63 043 600
9	10	212 553 800
9	9	716 632 900

$\lvert K_1\rvert$	$\lvert K_2\rvert$	
8	9	1 497 165 790
8	8	3 127 829 329
7	8	4 308 280 518
7	7	5 934 237 156
6	7	5 530 733 064
6	6	5 154 665 616
5	6	3 252 071 616
5	5	2 051 727 616
4	5	854 962 000

$\lvert K_1\rvert$	$\lvert K_2\rvert$	
4	4	356 265 625
3	4	92 487 500
3	3	24 010 000
2	3	3 449 600
2	2	495 616
1	2	30 976
1	1	1 936
0	1	44
0	0	1

Tabela 4.1: Zestawienie liczby wszystkich par możliwości zapisowych.

oraz liczbę możliwości zapisowych (K_2, c_2) i przemnażamy te dwie liczby. Sumaryczna liczba stanów wynosi 33 192 583 276. Jest to już dosyć spora liczba jak na dzisiejsze możliwości komputerów osobistych, a my dla każdego z tych stanów powinniśmy pamiętać całą dystrybucję, która jest nierosnącym ciągiem liczb rzeczywistych z przedziału $(0, 1)$.

Zastanówmy się jak szacować rozmiar dystrybucji P, którą należy zapamiętać dla danego stanu możliwości zapisowych graczy $\big((K_1, c_1), (K_2, c_2)\big)$, gdzie c_1 i c_2 są osiągalnymi częściówkami odpowiednio dla K_1 i K_2. Interesują nas tylko takie argumenty $x \in \mathbb{Z}$, że $1 > P(x) > 0$. Pamiętajmy, że $P(x)$ reprezentuje wartość oczekiwaną punktów meczowych gracza pierwszego, dla $x = p_2 - p_1$, gdzie p_1 i p_2 są aktualnymi wynikami graczy, czyli x mówi, ile aktualnie punktów brakuje graczowi pierwszemu do gracza drugiego. Oznaczmy przez $\min o_1$ i $\max o_1$ najmniejszą i największą liczbę punktów, jaką może zdobyć gracz pierwszy ze stanu (K_1, c_1). Analogicznie oznaczmy przez $\min o_2$ i $\max o_2$ najmniejszą i największą liczbę punktów, jaką może zdobyć gracz drugi ze stanu (K_2, c_2). Jeśli x jest tak małe, że gracz drugi nie jest w stanie zniwelować różnicy punktów do gracza pierwszego przy dowolnej rozgrywce, to jasne jest, że $P(x) = 1$. Ma to miejsce wtedy, gdy $p_1 + \min o_1 > p_2 + \max o_2$, czyli gdy $x < \min o_1 - \max o_2$. Zatem dolne ograniczenie na x wynosi $\min o_1 - \max o_2$. Analogicznie wyprowadzamy, że $P(x) = 0$ dla $x > \max o_1 - \min o_2$, skąd górne ograniczenie na x to $\max o_1 - \min o_2$.

Dystrybucja P dla stanu możliwości zapisowych $\big((K_1, c_1), (K_2, c_2)\big)$ jest taka sama dla wszystkich stanów $\big((K_1, c_1'), (K_2, c_2')\big)$, dla których $NRC_{S_1}[c_1'] = c_1$

i $NRC_{S_2}[c'_2] = c_2$, gdzie $S_1 = K_1 \cap K_g$ i $S_2 = K_2 \cap K_g$. Zatem dla tego zbioru stanów wystarczy pamiętać jedną dystrybucję.

Na koniec zauważmy, że nie potrzebujemy pamiętać całej dystrybucji, tzn. wszystkich wartości $P(x)$, dla których $1 > P(x) > 0$, ponieważ nie wszystkie wartości x mogą być osiągalne w stanie $\big((K_1, c_1), (K_2, c_2)\big)$. Oznaczmy przez $\min p_1$ i $\max p_1$ najmniejszą i największą liczbę punktów, jaką gracz pierwszy może posiadać w stanie (K_1, c_1). Analogicznie oznaczmy przez $\min p_2$ i $\max p_2$ najmniejszą i największą liczbę punktów, jaką gracz drugi może posiadać w stanie (K_2, c_2). Wtedy najmniejsze osiągalne x to $\min p_2 - \max p_1$, a największe x to $\max p_2 - \min p_1$, więc można ograniczyć dystrybucję tak, aby argumenty wpadały do przedziału $[\min p_2 - \max p_1, \max p_2 - \min p_1]$. Ponieważ jedna dystrybucja może być pamiętana dla wielu stanów, więc aby ograniczyć zbiór argumentów, który trzeba pamiętać, należy wziąć pod uwagę wszystkie takie przedziały ograniczające dla tych stanów i wziąć najmniejszy przedział, który zawiera je wszystkie.

Tabela 4.2 przedstawia sumarycznie ile rzeczywiście trzeba pamiętać wartości dystrybucji, po uwzględnieniu wszystkich wymienionych ograniczeń. Tabelę

$\lvert K_1\rvert$	$\lvert K_2\rvert$	
13	13	4
12	13	16 108
12	12	1 325 955
11	12	25 539 171
11	11	453 792 562
10	11	3 217 757 027
10	10	22 211 567 786
9	10	82 783 641 016
9	9	300 915 379 820
8	9	671 255 866 032
8	8	1 457 509 326 327
7	8	1 999 968 096 032
7	7	2 552 435 835 158
6	7	2 146 557 157 931

$\lvert K_1\rvert$	$\lvert K_2\rvert$	
6	6	1 743 391 907 424
5	6	955 286 057 050
5	5	499 277 719 048
4	5	170 624 278 183
4	4	53 794 722 569
3	4	10 362 427 034
3	3	1 725 140 368
2	3	166 637 548
2	2	12 263 680
1	2	525 448
1	1	17 776
0	1	224
0	0	1

Tabela 4.2: Zestawienie sumaryczne liczby wartości dystrybucji, które trzeba pamiętać w grze dwuosobowej.

tą tworzy się w ten sposób, że dla każdej pary K_1 i K_2, przechodzimy wszystkie częściówki osiągalne c_1 i c_2 rejestrując ile dystrybucji trzeba pamiętać, jakie są rozmiary każdej dystrybucji oraz jakie są wszystkie przedziały ograniczające daną dystrybucję. Wcześniej przygotowujemy sobie wszystkie wartości $\min o, \max o, \min p, \max p$ dla wszystkich możliwości zapisowych (K, c) w sposób dynamiczny. Wartości $\min o, \max o$ obliczamy „od tyłu" podobnie jak liczyliśmy wartości oczekiwane, ale już bez potrzeby przeglądania stanów kolejki, a $\min p, \max p$ tablicujemy „od przodu", od najmniejszej do największej liczby zapisanych kategorii.

Łącznie liczba wszystkich wartości dystrybucji, które trzeba zapamiętać wynosi 12 671 976 997 282. Zakładając, że jedna wartość zajmuje 8 bajtów, czyli tyle co liczba zmiennopozycyjna wysokiej precyzji, łączna wielkość potrzebnej przestrzeni dyskowej wynosi 92.2 TB (terabajty). Taki rozmiar jest po za zasięgiem komputerów osobistych w najbliższym czasie, jednakże jest na pewno w zasięgu serwerów. Zakładając, że jesteśmy w stanie wyliczyć 10 000 wartości na sekundę (taką efektywność osiągamy na komputerze Pentium 4, 2.8 GHz), łączny czas wynosi 40 lat. Obliczenia łatwo się zrównoleglają. Na klastrze 60 czterordzenio-

wych komputerów klasy PC obliczenia zajęłyby tylko 2 miesiące. My stablicowaliśmy tylko końcowe stany dla $|K_1| + |K_2| \geqslant 10 + 11 = 21$, czyli wszystkie stany o głębokości co najwyżej 5, tzn. takie, w których brakuje co najwyżej pięciu zapisów do końca gry, łącznie przez obu graczy. Potrzeba na to 27.5 GB. W zasięgu są też tablice o głębokości 6 i potrzeba na nie 193 GB, ale niestety posiadany komputer nie miał takiej pojemności dysku twardego.

4.1.4 Gra co najmniej trzyosobowa

W przypadku większej liczby osób wyliczenie strategii optymalnej staje się dużo trudniejsze. Po pierwsze znacznie rośnie liczba stanów, co w praktyce nie pozwala na tablicowanie końcówek. Po drugie trzeba dodatkowo coś zakładać o strategii przeciwników, gdyż mogą ze sobą współpracować. Na przykład, gdy jest trzech graczy i mamy dwóch przeciwników, to mogą oni kooperować ze sobą i grać na minimalizację naszych punktów meczowych. Z drugiej strony każdy z nich może grać na siebie i maksymalizować swoje punkty meczowe. Zależnie od tego, czy przeciwnicy kooperują, czy też nie, możemy otrzymać różne strategie optymalne. Z powyższych powodów nie będziemy zajmować się analizą złożoności w tym przypadku. Tworzenie końcówek w przypadku dwuosobowym już jest kłopotliwe ze względu na ich wielkość, więc dla większej liczby osób staje się już to praktycznie niewykonalne.

4.2 Strategie heurystyczne

W poprzednich punktach pokazaliśmy, że podanie dokładnej wartości dla każdego stanu dla strategii optymalnej jest na razie praktycznie nieosiągalne w grze dwuosobowej, a tym bardziej w grze co najmniej trzyosobowej. Dokładna ocena stanu nie jest na ogół możliwa, więc trzeba szukać innych sposobów.

Załóżmy, że dla każdego stanu potrafimy ocenić jego jakość wartością liczbową. Na tej podstawie potrafimy już wytworzyć strategię. Otóż, gdy dochodzi do naszej decyzji (wybór kości do rzutu albo wybór kategorii do zapisu), to dla każdego posunięcia bierzemy ocenę stanu powstałego w wyniku wykonania go. Najlepszym posunięciem będzie takie, które daję największą ocenę.

Pozostaje wytworzyć taką funkcję oceniającą. W tym celu bardzo przydatne są dystrybucje. W danej sytuacji możemy bardzo szybko dostać dla każdego gracza dystrybucję maksymalnych prawdopodobieństw osiągnięcia poszczególnych wyników. Okazuję się, że na tej podstawie można w prosty sposób budować skuteczne funkcje oceniające.

4.2.1 Dynamiczna strategia na ustalony wynik

Znajomość dystrybucji pozwala nam dla ustalonego wyniku określić maksymalne prawdopodobieństwo jego osiągnięcia. W grze wieloosobowej chodzi o to, żeby zdobyć więcej punktów niż przeciwnicy, więc chcemy grać na taki wynik, który jest dla nas w miarę łatwy do osiągnięcia, a dla przeciwników jak najtrudniejszy.

Niech P będzie naszą dystrybucją, a $P_1, \ldots, P_n$ dystrybucjami przeciwników. Dla ustalonego x, maksymalne prawdopodobieństwo tego, że osiągniemy wynik x wynosi $P(x)$. Rozważmy i-tego przeciwnika. Jeżeli zdobędzie on mniej niż x punktów, to mamy gwarancję, że go wyprzedziliśmy, a jeśli zdobędzie co

najmniej x punktów, to zakładamy, że go nie wyprzedziliśmy. Wtedy najgorszą dla nas strategią jest granie tego przeciwnika na wynik x. Prawdopodobieństwo, że nie uda mu się go osiągnąć wynosi $1 - P_i(x)$. Zatem przy danych założeniach prawdopodobieństwo tego, że wyprzedzimy i-tego przeciwnika wynosi $P(x)\big(1-P_i(x)\big)$. Oczekiwana liczba przeciwników, których wyprzedzimy wynosi

$$P(x)\sum_{i=1}^{n}\big(1 - P_i(x)\big). \tag{4.2}$$

Naszym celem jest dobranie x tak, aby wartość ta była największa. Zatem naszą oceną jest maksimum po wszystkich x z (4.2). Zauważmy, że najlepszy x może być inny dla różnych ocenianych stanów i w czasie gry może on się zmieniać, dlatego strategię korzystającą z tej funkcji oceniającej nazywamy *dynamiczną strategią na ustalony wynik*.

Funkcja OcenaDynW, przedstawiona w algorytmie 4.3, realizuje powyższą ocenę. Zauważmy, że wartości x, dla których liczymy maksimum to są wszystkie

OcenaDynW$(P, P_1, \ldots, P_n)$
argumenty: nasza dystrybucja P i dystrybucje przeciwników $P_1, \ldots, P_n$
wynik: ocena sytuacji
1: $P \equiv \big(x_0, [P(x_0), \ldots, P(x_1)]\big)$
2: **return** $\max\limits_{x_0-1 \leqslant x \leqslant x_1} P(x)\sum\limits_{i=1}^{n}\big(1 - P_i(x)\big)$

Algorytm 4.3: Funkcja oceniająca, która dynamicznie dobiera grę na ustalony wynik.

te wartości, dla których $1 > P(x) > 0$, czyli $x_0 \leqslant x \leqslant x_1$. Ponadto dla $x > x_1$ ocena zawsze wynosi 0, a dla $x < x_0$ może się zdarzyć, że ocena będzie jednak największa i wystarczy wziąć do sprawdzenia tylko największe takie x, czyli $x = x_0 - 1$.

4.2.2 Przybliżanie wartości oczekiwanej punktów meczowych przez dystrybucje

Innym sposobem budowania funkcji oceniającej na podstawie dystrybucji jest potraktowanie ich jak rzeczywistych rozkładów wyników każdego z graczy. Oczywiście, jak zostało to pokazane wcześniej, dystrybucje maksymalnych prawdopodobieństw osiągnięcia określonych wyników nie odzwierciedlają żadnego rozkładu dla żadnej strategii. Wynika to z tego, że wartość oczekiwana takiego rozkładu jest większa od tej, która otrzymujemy przy stosowaniu strategii na maksymalną wartość oczekiwaną. Mimo to dystrybucja w jakiś sposób obrazuje możliwości punktowania każdego z graczy. Średnia jest zawyżona, ale efekt ten występuje również dla dystrybucji przeciwników. Zatem porównanie do siebie naszej dystrybucji do dystrybucji przeciwników powinno być dosyć wymierne.

Pozostaje wyliczyć wartość oczekiwaną punktów meczowych na podstawie rozkładów wyników każdego z graczy. Rozkłady mamy dane w formie dystrybucji. Niech P będzie naszą dystrybucją, a $P_1, \ldots, P_n$ dystrybucjami przeciwników. Na tej podstawie prawdopodobieństwo tego, że uzyskamy wynik dokładnie

x wynosi $P(x) - P(x+1)$. Gracz i będzie mieć wynik mniejszy niż x z prawdopodobieństwem $1 - P_i(x)$, równy x z prawdopodobieństwem $P_i(x) - P_i(x+1)$ i większy od x z prawdopodobieństwem $P_i(x+1)$. Zatem oczekiwana liczba punktów meczowych kosztem gracza i, przy założeniu, że zdobędziemy dokładnie x punktów, wynosi:

$$1\cdot\big(1-P_i(x)\big)+\frac{1}{2}\cdot\big(P_i(x)-P_i(x+1)\big)+0\cdot P_i(x+1) = 1-\frac{1}{2}\big(P_i(x)+P_i(x+1)\big). \quad (4.3)$$

Po zsumowaniu (4.3) po wszystkich i otrzymujemy oczekiwaną liczbę punktów meczowych w przypadku, gdybyśmy zdobyli dokładnie x punktów. Po uwzględnieniu wszystkich możliwych wartości x wraz z prawdopodobieństwami, otrzymujemy funkcję OcenaEMP (algorytm 4.4) dającą przybliżoną oczekiwaną

OcenaEMP$(P, P_1, \ldots, P_n)$
argumenty: nasza dystrybucja P i dystrybucje przeciwników $P_1, \ldots, P_n$
wynik: ocena sytuacji
1: $P \equiv \big(x_0, [P(x_0), \ldots, P(x_1)]\big)$
2: **return** $\displaystyle\sum_{x=x_0-1}^{x_1+1} \big(P(x) - P(x+1)\big) \sum_{i=1}^{n} \left(1 - \frac{1}{2}\big(P_i(x) - P_i(x+1)\big)\right)$

Algorytm 4.4: Funkcja oceniająca zwracająca przybliżoną oczekiwaną liczbę punktów meczowych.

liczbę punktów meczowych. Wystarczy, że x przebiega wartości $x_0-1, \ldots, x_1+1$, gdyż tylko dla nich różnica $P(x) - P(x+1)$ jest niezerowa.

Funkcja oceniająca OcenaEMP w praktyce działa zaskakująco dobrze, co jest pokazane w punkcie 4.4.1. W połączeniu z przeszukiwaniem w głąb, opisanym w punkcie 4.3, można otrzymać strategię bardzo bliską optymalnej.

Alternatywne punktacje. Przy innych sposobach przydzielania punktów za kolejne miejsca i miejsca ex aequo (niż punkty meczowe), funkcja oceny może mieć dużo bardziej skomplikowany wzór. Wiąże się to z tym, że dla ustalonej wartości x, trzeba osobno rozważać każdą kombinację pozycji każdego przeciwnika względem nas. Każdy gracz, od 1 do n, może osiągnąć wartość mniejszą od x, równą x, czy też większą od x. Zatem jest 3^n kombinacji, z której każda może mieć inną wagę, tzn. każda może oznaczać inną liczbę przyznanych dla nas punktów. Jednakże taki wzór zawsze można wyprowadzić i przedstawiona heurystyka jest również użyteczna w alternatywnych sposobach punktacji.

4.3 Przeszukiwanie w głąb

Załóżmy, że mamy pewną funkcję oceniającą Ocena, która na podstawie naszej dystrybucji oraz dystrybucji przeciwników daje ocenę sytuacji. Znamy dwie takie funkcje: OcenaDynW i OcenaEMP. W algorytmie 4.5 znajduje się prosta implementacja strategii z użyciem funkcji oceny. W wierszu 3 inicjujemy tablicę *zapis* odpowiednią dystrybucją przesuniętą dla gracza pierwszego, korzystając z tablicy P dystrybucji maksymalnych prawdopodobieństw osiągnięcia określonego wyniku. Następnie propagujemy dystrybucje dla wszystkich stanów rundy.

Strategia-Z-Oceną$(K_1, c_1, p_1, K_2, c_2, p_2, \mathit{rzuty}, R_1)$
argumenty: opisy zapisu (K_1, c_1, p_1) i (K_2, c_2, p_2) gracza pierwszego i drugiego, liczba wykonanych rzutów *rzuty* oraz aktualny rzut R_1 gracza pierwszego
wynik: posunięcie — zatrzymanie kości $Z \subseteq R_1$, bądź zapisywana kategoria $k \in$ Kategorie-Wolne(K_1)

```
1:  foreach R ∈ ℛ ∧ k ∈ Kategorie-Wolne(K_1) do
2:      (K'_1, c'_1, p'_1) ← Zapis((K_1, c_1, p_1), R, k)
3:      zapis[R, k] ← P[K'_1, c'_1] + p'_1
4:  Propaguj-Wartości-Stanów-Rundy(K_1)
5:  P_2 ← P[K_2, c_2] + p_2
6:  if rzuty < 3 then
7:      return Z ⊆ R_1 maximizing Ocena(zatrzymanie_rzuty[Z], P_2)
8:  else
9:      return
10:         k ∈ Kategorie-Wolne(K_1) maximizing Ocena(zapis[R_1, k], P_2)
```

Algorytm 4.5: Prosta strategia w grze dwuosobowej z użyciem funkcji oceny.

Dystrybucję dla gracza drugiego tworzymy w wierszu 5. Na końcu dystrybucje obu graczy są używane do oceniania wartości poszczególnych posunięć.

Propagowanie dystrybucji funkcją Propaguj-Wartości-Stanów-Rundy jest znacznie kosztowniejsze niż propagowanie pojedynczych wartości. W praktyce nie ma to znaczenia, gdyż i tak strategia z propagowaniem dystrybucji jest wystarczająco szybka. Jednakże możemy zwiększyć siłę strategii, jeśli ocenę sytuacji przesuniemy w przyszłość na parę posunięć do przodu. Najprostszą optymalizacją jest użycie funkcji oceny od razu przy inicjacji tablicy *zapis*, a dopiero potem rozpropagowanie wartości rundy. Możemy posunąć się dalej i do oceny stanu zapisu (tj. w sytuacji gdy pierwszy gracz dokonał zapisu) możemy wykonać do przodu analizę rundy z punktu widzenia gracza drugiego. Wtedy funkcję Ocena stosujemy dopiero po wykonaniu zapisu przez gracza drugiego. Oczywiście możemy to rozumowanie kontynuować i odsunąć stosowanie funkcji oceny na daną liczbę zapisów do przodu. Stosowna rekurencyjna implementacja przedstawiona jest w algorytmie 4.6. W wywołaniu rekurencyjnym w wierszu 5

Ocena-W-Głąb$(K_1, c_1, p_1, K_2, c_2, p_2, d)$
argumenty: opisy zapisu (K_1, c_1, p_1) i (K_2, c_2, p_2) gracza pierwszego i drugiego oraz głębokość w liczbie zapisów do przodu d
wynik: ocena sytuacji na d zapisów do przodu w danym stanie zapisu

```
1:  if d = 0 then
2:      return Ocena(P[K_1, c_1] + p_1, P[K_2, c_2] + p_2)
3:  foreach R ∈ ℛ ∧ k ∈ Kategorie-Wolne(K_1) do
4:      (K'_1, c'_1, p'_1) ← Zapis((K_1, c_1, p_1), R, k)
5:      zapis[R, k] ← 1 − Ocena-W-Głąb(K_2, c_2, p_2, K'_1, c'_1, p'_1, d − 1)
6:  Propaguj-Wartości-Stanów-Rundy(K_1)
7:  return wynik
```

Algorytm 4.6: Ocena sytuacji w głąb.

role gracza pierwszego się zamieniają, stąd otrzymany wynik jest z punktu widzenia gracza drugiego. Odpowiednim przekształceniem zmieniamy tę wartość

na wynik z punktu wiedzenia gracza pierwszego.

Funkcja Ocena-W-Głąb daje ocenę sytuacji tylko dla stanów zapisu. Do określenia wartości stanów rundy wystarczy odpowiednia propagacja. W algorytmie 4.7 przedstawiono strategię korzystającą z tej funkcji oceniającej.

Strategia-Z-Oceną-W-Głąb($K_1, c_1, p_1, K_2, c_2, p_2$, *rzuty*, R_1, d)
argumenty: opisy zapisu (K_1, c_1, p_1) i (K_2, c_2, p_2) gracza pierwszego i drugiego, liczba wykonanych rzutów *rzuty*, aktualny rzut R_1 gracza pierwszego oraz głębokość oceny d, $d \geqslant 1$
wynik: posunięcie — zatrzymanie kości $Z \subseteq R_1$, bądź zapisywana kategoria $k \in$ Kategorie-Wolne(K_1)

1: **foreach** $R \in \mathcal{R} \wedge k \in$ Kategorie-Wolne(K_1) **do**
2: $\quad (K_1', c_1', p_1') \leftarrow$ Zapis$\big((K_1, c_1, p_1), R, k\big)$
3: $\quad$ *zapis*$[R, k] \leftarrow 1 -$ Ocena-W-Głąb$(K_2, c_2, p_2, K_1', c_1', p_1', d-1)$
4: Propaguj-Wartości-Stanów-Rundy(K_1)
5: **if** *rzuty* < 3 **then**
6: $\quad$ **return** $Z \subseteq R_1$ **maximizing** $zatrzymanie_{rzuty}[Z]$
7: **else**
8: $\quad$ **return** $k \in$ Kategorie-Wolne(K_1) **maximizing** *zapis*$[R_1, k]$

Algorytm 4.7: Strategia w grze dwuosobowej z przeszukiwaniem w głąb.

W dalszej części przyjmiemy skrócone nazewnictwo na strategie uzyskane z użyciem różnych funkcji oceny. Funkcję Strategia-Z-Oceną z użyciem oceny OcenaDynW będziemy nazywać strategią $DynW(0)$, a funkcję Strategia-Z-Oceną-W-Głąb z oceną OcenaDynW będziemy nazywać strategią $DynW(d)$, gdzie d jest argumentem określającym głębokość przeszukiwań. Analogiczne nazewnictwo przyjmujemy w przypadku, gdy używamy funkcji oceniającej Ocena-EMP, mianowicie $EMP(0)$ dla Strategia-Z-Oceną i $EMP(d)$ dla Strategia-Z-Oceną-W-Głąb.

4.3.1 Usprawnienia

Funkcja Ocena-W-Głąb daje tym lepszą ocenę im większa jest głębokość d. Zostanie to pokazane w dalszej części. Aby wyliczać wartości tej funkcji dla możliwie największych głębokości i w rozsądnym czasie, potrzebne są usprawnienia implementacyjne.

Bufor podręczny dystrybucji. W funkcji Ocena-W-Głąb w momencie, gdy dochodzi do wywołania funkcji oceniającej Ocena, odczytywane są odpowiednie dystrybucje z tablicy P, która składowana jest na dysku. Indeksy K i c, dla których odczytujemy dystrybucje, niewiele się różnią przy małych głębokościach, a wręcz często się powtarzają. Aby przyspieszyć dostęp, ostatnie i najczęściej odczytywane dystrybucje można przechowywać w pamięci podręcznej.

Tablica haszująca. Wiele różnych układów kości, dla wybranej kategorii do zapisu, prowadzi do tego samego stanu zapisu. Jest tak dlatego, że wiele zapisów może być po prostu zerowych. Co więcej kilka różnych zapisów po rząd często może prowadzić do identycznych stanów zapisu. Wynika stąd, że funkcja

Ocena-W-Głąb będzie wielokrotnie wywoływana z tymi samymi argumentami. Żeby za każdym razem nie wyliczać pracowicie ponownie tych samych wartości, wprowadza się spamiętywanie z użyciem tablicy haszującej. Ta optymalizacja ma największe przełożenie na efektywność funkcji Ocena-W-Głąb.

Tablice końcówek. W przypadku, gdy łączna liczba wolnych kategorii u obu graczy jest nieduża, to możemy odczytać dokładną ocenę z tablicy końcówek strategii optymalnej. Jeżeli funkcja Ocena-W-Głąb dochodzi do głębokości, na której są już dostępne wartości optymalne, to nie potrzebujemy już czytać z tablicy dystrybucji P, lecz możemy użyć tablicy końcówek EMP. W takim wypadku funkcja Ocena-W-Głąb zwróci nam wartość dokładną, tzn. oczekiwaną liczbę punktów meczowych w strategii optymalnej. To usprawnienie wpływa na jakość funkcji oceniającej.

4.4 Wyniki eksperymentalne

W tym punkcie zaprezentujemy wyniki szeregu eksperymentów. Skoncentrujemy się na analizie strategii $EMP(\cdot)$. Pokażemy jej zachowanie w praktyce. Między innymi sprawdzimy, jak dobrze przybliża ona strategię optymalną. Pokażemy też, jak słabo zachowują się inne strategie w grze dwuosobowej. Ponadto zmierzymy, jak dalecy od strategii optymalnej są ludzie i ile tak naprawdę daje użycie strategii optymalnej w tej grze losowej przeciwko takim nieoptymalnym graczom.

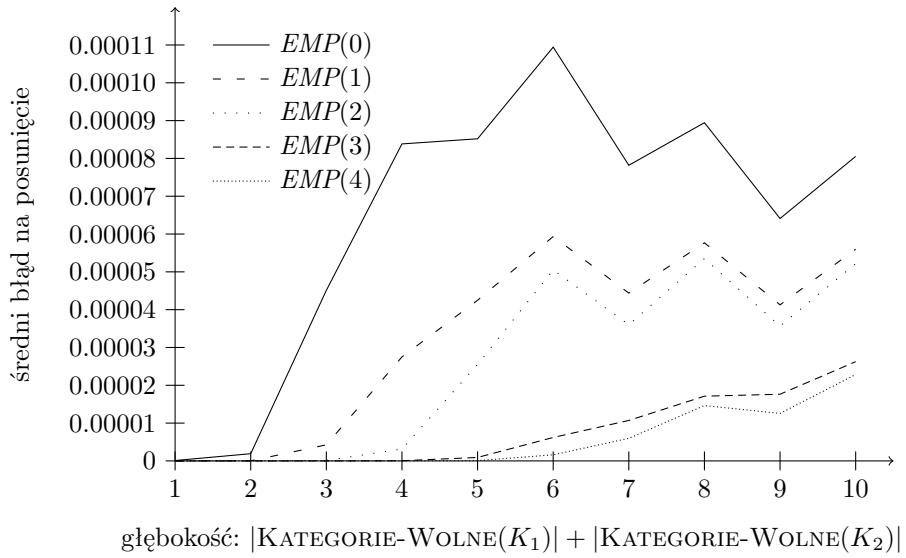
4.4.1 Siła strategii $EMP(\cdot)$

Jak zmierzyć siłę strategii? Najlepiej porównać daną strategię do optymalnej strategii. Strategia polega na wybieraniu posunięć w momentach, gdy gracz dostaje wybór. To, czy posunięcie jest optymalne lub to, czy posunięcie jest błędne, możemy stwierdzić, jeśli potrafimy dla danych sytuacji określić optymalną wartość gry, tj. oczekiwaną liczbę punktów meczowych jaką możemy zdobyć przy założeniu, że obaj gracze grają optymalnie.

Załóżmy, że jesteśmy w pewnym stanie gry i mamy n wyborów posunięć. Niech $s_1, \ldots, s_n$ oznaczają sytuacje po wykonaniu każdego z tych posunięć. Przedstawione dotychczas strategie działają tak, że oceniają liczbowo każdą z sytuacji i wybierają posunięcie, które prowadzi do sytuacji o największej wartości. Załóżmy, że potrafimy dla danych stanów policzyć wartości optymalne i oznaczamy je $v(s_1), \ldots, v(s_n)$. Bez straty ogólności możemy przyjąć, że $v(s_1) \geqslant v(s_2) \geqslant \ldots \geqslant v(s_n)$. Oznacza to, że najlepszym posunięciem jest s_1. Jeżeli strategia wskaże posunięcie s_i, to możemy powiedzieć, że wartość pomyłki dla tego posunięcia wynosi $v(s_1) - v(s_i)$. Innymi słowy, po wykonaniu posunięcia s_i średnio zdobędziemy o $v(s_1) - v(s_i)$ punktów meczowych mniej, niż gdybyśmy zagrali optymalnie.

Wartość optymalną potrafimy liczyć w stanach, w których łączna liczba kategorii wolnych u obu graczy jest dosyć mała. W przeprowadzonym eksperymencie skupiliśmy się na stanach, w których liczba ta nie przekracza 10. Dzięki tablicom końcówek do głębokości 5 wolnych kategorii i przeszukiwaniu w głąb, możliwe jest określanie dla takich stanów wartości optymalnej w rozsądnym czasie. Dla

każdej liczby wolnych kategorii od 1 do 10 wygenerowaliśmy po $2^{16} = 65\,536$ takich stanów, co daje w sumie aż 655 360 wszystkich stanów. Dla każdego z tych stanów, wyliczaliśmy wartości optymalne dla każdego możliwego posunięcia. Następnie braliśmy posunięcie wyznaczone przez strategię $EMP(d)$, $d = 0, 1, 2, 3, 4$, i wyliczaliśmy błąd tego posunięcia. W ten sposób dostajemy średni błąd na posunięcie strategii $EMP(\cdot)$ w zależności od łącznej liczby wolnych kategorii. Wynik przedstawiony jest na rysunku 4.1.



Rysunek 4.1: Średni błąd na posunięcie strategii $EMP(d)$, dla $d = 0, 1, 2, 3, 4$, w zależności od liczby wolnych kategorii u obu graczy.

Wyjaśnienia wymaga jeszcze sposób generacji takich stanów. Otóż interesują nas „losowe" stany, które pojawiają się podczas „typowych" rozgrywek. Aby wygenerować taki stan należy przeprowadzić rozgrywkę „typowymi" graczami. Ponieważ gra zależy nie tylko od wyborów graczy, ale także od zdarzeń losowych, więc za typowego gracza zapewne wystarczy przyjąć jakąkolwiek strategie, która rzadko popełnia rażące błędy. W związku z tym użyliśmy strategii $EMP(1)$, dodatkowo rozmywając wybór posunięcia odpowiednio wyważonym losowaniem. Mianowicie dla posunięć $s_1, \ldots, s_n$, wpierw określaliśmy ich wartości aplikując strategię $EMP(1)$, a następnie wagę posunięcia $w(s_i)$ liczyliśmy według wzoru:

$$w(s_i) = e^{100 \cdot \left(v(s_i) - \max_j v(s_j)\right)},$$

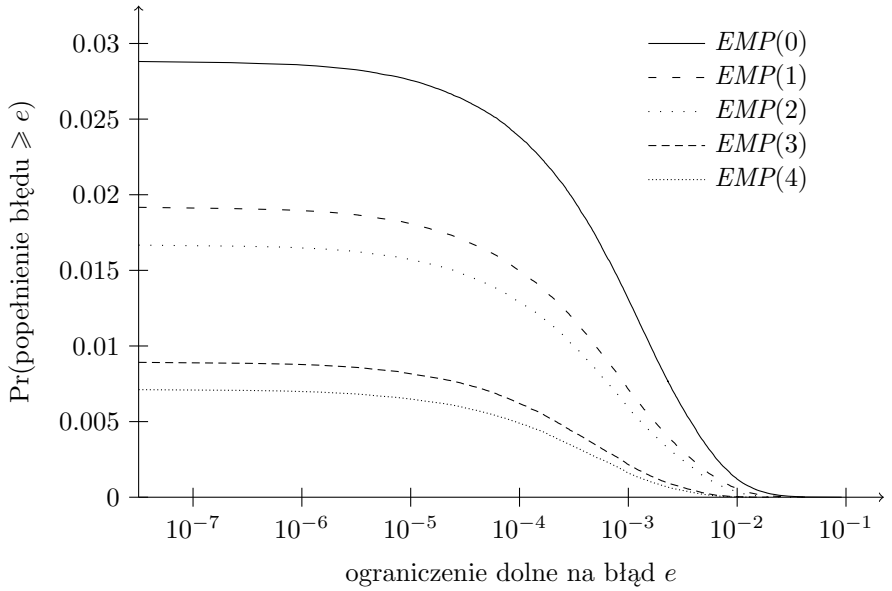
gdzie $v(s_i)$ jest wartością danego posunięcia. Posunięcie było losowane, przy czym prawdopodobieństwo każdego posunięcia było proporcjonalne do jego wagi. W ten sposób, jeśli istniało kilka dobrych posunięć, to możliwe było zagranie dowolnego z nich z dużym prawdopodobieństwem, a posunięcia słabe zagrywane były zdecydowanie rzadziej.

Na wykresie przedstawionym na rysunku 4.1 widzimy, że średni błąd na posunięcie jest bardzo mały. Z wyjątkiem jednego przypadku nie przekracza on 0.0001, dla każdej strategii i głębokości. Widzimy, że strategia $EMP(d)$ jest tym

lepsza, im większe jest d. Wzrost liczby wolnych kategorii, czyli odległość od końca gry, powinien powodować coraz większe odstępstwo od strategii optymalnej, ponieważ dokładna ocena staje się coraz trudniejsza. Jednak na wykresie nie obserwujemy takiego zachowania. $EMP(0)$ największy średni błąd 0.00011 osiąga dla stanów o głębokości 6, po czym wraz ze wzrostem głębokości stopniowo i nieregularnie maleje. Z kolei średnie błędy $EMP(1)$ i $EMP(2)$ rosną aż do głębokości 6 i od tego momentu oscylują w przedziale $[0.00004, 0.00006]$. Jedynie dla strategii $EMP(3)$ i $EMP(4)$ obserwujemy ciągły wzrost błędu wraz ze wzrostem głębokości, jednakże tutaj średnie błędy nie przekraczają 0.00003.

Dlaczego nie obserwujemy wzrostu błędu wraz ze wzrostem głębokości? Przypuszczalnie wiąże się to z tym, że po pierwsze wraz z odległością do końca gry maleje liczba krytycznych decyzji, a po drugie ze względu na dużą liczbę elementów losowych, różnice wartości między posunięciami są tym mniejsze, im więcej rund jest do końca gry.

Oprócz średnich błędów popełnianych przez strategie $EMP(\cdot)$, interesująca może być dystrybucja tych błędów. Dla każdej strategii, dla każdej możliwej wartości błędu e, policzyliśmy liczbę stanów, w których błąd był większy lub równy od e. Po podzieleniu przez liczbę stanów 655 360 otrzymujemy dystrybucję. Wykresy dystrybucji błędów strategii $EMP(d)$, dla $d = 0, 1, 2, 3, 4$, przedstawione są na rysunku 4.2.



Rysunek 4.2: Prawdopodobieństwo popełnienia błędu większego niż zadany w strategii $EMP(d)$, dla $d = 0, 1, 2, 3, 4$, przy założeniu, że jest co najwyżej 10 kategorii wolnych łącznie u obu graczy.

We wszystkich strategiach większość błędów jest z przedziału $[10^{-5}, 10^{-2}]$. Prawdopodobieństwo, że strategia $EMP(0)$ popełni błąd jest mniejsze od 0.03; strategie $EMP(1)$ i $EMP(2)$ mylą się z prawdopodobieństwem mniejszym niż 0.02, a strategie $EMP(3)$ i $EMP(4)$ z prawdopodobieństwem mniejszym niż 0.01. W praktyce stosujemy strategię $EMP(4)$. Dla tej strategii prawdopodobieństwo popełnienia błędu większego niż 0.01 nie przekracza 0.000 014, a prawdopo-

dobieństwo popełnienia błędu większego niż 0.001 nie przekracza 0.0016, co w praktyce nie wiele różni się od strategii optymalnej.

Największy błąd jaki udało się uzyskać dla strategii *EMP*(4) wynosi 0.014. Jest to następująca sytuacja. Każdy z graczy ma po 5 kategorii wolnych. Opisem zapisu gracza pierwszego jest $(12456\mathrm{KMY}_0, 59, 111)$, a gracza drugiego $(12345\mathrm{TFS}, 34, 102)$. Gracz pierwszy jest po trzecim rzucie ⚀⚀⚂⚄⚅ i musi podjąć decyzję, którą kategorię zapisać. W tabeli 4.3 przedstawione są wartości jakie zwracają niektóre strategie dla dwóch najlepszych posunięć. Najlepsze

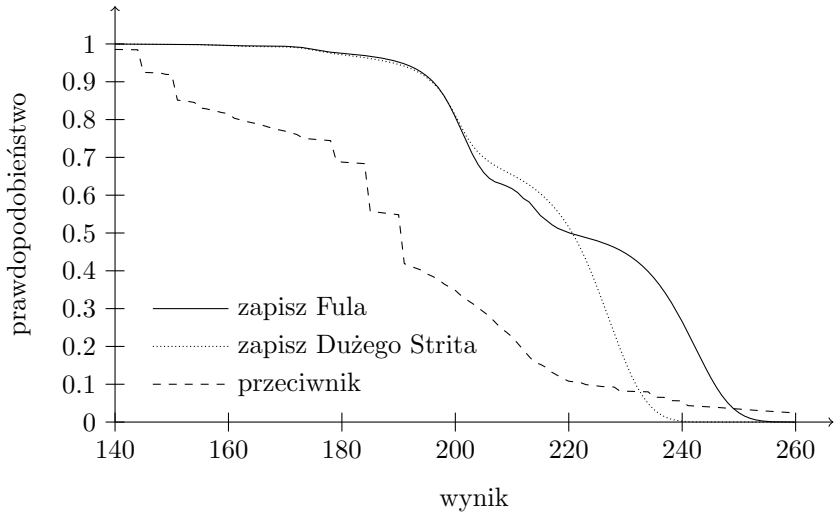
	zapisz Fula	zapisz Dużego Strita
E	**215.178**	212.067
EMP(1)	**0.807**	0.797
EMP(2)	**0.811**	0.802
EMP(3)	**0.803**	0.798
EMP(4)	**0.807**	0.804
EMP(5)	0.795	**0.798**
strategia optymalna	0.788	**0.802**

Tabela 4.3: Oceny posunięć przez różne strategie w sytuacji z opisem zapisu gracza pierwszego $(12456\mathrm{KMY}_0, 59, 111)$, opisem zapisu gracza drugiego $(12345\mathrm{TFS}, 34, 102)$ i po trzecim rzucie gracza pierwszego ⚀⚀⚂⚄⚅.

posunięcia to skreślenie Fula lub Dużego Strita. *EMP*(4) wyceniło wartość skreślenia Fula na 0.807 podczas, gdy faktyczna wartość (wg. strategii optymalnej) jest znacznie mniejsza i wynosi 0.788, natomiast skreślenie Fula zostało wycenione na 0.804 podczas, gdy faktyczna wartość wynosi tu 0.802. Zatem optymalne jest skreślenie Dużego Strita, a strategia *EMP*(4) wybierze skreślenie Fula. Dodajmy, że strategia *EMP*(5) nie popełni już tutaj błędu, więc w tym przypadku wystarczy wykonać przeszukiwanie o jeden poziom głębiej.

Przeanalizujmy dokładniej powyższą sytuację. Na rysunku 4.3 przedstawione są dystrybucje dla posunięć pierwszego gracza zapisz Fula i zapisz Dużego Strita oraz dystrybucja dla drugiego gracza. Są to dystrybucje maksymalnych prawdopodobieństw osiągnięcia poszczególnych wyników. Dystrybucje te są użyte do wyliczenia oceny OcenaEMP w strategii *EMP*(1). Zauważmy, że z perspektywy dystrybucji, zapisanie Fula wygląda na znacznie lepsze posunięcie od zapisania Dużego Strita. Dystrybucje prawie się pokrywają aż do 200 punktów, później do 220 nieznacznie większe prawdopodobieństwa osiągnięcia danego wyniku daje zapisanie Dużego Strita, a dla większych punktacji dużo lepsze jest zapisanie Fula. Jednak różnica ocen jaką daje strategia *EMP*(1) jest niewielka — zapisanie Fula: 0.807, a zapisanie Dużego Strita: 0.797. Bierze się to stąd, że z dystrybucji dla przeciwnika wynika, że prawdopodobieństwo osiągnięcia przez niego wyniku w przedziale między 200 i 220 punktów jest znacznie większe niż przekroczenie 220 punktów.

Zastanówmy się dlaczego w strategii optymalnej skreślenie Dużego Strita jest lepsze od skreślenia Fula. Kategorie wolne gracza pierwszego to: Trójki, Trójka, Ful, Duży Strit i Szansa. Duży Strit jest za 40 punktów, a Ful za 25. Wydaje się więc, że warto w następnych rundach próbować rzucać do Dużego Strita. W przypadku niepowodzenia jednak możliwe jest, że trzeba będzie coś skreślić narażając się na straty. Aby ich uniknąć możemy zapisywać Trójki lub Szansę. Do premii za szkółkę brakuje 4 punktów, czyli przy zapisywaniu Trójek trzeba



Rysunek 4.3: Dystrybucje gracza pierwszego dla zapisu Fula i Dużego Strita w sytuacji z opisem zapisu $(12456\mathrm{KMY}_0, 59, 111)$ i po trzecim rzucie ⚀⚀⚁⚄⚅, oraz dystrybucja gracza drugiego w stanie z opisem zapisu $(12345\mathrm{TFS}, 34, 102)$.

mieć co najmniej dwie ⚄. W przypadku nieudanej próby uzbierania Dużego Strita jest raczej mała szansa posiadania dwóch ⚄, zatem zapisywanie Trójek w takiej kłopotliwej sytuacji jest zupełnie nieopłacalne. Jeżeli by brakowało tylko 3 punktów do premii, czyli wystarczałaby tylko jedna ⚄, wtedy opłacałoby się w kolejnych rundach próbować wyrzucić Dużego Strita, ale niestety taka sytuacja nie zachodzi.

Oprócz Trójek, w przypadku kłopotów, kategorią pomocniczą jest Szansa. Aczkolwiek posiadanie tej kategorii wolnej w końcówkach, w których obydwaj gracze mają podobną liczbę punktów, jest sporym atutem i nie opłaca się jej w takich sytuacjach przedwcześnie zapisywać. Wiąże się to z tym, że Szansa daje możliwość regulacji ryzyka, tzn. zależnie od tego ile brakuje nam punktów do przeciwnika, zupełnie inaczej dokonujemy wyborów kości do rzutów. W ekstremalnej sytuacji, kiedy obu graczy ma po tyle samo punktów i obydwaj mają tylko jedną wolną kategorię – Szansę, wydaje się, że obydwaj mają dokładnie takie same prawdopodobieństwo wygranej. Jednak drugi gracz ma przewagę. Wynika to z tego, że swoje decyzje uzależnia on od tego co wyrzuci i zapisze gracz pierwszy. Dokładna wartość tej sytuacji z punktu widzenia gracza drugiego wynosi 0.518.

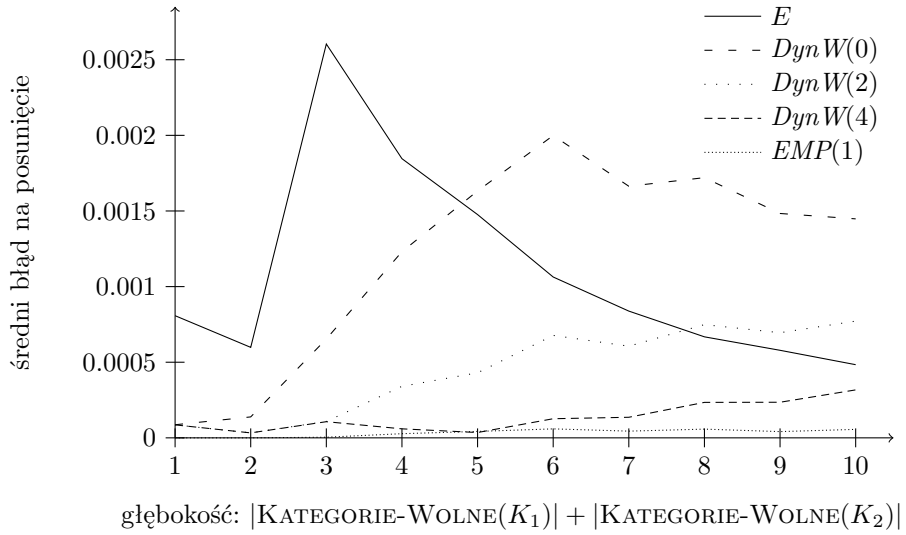
Podsumowując, w przypadku zostawienia sobie wolnego Dużego Strita na potem, nie będziemy mieli za bardzo okazji do uzyskania go, ze względu na brak kategorii alternatywnych, które moglibyśmy zapisywać w przypadku niepowodzenia. Z kolei posiadanie wolnych kategorii Ful, Trójka i Szansa, daje dużo możliwości uzbierania Fula z jednoczesnym zabezpieczeniem w przypadku niepowodzenia. Strategia w takim wypadku jest mniej więcej taka. W przypadku wyrzucenia trzech jednakowych kości, mamy już pewną Trójkę, wtedy w dalszych rzutach próbujemy pozostałymi dwoma kośćmi wyrzucić parę w celu uzbierania Fula. Gdy wyrzucimy dwie ⚄ zostawiamy je i z reszty również próbujemy złożyć

Fula, ewentualnie zapisując Trójki. W innych przypadkach próbujemy zostawiać tylko kości o dużej liczbie oczek, aby uzyskać wysoko punktowaną Trójkę, może nawet Fula, a w najgorszym razie wysoko punktowaną Szansę.

Widać, że posiadanie wolnego Fula zamiast Dużego Strita ma sporo zalet, jednak, aby je dostrzec, trzeba bardziej się wgłębić w niuanse taktyczne. Heurystyka OcenaEMP nie wyłapuje takich szczegółów. Dopiero wraz ze wzrostem głębokości przeszukiwań d strategia $EMP(d)$ jest coraz bardziej czuła. W tabeli 4.3 widzimy, że różnica w ocenie skreśl Dużego Strita do oceny skreśl Fula, stopniowo rośnie wraz z głębokością d. Kolejno dla $d = 1, 2, 3, 4, 5$ różnica ta wynosi $-0.010, -0.009, -0.005, -0.003, 0.003$ i dopiero dla $d = 5$ jest dodatnia. Dla strategii optymalnej różnica ta wynosi aż 0.014.

4.4.2 Inne strategie

W poprzednim punkcie pokazaliśmy, że strategie $EMP(\cdot)$, czyli strategie przybliżające wartość oczekiwaną punktów meczowych przez dystrybucje wraz z przeszukiwaniem w głąb, są praktycznie optymalne. Pojawia się pytanie, czy inne strategie, jak strategia E — strategia na maksymalną wartość oczekiwaną (niezależna od sytuacji przeciwnika), czy też strategie $DynW(\cdot)$ — dynamiczne strategie na ustalony wynik poprawione przeszukiwaniem w głąb, są również w praktyce bliskie optymalnej strategii? Siłę strategii zmierzyliśmy w ten sam sposób, jak siłę strategii $EMP(\cdot)$ w poprzednim punkcie. Wynik przedstawiony jest na rysunku 4.4. Oprócz wybranych strategii, dla porównania dodaliśmy do



Rysunek 4.4: Średni błąd na posunięcie kilku wybranych strategii w zależności od liczby wolnych kategorii u obu graczy.

wykresu strategię $EMP(1)$. Po pierwsze można zauważyć, że średni błąd strategii E oraz strategii $DynW(\cdot)$ jest wielokrotnie większy niż strategii $EMP(1)$.

Heurystyka użyta w strategii $DynW$ jest dosyć naturalna i wygląda na dobrze dobraną. Jednakże ocena sytuacji w strategii EMP okazuje się być znacznie

lepsza. Świadczy to o wyjątkowo trafnej heurystyce wartości oczekiwanej punktów meczowych na podstawie dystrybucji, lub też o słabości strategii *DynW*. To drugie jednak możemy wykluczyć, gdyż widzimy na przykład, że średni błąd strategii *DynW*(4) jest znacznie mniejszy od średniego błędu strategii *E*.

W ogóle średnie błędy na posunięcie przedstawionych tutaj strategii zazwyczaj nie przekraczają wartości 0.002. Mogłoby się wydawać, że wszystkie strategie są bliskie optymalnej. Należy zwrócić uwagę, że podane wartości są średnim błędem na posunięcie, a w czasie jednej gry może zostać wykonanych nawet $3 \cdot 13 = 39$ posunięć. Zakładając, że średni błąd na posunięcie wynosi 0.001, otrzymamy, że łączna strata na mecz wynosi mniej więcej 0.04. Oznacza to, że na 100 rozgrywek przegramy średnio o 4 mecze więcej niż w przypadku użycia strategii optymalnej, a to dla tak losowej gry należy uznać raczej za sporą stratę.

Na koniec zauważmy, że strategia jednoosobowa na maksymalną wartość oczekiwaną, całkiem dobrze sprawdza się w grze dwuosobowej o ile liczba wolnych kategorii jest dosyć duża u obu graczy, bądź też aktualny gracz ma tylko jedną kategorię wolną. Natomiast w końcówkach, gdy dany gracz ma co najmniej dwie kategorie wolne, strategia *E* popełnia już dosyć duże błędy.

Przykład 4.4. Rozważmy rozgrywkę dwuosobową. Załóżmy, że gracz pierwszy jest w stanie z opisem zapisu $(123456\text{KFMDS}, 51, 198)$ i wyrzucił w trzecim rzucie ⚀⚀⚀⚄⚂. Natomiast gracz drugi jest w stanie z opisem zapisu $(12356\text{TKFMDS}, 39, 205)$. Gracz pierwszy ma dwa możliwe posunięcia: zapisać Trójkę lub zapisać Yahtzee. Maksymalna wartość oczekiwana wyniku pierwszego posunięcia to 214.301, a drugiego to 213.195. Zatem przy grze na maksymalną wartość oczekiwaną wyniku najlepiej zapisać Trójkę. Natomiast oczekiwana liczba punktów meczowych, przy strategii optymalnej, przy zapisie Trójki wynosi 0.130, podczas gdy przy zapisie Yahtzee wynosi 0.540. W tym przypadku granie na maksymalną wartość oczekiwaną wyniku powoduje znaczne zmniejszenie szans na wygranie z przeciwnikiem.

4.4.3 Zastosowanie w praktyce

Strategia *EMP*(*d*) wraz z tablicą końcówek jest bardzo silną strategią. W końcówkach jest to strategia optymalna, a w pozostałych etapach gry jest strategią „prawie" optymalną. W punkcie 4.4.1 pokazaliśmy, że im większa głębokość przeszukiwań *d*, tym strategia *EMP*(*d*) mniej różni się od strategii optymalnej. Jaką w praktyce możemy stosować głębokość tak, żeby czas na posunięcie był w okolicach jednej sekundy na komputerze klasy Pentium 4 z zegarem 2.8 GHz?

Skupimy się na grze dwuosobowej. Rozpatrzmy wpierw stany bliskie końca gry, czyli sytuacje, w których jest mało kategorii wolnych, gdyż wtedy jest najwięcej krytycznych decyzji. Tablice końcówek jesteśmy w stanie przechowywać dla stanów zapisu, w których łączna liczba wolnych kategorii u obu graczy nie przekracza 5. Przy 6, 7 i 8 wolnych kategoriach możemy stosować strategię *EMP*(3), co wraz z użyciem tablicy końcówek jest równoznaczne ze strategią optymalną. Przy 8 wolnych kategoriach czas na wyliczenie wartości dla stanów rundy waha się w przedziale od 100 do 600 milisekund. Przy 9 wolnych kategoriach potrzebujemy od 2 do 10 sekund na użycie *EMP*(4), zatem tutaj nie możemy sobie już pozwolić na użycie strategii optymalnej.

W sytuacjach, w których nie możemy zastosować strategii optymalnej, czyli takich, w których łączna liczba wolnych kategorii u obu graczy wynosi 9 i więcej,

możemy stosować strategię $EMP(2)$ lub nawet $EMP(3)$. Na początku gry czas liczenia $EMP(2)$ wynosi około 600 milisekund, $EMP(3)$ aż kilkanaście sekund. Zatem w tym stadium gry praktyczna jest tylko strategia $EMP(2)$. Kolejne zapisane kategorie redukują zapotrzebowanie czasowe. W okolicach 14–15 rundy (czyli, gdy jest 11–12 kategorii wolnych, łącznie u obu graczy) czasy wyliczania $EMP(3)$ spadają poniżej 1 sekundy i od tego momentu można stosować już tą strategię aż do końca gry.

4.4.4 Analiza gier ludzi

W tym punkcie postaramy się pokazać, ile tak naprawdę można zyskać w grze z ludźmi stosując strategię optymalną. Dzięki uprzejmości serwisu z grami kurnik.pl [Fut07] otrzymaliśmy logi z wielu gier rozegranych w roku 2007. Wśród nich wybrane zostały mecze, w których brały udział dokładnie dwie osoby i wszystkie zostały przeprowadzone do końca, tzn. nie skończyły się przed upływem czasu. Do analizy zostało użytych 24 352 929 meczów, w których brało udział 209 207 użytkowników. Tak duża liczba meczów daje już statystycznie obiektywne wyniki.

Dla każdej gry i dla każdej sytuacji liczyliśmy wpierw wartość optymalną lub „prawie" optymalną każdego posunięcia. Na tej podstawie porównywaliśmy najlepsze posunięcie z posunięciem wykonanym przez danego gracza i rejestrowaliśmy różnicę określającą wielkość błędu. Dla sytuacji, w których było co najwyżej 8 wolnych kategorii (czyli 18 zapisanych) u obu graczy, byliśmy w stanie policzyć wartości posunięć z użyciem strategii optymalnej. Natomiast dla sytuacji, w których było więcej niż 8 wolnych kategorii u obu graczy, stosowaliśmy strategie możliwie najbliższe optymalnej — $EMP(1)$ lub nawet $EMP(d)$ z d większym od 1 blisko początku gry lub blisko końca gry. Takie przybliżenie strategii optymalnej mimo wszystko powinno dać wymierne oceny. W punkcie 4.4.1 widzieliśmy, że strategia $EMP(1)$ średnio na jedno posunięcie nie myli się więcej niż 0.0001. Możemy zatem przyjąć, że nasze oszacowanie na średni błąd na posunięcie u analizowanych graczy jest policzone z dokładnością do czterech miejsc po przecinku. Natomiast pomyłka przy liczeniu średniego błędu na mecz jest większa $3 \cdot 18 = 54$ razy (3 posunięcia w ciągu rundy i 18 rund, dla których stosowaliśmy nieoptymalną strategię $EMP(d)$ do oceny posunięcia), czyli w przybliżeniu nie mylimy się więcej niż 0.005.

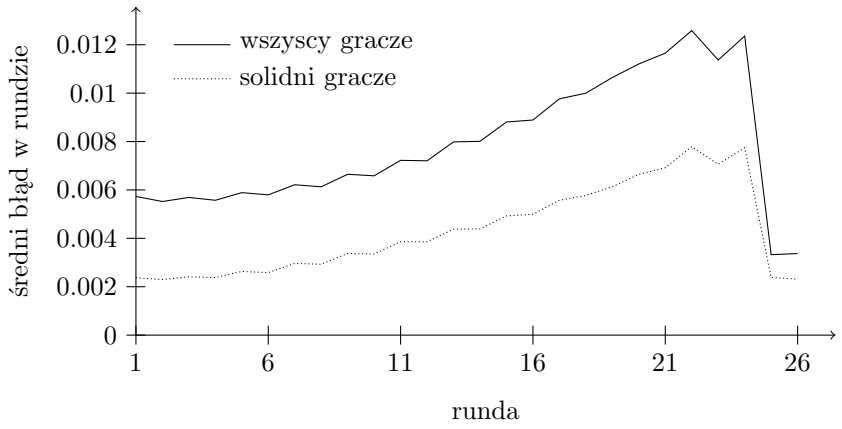
Dla każdego gracza zostały zsumowane wszystkie błędy, policzona liczba meczów i wykonanych posunięć. W ten sposób dla każdego gracza uzyskaliśmy średni błąd jaki gracz popełnia na mecz i na posunięcie. Siłę gracza najlepiej odzwierciedla średni błąd na mecz. Ponieważ większość graczy rozegrała małą liczbę meczy, więc wielu z nich szczęśliwie może mieć tą statystykę wygórowaną, ze względu na małą liczbę sytuacji, w których musieli podjąć krytyczną decyzję. W związku z tym przyjrzyjmy się tylko graczom, którzy rozegrali co najmniej 500 partii. Takich graczy jest 23 009, czyli nieco ponad 10% wszystkich analizowanych graczy. Ci gracze zostali posortowani ze względu na średni błąd na mecz. W połowie stawki, czyli w okolicach miejsca 10 500, średni błąd na mecz wynosi 0.093, a średni błąd na posunięcie wynosi 0.0025. Wśród czołówki jest 179 graczy, których średni błąd na mecz jest mniejszy od 0.05 i tylko czterech, którzy średnio nie mylą się o więcej niż 0.04. Średni błąd najlepszego gracza o loginie simson11 wynosi 0.0347 i rozegrał on 1133 pojedynki. Jego średni błąd na posunięcie wynosi 0.000 95. Co oznacza, że trochę mu brakuje do optymalnej

gry. W eksperymentach dotyczących strategii *EMP* widzieliśmy, że średni błąd na posunięcie nie przekracza 0.000 06.

Najlepsi gracze popełniają porównywalne błędy do strategii na maksymalną wartość oczekiwaną. Weźmy pod uwagę tylko sytuację, w których jest 8 lub mniej kategorii wolnych u obu graczy, czyli sytuacje, dla których policzyliśmy dokładne wartości posunięć. Średni błąd na posunięcie najlepszej dziesiątki graczy jest mniejszy niż 0.0013, a czołowej trójki jest mniejszy niż 0.001 23. Średni błąd na posunięcie dla strategii na maksymalną wartość oczekiwaną został policzony w punkcie 4.4.2. Dla rozważanych sytuacji wynosi on 0.001 24. Widzimy, że najlepsi gracze są w stanie wygrywać ze strategią na wartość oczekiwaną. Oczywiście możliwe, że statystyki dla najlepszych graczy są efektem szumu, jednak każdy z badanych graczy rozegrał odpowiednio dużo meczy. Na przykład, każdy z trzech najlepszych graczy rozegrał ich grubo ponad tysiąc.

Podsumowując, na 100 gier, dzięki stosowaniu strategii optymalnej, średnio wygramy od 4 partii więcej, dla najlepszych graczy, do 9 partii więcej, dla średnich graczy. Nie są to znaczące liczby, co świadczy o losowym charakterze gry oraz o znaczącej liczbie sytuacji, w których decyzja jest oczywista.

Przyjrzyjmy się jeszcze, które fazy gry są najtrudniejsze dla ludzi. Wśród 12 484 graczy, którzy rozegrali 1 000 i więcej partii, wybraliśmy 1 000 najlepszych zawodników, tzn. takich, dla których średni błąd na mecz jest najmniejszy. Tych wyselekcjonowanych graczy nazwiemy *solidnymi*. Na rysunku 4.5 pokazane jest jak rozkłada się średni sumaryczny błąd w meczu wszystkich graczy i solidnych graczy na poszczególne rundy, których jest 26 (po 13 na gracza).



Rysunek 4.5: Średni błąd na rundę wszystkich graczy oraz solidnych graczy.

Widzimy, że im bliżej końca rozgrywki tym wybory są coraz trudniejsze i popełnianych jest coraz więcej błędów. Jedynie, gdy graczowi zostaje jedna kategoria do zapisania, to wybór staje się dużo łatwiejszy, ale mimo to ludzie potrafią robić spore błędy przy podejmowaniu najlepszych decyzji.

Rozdział 5

Podsumowanie

Ta praca zawiera pierwszą, tak dokładną analizę gry Yahtzee wraz z implementacjami strategii optymalnych lub „prawie" optymalnych, zarówno dla wersji jednoosobowej, jak i wieloosobowej. W rzeczywistości wersja wieloosobowa nigdy wcześniej nie była analizowana. W tej pracy całkowicie zmieniliśmy status wiedzy o grze dwuosobowej w kości Yahtzee, a także zaprezentowaliśmy silne strategie dla większej liczby osób.

Wkład tej pracy dla wersji jednoosobowej jest następujący:

1. szczegółowy opis stanów gry Yahtzee wraz z możliwymi redukcjami stanów,
2. efektywna implementacja strategii na maksymalną wartość oczekiwaną,
3. wprowadzenie pojęcia dystrybucji,
4. stablicowanie strategii maksymalizującej prawdopodobieństwo osiągnięcia określonego wyniku.

Dla wersji wieloosobowej podajemy:

5. implementację tablicowania strategii optymalnej dla dwóch osób,
6. analizę zasobów potrzebnych do wyliczenia strategii optymalnej dla dwóch osób,
7. strategię heurystyczną wysokiej jakości w wersji wieloosobowej, wykorzystującą pojęcie dystrybucji,
8. wyniki eksperymentów wykazujące „prawie" optymalność strategii heurystycznej $EMP(\cdot)$,
9. analizę siły innych strategii oraz ludzi.

Wprowadzenie dystrybucji i tworzenie na ich podstawie heurystyk oceniających jest nową techniką, która może być stosowana w innych grach losowych z pełną informacją. Oprócz Yahtzee istnieje wiele innych gier tego typu, które są popularne i które bada się metodami informatycznymi i matematycznymi. Przykładem jest gra o nazwie *Can't Stop*. W ostatnich latach poświęcono jej sporo uwagi [GFK07a, GFK07b]. Ta gra jest jednak trudniejsza do analizy od

gry Yahtzee, gdyż występują w niej cykle. Metoda propagacji wyników wymaga użycia dodatkowych technik, jak na przykład aproksymacja metodą Newtona [GFK07a, GFK07b]. O ile dla wersji jednoosobowej jest szansa na pełne stablicowanie strategii optymalnej, o tyle w wersji dwuosobowej jest to praktycznie niemożliwe. Podejście z użyciem dystrybucji mogłoby dać „prawie" optymalne strategie dla tej gry.

Dodatek A

Program

Do pracy dołączona jest płyta DVD z następującymi plikami:

- `yahtzee.zip` — zawiera program do gry w Yahtzee wraz z tablicami niezbędnymi do realizacji przedstawionych strategii,
- `svnrepo.tgz` — zawiera kompletne repozytorium SVN źródeł programu wraz z wszystkimi odmianami użytymi do generowania eksperymentów,
- `thesis.pdf` — ten dokument.

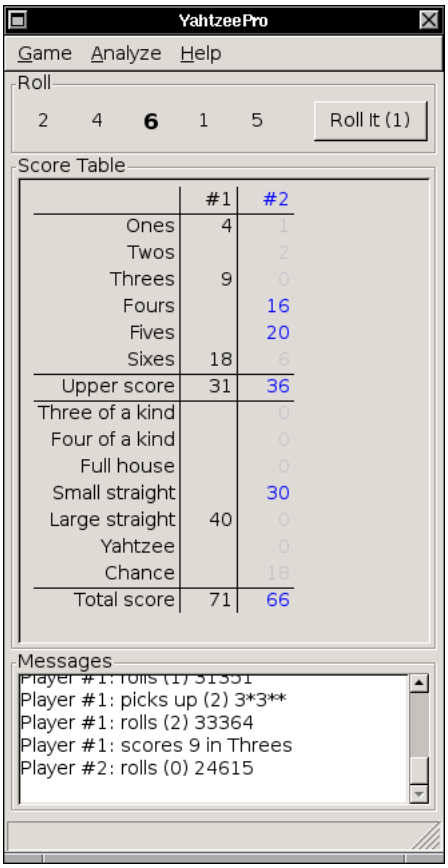
A.1 Program do gry w Yahtzee

W archiwum `yahtzee.zip` znajdują się następujące pliki i katalogi:

- `yahtzee` — program skompilowany statycznie pod system Linux,
- `yahtzee.exe`, `mingwm10.dll` — program skompilowany statycznie pod system Windows,
- `E.ye` — plik zawierający tablice do strategii na maksymalną wartość oczekiwaną wyniku,
- `dist` — katalog zawierający tablice do strategii na maksymalne prawdopodobieństwo osiągnięcia określonego wyniku oraz tablice z oczekiwaną liczbą punktów meczowych w grze dwuosobowej dla sytuacji z łączną liczbą wolnych kategorii u obu graczy nie przekraczającą 4; tablice te po rozpakowaniu zajmują 6.7 GB.

Aby uruchomić program z wykorzystaniem wszystkich tablic należy rozpakować archiwum. Program `yahtzee` należy uruchamiać w katalogu, w którym znajduje się plik `E.ye` i katalog `dist`. Program umożliwia grę w Yahtzee. Podstawowy interfejs jest intuicyjny. Po uruchomieniu pojawia się główne okno programu (rysunek A.1). Aby rozpocząć grę należy z menu `Game` wybrać pozycję `New game`. Dodatkowa funkcjonalność jaką udostępnia menu `Game` jest następująca:

- zapisywanie i wczytywanie stanu gry,
- możliwość ręcznego podawania wyrzucanych kości,

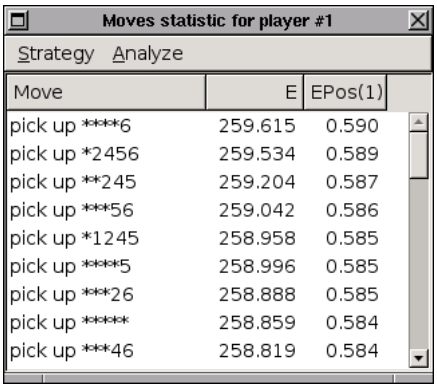


Rysunek A.1: Główne okno programu.

- możliwość grania z komputerem — używana jest strategia $EMP(1)$,
- pamiętanie historii gry — możliwość cofania ruchów (undo i redo).

Oprócz wykonywania rozgrywki program umożliwia dokonywanie prostych analiz. Podstawową statystyką jest wyświetlenie wartości wszystkich możliwych posunięć dla różnych strategii. Z menu `Analyze` należy wybrać pozycję `Move statistics` i pojawia się okno ze statystyką posunięć (rysunek A.2). W tym oknie możemy wybierać strategie, dla których chcemy poznać wartości dla możliwych posunięć. Tabela A.1 przedstawia dostępne strategie do analizy. Zwróćmy uwagę, że nazewnictwo użyte w programie różni się od przedstawionego w pracy. Wynika to z tego, że program powstał znacznie wcześniej i dopiero w tej pracy terminologia została uporządkowana. Posunięcia można sortować po wartościach wybranej strategii poprzez kliknięcie w etykietę odpowiedniej kolumny.

Drugiem rodzajem analiz jest wyświetlanie wykresu dystrybucji (rysunek A.3). Możemy wyświetlać dystrybucję aktualnego stanu dla każdego z graczy oraz dystrybucję każdego z posunięć dostępnych dla bieżącego gracza. Wykres dystrybucji możemy wywołać na kilka sposobów. Z menu `Analyze` okna głównego pozycja `Current state distribution` da wykresy dystrybucji aktualnego stanu każdego z graczy. Z kolei w oknie statystyki posunięć możemy zaznaczyć



Rysunek A.2: Okno z wartościami posunięć.

Pozycja w menu `Strategy`	Nazwa kolumny	Nazwa używana w pracy
Expected score	E	maksymalna wartość oczekiwana wyniku
Given score	S $\geqslant w$	maksymalne prawdopodobieństwo osiągnięcia wyniku w
Given probability	P $\geqslant p$	ta strategia nie została opisana w pracy
Best expected position	EPos(d)	$EMP(d)$
Best weighted position	WPos(d)	$DynW(d)$

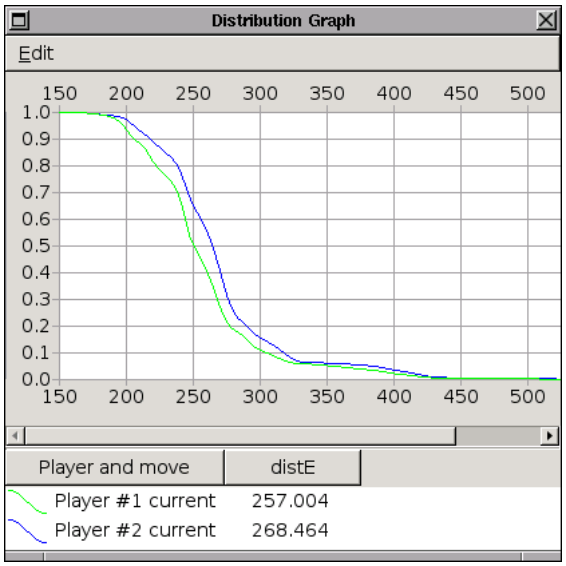
Tabela A.1: Zestawienie strategii.

niektóre posunięcia, a pozycja `Selected moves distribution` z menu `Analyze` otworzy wykresy dystrybucji wybranych posunięć.

Należy dodać, że program pozwala wygenerować wszystkie tablice. W tym celu należy go uruchomić z odpowiednimi opcjami (w nawiasach przybliżony czas potrzebny na wygenerowanie danych tablic na maszynie Pentium 4 2.8 GHz):

- `--gen-E`: generuje tablicę dla strategii na maksymalną wartość oczekiwaną wyniku (30 sek.),
- `--gen-dist`: generuje tablicę prawdopodobieństw na potrzeby strategii na maksymalne prawdopodobieństwo osiągnięcia określonego wyniku oraz szybkiego otrzymywania dystrybucji dla zadanych sytuacji (6 godz.),
- `--gen-escore`: generuje tablicę oczekiwanych punktów meczowych w wersji dwuosobowej dla sytuacji, w których łączna liczba wolnych kategorii u obu graczy nie przekracza 4 (6 dni).

Generowanie tablic można przerywać w dowolnym momencie sygnałem `SIGINT` lub z klawiatury `Ctrl-C`. Dodatkowo tworzone są punkty kontrolne na wypadek nieoczekiwanego przerwania obliczeń, na przykład wskutek zaniku prądu. Przy generowaniu tablicy oczekiwanych punktów meczowych dozwolona łączna liczba wolnych kategorii ustawiona jest na 4, ale stałą tą można zmienić w źródłach programu. Generowanie tablic z większa liczbą wolnych kategorii wymaga jednak



Rysunek A.3: Okno z wykresem dystrybucji.

znacznie więcej czasu i dostępnej przestrzeni dyskowej, co zostało dokładniej opisane w punkcie 4.1.3.

A.2 Źródła programu

Program został napisany w języku `C++` z użyciem biblioteki `wxWidgets` w wersji 2.6. W pliku `svnrepo.tgz` znajduje się archiwum z repozytorium SVN. W archiwum znajduje się jeden katalog `svnrepo`, w którym znajduje się właściwe repozytorium. Podczas rozwoju programu powstało szereg gałęzi rozwojowych w celu przeprowadzenia różnych eksperymentów. Najważniejsze gałęzie to:

- `yahtzee/trunk` — główna gałąź zawierająca źródła udostępnionego programu,
- `yahtzee/branches/experiments/state_stats` — wersja zliczająca zasoby pamięciowe potrzebne do stablicowania gry dwuosobowej,
- `yahtzee/branches/experiments/test_strategies-2` — wersja służąca do eksperymentalnego mierzenia siły różnych strategii w grze dwuosobowej.
- `yahtzee/branches/kurnik_games_stats` — wersja z dodatkowymi opcjami służącymi do parsowania i analizy gier z portalu `kurnik.pl`.

Bibliografia

[AvdHH96] L. Victor Allis, H. Jaap van den Herik, and M. P. H. Huntjens. Gomoku solved by new search techniques. *Computational Intelligence*, 12:7–23, 1996.

[BBB+08] Yngvi Bjornsson, Martin Bryant, Neil Burch, Joe Culberson, Akihiro Kishimoto, Rob Lake, Paul Lu, Martin Mueller, Jonathan Schaeffer, Steve Sutphen, and Norman Treloar. Chinook, 2008. `http://www.cs.ualberta.ca/~chinook/`.

[BCG04] Elwyn R. Berlekamp, John Horton Conway, and Richard K. Guy. *Winning Ways for Your Mathematical Plays*, volume 1, 2, 3 and 4. A K Peters, Ltd., 2nd edition, 2001, 2003, 2004.

[Bur97] Michael Buro. The Othello match of the year: Takeshi Murakami vs. Logistello. *ICCA Journal*, 20(3):189–193, 1997.

[Che06] Kramnik vs Deep Fritz: Computer wins match by 4:2, 2006. `http://www.chessbase.com/newsdetail.asp?newsid=3524`.

[CLR97] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Wprowadzenie do algorytmów*. WNT, Warszawa, 1997.

[Cre02] C.J.F. Cremers. How best to beat high scores in Yahtzee: A caching structure for evaluating large recurrent functions. Master's thesis, Fac. of Math. and CS, Technische Universiteit Eindhoven, The Netherlands, 2002.

[Fut07] Marek Futrega. Kurnik – gry online, 2007. `http://www.kurnik.pl/`.

[GFK07a] James Glenn, Haw-ren Fang, and Clyde P. Kruskal. A retrograde approximation algorithm for one-player Can't Stop. In H.J. van den Herik, P. Ciancarini, and H.H.L.M. Donkers, editors, *5th International Conference on Computers and Games (CG2006)*, volume 4630 of *Lecture Notes in Computer Science*, pages 148–159. 2007.

[GFK07b] James Glenn, Haw-ren Fang, and Clyde P. Kruskal. A retrograde approximation algorithm for two-player Can't Stop. In *Computers and Games Workshop*, 2007.

[Gin99] Matthew L. Ginsberg. GIB: Steps toward an expert-level Bridge-playing program. In *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 584–589, 1999.

[Gle06] James Glenn. An optimal strategy for Yahtzee. Technical report, Department of Computer Science, Loyola College in Maryland, 2006.

[Gle07] James Glenn. Computer strategies for solitaire Yahtzee. In *2007 IEEE Symposium on Computational Intelligence and Games (CIG2007)*, pages 132–139, 2007.

[Hee06] Wim Heemskerk. Computer Bridge. *Dutch Bridge Magazine IMP*, 2005–2006.

[Ide04] The great idea finder – Yahtzee, 2004. `http://www.ideafinder.com/history/inventions/yahtzee.htm`.

[Joh07] Michael Johanson. Robust strategies and counter-strategies: Building a champion level computer Poker player. Master's thesis, University of Alberta, 2007.

[KK06] Kuijf and Kuijf Software. Program do gry w Brydża Jack, 2006. `http://www.jackbridge.com/`.

[RK08] Vasik Rajlich and Larry Kaufman. Program szachowy Rybka, 2008. `http://www.rybkachess.com/`.

[RV02] John Romein and Kees Verstoep. The Awari oracle, 2002. `http://awari.cs.vu.nl/awari/`.

[SBB+07] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.

[SP97] Jonathan Schaeffer and Aske Plaat. Kasparov versus Deep Blue: The re-match. *ICCA Journal*, 20(2):95–102, 1997.

[Var08] Jim Varnon. The second man vs machine Poker championship, 2008. `http://www.stoxpoker.com/man_vs_machine.html`.

[Ver99] Tom Verhoeff. Optimal solitaire Yahtzee advisor and Yahtzee proficiency test, 1999. On-line since July 1999: `http://www.win.tue.nl/~wstomv/misc/yahtzee/`.

[Woo03] Phil Woodward. Yahtzee: The solution. *Chance*, 16(1):17–20, 2003.