Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Informatyka w Multimediach

Algorytmy wspomagania gracza w grze Scrabble

Andrzej Sawicki
Numer albumu 304092

promotor
dr inż. Sebastian Plamowski

WARSZAWA 2025

Algorytmy wspomagania gracza w grze Scrabble

Streszczenie. Scrabble to gra słowna w którą gra od dwóch do czterech graczy. Gracze układają litery ze stojaków na planszy do gry, którą stanowi siatka o wymiarach 15 na 15 tak, aby utworzyć słowa. Przynajmniej jedna litera musi zostać położona przy innej, która już się znajdowała na planszy. Za otrzymane słowa otrzymywane są punkty. Wysokość nagrody punktowej zależy od wartości liter oraz pól premiowanych. Gracz z najwyższym wynikiem wygrywa.

Ta praca skupia się na analizie skuteczności algorytmów do gry Scrabble w wariancie turniejowym. Jest to wariant, w którym biorą udział jedynie dwaj gracze. Informacje o stanie worka z literami oraz stanu stojaka przeciwnika są niedostępne, więc gra jest z początku niedeterministyczna, a dopiero później deterministyczna, co wymaga wykorzystania różnych algorytmów w zależności od etapu gry. Przedstawiona jest platforma do testowania algorytmów z fragmentami kodu, a także wyniki symulacji.

Słowa kluczowe: scrabble, algorytmy, symulator, sztuczna inteligencja

Player supporting algorithms in Scrabble game

Abstract. Scrabble is a word game played by two to four players. Players arrange letters from racks on a game board, which is a 15x15 grid, to create words. At least one letter must be placed next to another that is already on the board. Points are awarded for the words obtained. The amount of the point award depends on the value of the letters and the bonus fields. The player with the highest score wins.

This work focuses on the analysis of the effectiveness of algorithms for the tournament version of Scrabble. This is a version in which only two players participate. Information about the state of the bag with letters and the state of the opponent's rack is unavailable, so the game is initially non-deterministic and only later deterministic, which requires the use of different algorithms depending on the stage of the game. A platform for testing algorithms with code fragments is presented, as well as simulation results.

Keywords: scrabble, algorithms, simulator, artificial inteligence

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

……..
miejscowość i data

……..
imię i nazwisko studenta

……..
numer albumu

…….…….
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wprowadzenie . . . 9
1.1. Historia gier planszowych . . . 9
1.2. Porównanie z innymi grami . . . 9
1.3. Sztuczna inteligencja w grach planszowych . . . 10
1.4. Początek rozwoju badań nad grą . . . 11
1.5. Problematyka implementacyjna . . . 11
1.6. Konkurencja między człowiekiem i komputerem . . . 11
1.7. Przegląd literatury . . . 12
1.8. Cel pracy . . . 14
1.9. Przegląd pracy . . . 14
2. Symulator rozgrywki . . . 16
2.1. Wprowadzenie . . . 16
2.2. Wykorzystane narzędzia . . . 16
2.3. Klasy i struktury danych . . . 16
2.4. Przegląd grafów . . . 17
2.4.1. Drzewo trie . . . 18
2.4.2. DAWG . . . 19
2.4.3. GADDAG . . . 20
2.5. Pozycje kandydatów . . . 22
2.6. Redukcja problemu do jednego wymiaru . . . 25
2.7. Generacja ruchu . . . 26
2.8. Wartość ruchu . . . 29
2.9. Rozgrywka między graczami . . . 29
2.10. Podsumowanie . . . 30
3. Analiza rozgrywek . . . 31
3.1. Wprowadzenie . . . 31
3.2. Zdobycz punktowa jako funkcja pozycji . . . 31
3.3. Zdobycz punktowa jako funkcja etapu gry . . . 31
3.4. Zdobycz punktowa jako funkcja płytek . . . 33
3.5. Zdobycz punktowa jako funkcja słownika . . . 36
3.6. Wpływ niedeterminizmu na przebieg rozgrywki . . . 37
3.6.1. Mydełka . . . 38
3.6.2. Litery . . . 38
3.7. Podsumowanie . . . 42
4. Ewaluacja stojaka . . . 43
4.1. Wprowadzenie . . . 43
4.2. Metodyka . . . 43

4.3. Strategia zachłanna . . . 43
4.4. Strategia Ballarda . . . 44
4.5. Strategie Gordona . . . 45
4.5.1. Balans liter . . . 45
4.5.2. Balans samogłosek i spółgłosek . . . 46
4.6. Podsumowanie . . . 47
5. Końcowy etap rozgrywki . . . 48
5.1. Wprowadzenie . . . 48
5.2. Szybkie zakończenie . . . 48
5.3. Drzewo Minimax . . . 49
5.4. Strategia zachłanna przeciwnika . . . 50
5.5. Podsumowanie . . . 51
6. Podsumowanie . . . 52
6.1. Ograniczenia . . . 52
6.2. Wnioski . . . 53
6.3. Możliwości w przyszłości . . . 54
6.4. Dalsze badania . . . 54
Bibliografia . . . 57
Spis rysunków . . . 59
Spis tabel . . . 60

1. Wprowadzenie

Niniejszy rozdział stanowi wprowadzenie do dalszej, zarówno inżynierskiej, jak i badawczej części pracy. Omówiona jest w nim historia gry Scrabble oraz porównuje się tę grę z innymi grami planszowymi, wskazując na cechy gry, które powodują, że jest unikalna. Następnie omawiana jest historia rozwoju sztucznej inteligencji w grach planszowych, problemy związane z implementacją gier, pierwsze starcia ludzi, którzy byli przeciwnikami napisanych na komputerach symulatorów. Podkreślona zostaje popularność gry jako przedmiotu prac dyplomowych. Wskazana jest literatura, która najczęściej jest przywoływana w pracy. Określony cel pracy, wskazana unikalność względem innych prac dyplomowych tego rodzaju. Na końcu znajduje się przegląd pozostałej części pracy dyplomowej.

1.1. Historia gier planszowych

Najwcześniejsza gra planszowe były grana przez Egipcjan 5000 lat przed naszą erą. Zasady tej gry pozostają jednak nieznane. Jedną z najstarszych znanych gier jest "Senet", która była grana 3500 lat p.n.e. Uchodziła za ulubioną grę faraonów. W przypadku tej gry pierwotny zestaw reguł także nie przetrwał do naszych czasów, obecnie rozgrywana jest na nieco innych zasadach. Gry planszowe z tamtych czasów często były nasycone znaczeniami religijnymi i symbolicznymi, odzwierciedlającymi wierzenia i wartości poszczególnych cywilizacji. W starożytnym Egipcie wierzono, że Senet reprezentuje podróż duszy przez zaświaty, a elementy gry symbolizują duchowy rozwój gracza [1]. Wraz z upływającym czasem na różnych częściach globu pojawiały się kolejne gry planszowe, w tym chaturanga datowana do 1500 p.n.e. w Indiach, która jest podobna do szachów. Inna z gier, taka jak Go, wywodzi się z Chin i sądzi się, że grywał w nią cesarz Shun w 2220 roku p.n.e. Scrabble, w odróżnieniu od tych gier, została wymyślona w 1931 roku przez bezrobotnego architekta Alfreda Moshera Buttsa w stanie Nowy Jork, który opatrzył ją nazwą Criss-Cross.

1.2. Porównanie z innymi grami

Scrabble różni się od innych gier w sposób znaczący. W przypadku innych gier planszowych elementy gry tj. pionki znajdują się na planszy do rozgrywki, najczęściej ułożone w ten sam sposób w początkowym stanie gry. Scrabble w stanie początkowym jest pustą planszą, na której nie znajduje się nic, zaś litery w worku są losowo pobierane przez graczy. To powoduje, że jest bardzo dużo różnych stanów początkowych tej gry. Dodatkowo, w grach planszowych najczęściej elementy gry można ułożyć na zaledwie kilka różnych sposobów, gdzie najlepszym kontrprzykładem byłyby warcaby, w których ruch pionka można wykonywać tylko na dwa sposoby. W Scrabble przeciętnie w każdej turze można ułożyć 100 różnych słów, w porywach do 10000 różnych słów, gdy gracz posiada dwa

mydełka w swoim stojaku i korzystny rozkład samogłosek i spółgłosek. Kolejną różnicą między tymi grami a Scrabble jest brak różnych wartości pionków. W szachach różne figury jedynie wyznaczają sposób, w jaki mogą się poruszać. W Scrabble każda litera ma swoją wartość, a po ułożeniu słowa gracz musi wykonać wysiłek w postaci działań matematycznych do obliczenia wartości ruchu. Ostatnią wymienioną różnicą jest stan determinizmu w grze. W przypadku innych wspomnianych gier każdy z graczy dysponuje pełną informacją, znając pozycje zarówno swoje, jak i przeciwnika. W przypadku gry Scrabble, gracz przed ostatnim etapem gry nie zna stanu worka ani stojaka przeciwnika.

Gra Scrabble jest zarówno niedeterministyczna, jak i deterministyczna, co sprawia, że jest świetną platformą do testowania metod sztucznej inteligencji. Od stanu początkowego do stanu, w którym worek z płytkami jest pusty, gra jest niedeterministyczna, ponieważ gracz nie jest w stanie określić, jakie płytki znajdują się zarówno w stojaku przeciwnika, jak i jakie płytki pozostały w worku. O końcowym etapie gry, gdy worek jest pusty, dysponując informacją o wszystkich literach, które są używane w grze, gracz może obliczyć zawartość stojaka przeciwnika, odejmując litery z planszy oraz własnego stojaka od wszystkich liter biorących udział w rozgrywce. A przynajmniej tak to wygląda w przypadku rozgrywki turniejowej, w której bierze udział jedynie dwóch graczy. To sprawia, że w zależności od etapu gry różne algorytmy będą najskuteczniejsze, w niedeterministycznej będą to rozmaite heurystyki, zaś w deterministycznej np. Minimax.

1.3. Sztuczna inteligencja w grach planszowych

Celem stosowania sztucznej inteligencji w grach planszowych jest znalezienie optymalnego ruchu, który gracz powinien wykonać, by wygrać rozgrywkę. W innych grach niż Scrabble, w których znana jest pełna informacja, zastosowane algorytmy można podsumować jako minimax lub jeden z jego wariantów. Każdy z węzłów takiego drzewa minimax reprezentuje stan planszy wraz z elementami na nim się znajdującymi oraz pewną wartością, która wskazuje, czy gracz jest na pozycji wygrywającej lub przegrywającej rozgrywkę, gdyby drugi z graczy grał bezbłędnie. Każda z krawędzi w drzewie reprezentuje przejście sprowadzające się do wykonania ruchu, zaznaczając, że krawędzie zawsze są jednokierunkowe, a drzewo tworzy graf acykliczny. Korzystając z takiego drzewa, poszukuje się optymalnego ruchu w grach innych niż Scrabble. W przypadku Scrabble, wygenerowanie takiego drzewa z zestawem wszystkich liter, które *być może* znajdują się w stojaku gracza, mogłoby przekroczyć zasoby pamięciowe nawet najnowszych komputerów, zaś czas, który byłby potrzebny do symulacji rozgrywania gry w przód od obecnego stanu gry, przekroczyłby czas, który ma gracz na wykonanie ruchu zgodnie z zasadami rozgrywki turniejowej. Liczba różnych stanów gry została oszacowana przez Briana Sheppard i wynosi 2^{1000} [2]. Z tego względu do dyspozycji pozostaje jedynie pewien zestaw heurystyk, które ułatwiłyby podejmowanie decyzji związanych z optymalnym ruchem w grze.

1.4. Początek rozwoju badań nad grą

Około 1980 roku pojawiło się pytanie, czy programy komputerowe mogą być w stanie grać w Scrabble w sposób konkurencyjny. W 1982 roku pojawiły się jedne z pierwszych artykułów, które zostały opublikowane na ten temat [3] [4]. Początkowe programy grały w grę słabo. Były ograniczone przez małe rozmiary pamięci komputerów z tamtych czasów i wyjątkowo mało wydajne procesory. Zasada ich działania ograniczała się do wyszukiwania specjalnych premiowanych pól na planszy, które mogłyby wykorzystać podczas wykonywania ruchu. Jednakże warto wspomnieć, że mimo to osiągały one wartość punktową na poziomie średnio 20 punktów na ruch. Rok później zaimplementowano pierwszy program na infrastrukturze IBMu, który był w stanie wygenerować wszystkie możliwe poprawne ruchy w grze, bazując na okrojonym do około 25 tysięcy słów słowniku do gry. Jedną z poszlak jest to, że okrojony słownik zawierał wszystkie ze słów JQXZ, które są najcenniejsze, a stopień ich użycia w rozgrywkach został opisany w dalszej części pracy w podrozdziale 3.5.

1.5. Problematyka implementacyjna

Scrabble znacznie różni się od innych gier pod kątem implementacyjnym. W innych wspomnianych grach wykonanie ruchu sprowadza się do przesunięcia figury w jedno z dozwolonych miejsc na planszy. Cała implementacja sprowadza się do wygenerowania poprawnych miejsc, na których figura może zostać umieszczona, co jest zadaniem łatwym. W przypadku Scrabble, należy wygenerować wszystkie poprawne słowa, które mogą być ułożone przylegając lub przechodząc przez ułożone już na planszy słowa. Warto do tego dodać fakt, że słownik SOWPODS do gry Scrabble składa się z 267 tysięcy słów, by pojąć, że wygenerowanie listy wszystkich możliwych poprawnych ruchów dla danego stanu planszy i stojaka gracza jest wyzwaniem, gdyż gracz musi znaleźć optymalny ruch w określonym dla rozgrywki turniejowej czasie przeznaczonym na wykonanie ruchu. Wspomina o tym Brian Sheppard, sugerując, że jest to być może największy problem związany z grą [2]. W dalszej części pracy znajduje się porównanie wybranych algorytmów służących do tego celu.

1.6. Konkurencja między człowiekiem i komputerem

Inspiracją do rozwoju badań nad grą Scrabble była konkurencyjność programów komputerowych w grze przeciwko ludziom, przede wszystkim mistrzom świata w tej grze. Napisany przez Briana Shepparda symulator gry MAVEN [2] był w stanie wygrywać około 2/3 gier rozgrywanych przeciwko mistrzom świata. W tym celu organizowano turnieje Scrabble, takie jak jeden ze zorganizowanych przez Alana Franka w grudniu 1986 roku. Alan Frank także był autorem innego programu do gry Scrabble o nazwie TYLER. Finalnie MAVEN, mimo stosowania dość dobrych strategii i algorytmów, przegrał rozgrywkę z Michaelem Wolfbergiem, ekspertem w grze Scrabble. Gra Scrabble, mimo stosowania w

niej coraz lepszych strategii, wciąż pozostaje jedną z tych, w których człowiek może okazać się być lepszy od maszyny.

Istnieją dedykowane techniki do gry w Scrabble zarówno dla ludzi, jak i dla komputerów. Techniki gry dla ludzi różnią się od tych dedykowanych dla komputerów ze względu na to, że człowiek nie jest w stanie w krótkim czasie wygenerować wszystkich możliwych zagrań, które są możliwe do wykonania.

Przeciętna dorosła osoba pochodząca z kraju, w którym pierwszym językiem jest angielski, posiada zasób słownictwa na poziomie 25 tysięcy słów [5]. Dozwolonych w grze słów do 8 liter jest ponad 80 tysięcy. Aby efektywnie układać słowa, należy zacząć od tego, które słowa najlepiej się nauczyć. Niektóre ze słów występują w grze znacznie częściej niż inne, ze względu na prawdopodobieństwo, z jakim dane litery mogą się znaleźć na stojaku gracza. W podrozdziale 3.5 rozwinięto ten temat podczas analizy przeprowadzonych rozgrywek w symulatorze. Ma na to wpływ między innymi ilość liter w grze. Za względu na to utworzono listę słów, która została uporządkowana ze względu na prawdopodobieństwo wystąpienia w grze [6].

Anagramowanie jest fundamentalną umiejętnością ludzkiego gracza w Scrabble. Polega na przekładaniu liter w stojaku w taki sposób, aby uzyskać pewien popularny prefiks lub sufiks. Popularny prefiks lub sufiks to taki, który często występuje w wyrazach. Przykładem jest -ING, który jest typowym zakończeniem czasownika w języku angielskim. W tym przypadku gracz układa literki tak, aby ING znajdowało się na prawym fragmencie stojaka, zaś po lewej od sufiksu znajdowały się wszystkie pozostałe litery. Następnie gracz próbuje ułożyć słowo, przekładając już jedynie pozostałe 4 litery, które nie należą do sufiksu.

Scrabble jest grą powszechnie używaną także jako narzędzie do nauki języka angielskiego dla osób uczących się języków obcych, a także jako umysłowa gra dla inżynierów, którzy uczą się kreatywnego myślenia, logiki i nabywają umiejętności analitycznych, a także rekomendowana jako element, który miałby rozluźniać studentów, a także zachęcać do nauki oraz zadawania pytań i odpowiadania na nie [7].

1.7. Przegląd literatury

Badania nad grą Scrabble są popularne nie tylko pod kątem technik komputerowych. Są również obiektem badań w dziedzinach takich jak psychologia lub edukacja. Niemniej jednak, w dziedzinie informatycznej dominują jako temat prac magisterskich. W większości przypadków część inżynieryjna pracy jest prawie pomijana lub ogranicza się do opisu grafów, które mogłyby być wykorzystane do celów generacji ruchu, a prace skupiają się na badaniu strategii, którą gracz mógłby przyjmować do wyboru najlepszego ruchu w rozgrywce. Przeciętna praca liczy od 30 do 40 stron i znaczna część pracy składa się z tabeli opisujących stan wyniku gracza po przeprowadzeniu rozgrywki. Prace nie zawierają analiz rozgrywki tudzież implementacji symulatora, na którym badania były wykonywane. Nie-

mniej jednak, wciąż stanowią pewne źródło wiedzy, które może służyć do porównywania własnej pracy.

Unikatem w tym obszarze jest praca doktorska Briana Shepparda [2], twórcy programu MAVEN. Zawiera wyjątkowo obszerny opis algorytmów, strategii gry oraz trafne analizy rozgrywek wraz z wnioskami.

W pracy odwołano się do następującej literatury.

Prace studenckie:

- [8] - praca magisterska, w której analizowano grę Scrabble z punktu widzenia pełnej informacji. W pracy porównano kilka strategii gry, m. in. losową, zachłanną lub ewaluacji stojaka. Charakterystyczne dla pracy są wyniki symulacji, w których zawsze zaczynał ten sam gracz z dwóch uczestniczących w rozgrywce.
- [9] - artykuł studencki, w którym podjęto ogólną tematykę gry Scrabble bazując na implementacji silnika gry z użyciem grafu trie. Przedstawiono w nim heurtystykę wyszukiwania poprawnych słów na planszy. W pracy nie odniosiono się do implementacji grafów do przeszukiwania planszy, tj. GADDAG.
- [10] - praca magisterska, w której zaimplementowano grę Scrabble z użyciem drzewa trie. Zawiera porównania w działaniu symulatorów MAVEN oraz QUACKLE. Autor w ramach pracy nie podjął się implementacji rozwiązania.
- [11] - 23-stronna praca magisterska wykorzystująca drzewo trie do generacji, skupiająca się na porównaniu strategii ewaluacji stojaka. Ewaluacja nie została wykonana w sposób zaproponowany przez badaczy z 1990, jedynie metodą uproszczoną.

Artykuły badawcze:

- [6] - artykuł opisujący ludzkie strategie uczenia słów do konkurencyjnej rozgrywki Scrabble oraz badania nad wpływem rozmiaru słownika na przebieg rozgrywki. Przedstawiono także listę najbardziej użytecznych słów do nauki.
- [12] - artykuł wprowadzający do grafu DAWG, który umożliwia szybsze generowanie poprawnych ruchów niż drzewo trie. Został wykorzystany między innymi w symulatorze MAVEN. Charakteryzuje się niskim użyciem pamięci podręcznej.
- [13] - artykuł przedstawiający graf GADDAG, który umożliwia szybsze generowanie poprawnych ruchów niż w przypadku grafu DAWG. Treść rozdziału 2. odnosi się do tego artykułu w znacznym stopniu, podobnie jak zaproponowane autorskie modyfikacje i optymalizacje algorytmów. GADDAG był brany pod uwagę między innymi w implementacji symulatora MAVEN.
- [14] - artykuł podejmujący temat ewaluacji stojaka na zasadzie balansu spółgłosek oraz samogłosek. Opisywana jest także metoda szacowania wartości liter w stojaku. Strategie z tego artykułu zostały zaimplementowane i zaprezentowano ich wyniki w rozdziale 4.

- [15] - artykuł opisujący podejście do ewaluacji stojaka na bazie liter, które się w nim znajdują. Jest to prawdopodobnie pierwszy artykuł o tematyce ewaluacji stojaka w grze Scrabble. Autor wyprodukował tabelę złożoną z dwóch kolumn - litery oraz wartości służącej do szacowania wartości.

Dodatkowo, w pracy odniesiono się do następujących repozytoriów kodu źródłowego:

- [16] - implementacja gry Scrabble wykorzystująca drzewo trie. Została napisana w Javie. To aplikacja do gry z interfejsem graficznym.
- [17] - implementacja drzewa trie do wykorzystania w silniku gry. Napisana w języku Typescript. Zawiera funkcje służące do dodania słów z leksykonu do grafu oraz wyszukiwania wszystkich słów, które można utworzyć z podanego zestawu liter.
- [18] - implementacja Scrabble z wykorzystaniem grafu GADDAG w formie niezminimalizowanej w języku Java. Pozwala na wyszukiwanie najwyżej punktowanego słowa w turze za pomocą posiadanych liter. Autor nie zaimplementował jednak algorytmów wspomnianych w artykule Gordona służących do optymalnego wyszukiwania na planszy.

1.8. Cel pracy

Celem głównym pracy jest zbadanie wybranych strategii ewaluacji ruchu, by zbliżyć się do znalezienia najlepszej z nich, w tym opracowanie swojego autorskiego wariantu strategii ewaluacji stojaka, który byłby konkurencyjny z technikami istniejącymi w literaturze. Dodatkowym celem, który powstał w trakcie realizacji pracy, jest udoskonalenie wybranych algorytmów opisanych w literaturze służących do wyszukiwania poprawnych ruchów za pomocą grafu GADDAG.

Jako cel poboczny wybrano implementację silnika gry z wykorzystaniem grafu GADDAG wraz ze wszystkimi optymalizacjami występującymi w literaturze włącznie, wygenerowanie kilkudziesięciu tysięcy rozgrywek w symulatorze gry, wykonanie szczegółowej analizy wygenerowanych przez symulator rozgrywek oraz implementację istniejących w literaturze wybranych strategii ewaluacji stojaka, celem porównania wyników i przeprowadzenia rozgrywek z zaimplementowaną autorską strategią.

1.9. Przegląd pracy

Niniejsza praca magisterska jest jedną z niewielu prac, w której tak szczegółowo zbadano optymalne algorytmy do generacji listy ruchów w zadanym stanie gry. Jest przede wszystkim jedną z niewielu prac, w której zostało podjęte zaimplementowanie grafu GADDAG i jego badanie, podczas gdy zdecydowana większość prac o tej tematyce bazuje na drzewie trie. W pracy zaprezentowano autorskie modyfikacje istniejących algorytmów przedstawionych w artykułach naukowych, które służą usprawnieniu wyszukiwania

wszystkich poprawnych ruchów na planszy, a także obliczania wartości ruchu i znajdowania pozycji pól kandydatów do rozpoczęcia wyszukiwania. Dodatkowo zbadane zostały algorytmy ewaluacji stojaka w grze zgodnie z zaleceniami autorów artykułów skupiających się na tym obszarze, którzy nie mieli możliwości w czasie pisania swoich prac na podjęcie takich działań jak uczenie maszynowe z propagacją wsteczną ze względu na czasy, w których komputery dysponowały małymi zasobami pamięciowymi oraz obliczeniowymi.

Rozdział 2. wprowadza do grafów wykorzystywanych do generacji ruchu w grze, optymalizacjach stosowanych w generacji ruchu oraz autorskich modyfikacjach algorytmów z artykułów naukowych.

Rozdział 3. jest analizą rozgrywek w grze na bazie wygenerowanych kilkudziesięciu tysięcy starć między graczami w wyprodukowanym wcześniej symulatorze, który został opisany w rozdziale 2.

Rozdział 4. opisuje strategie ewaluacji stojaka, włączając rezultaty przeprowadzonych rozgrywek. Podjęcie tematu ewaluacji stojaka było zainspirowane wynikami analiz z rozdziału 3.

Rodział 5. prezentuje kilka wybranych technik stosowanych w ostatnim etapie rozgrywki. Rozdział poświęcony badaniom w etapie deterministycznym gry.

Rozdział 6. to podsumowanie pracy oraz wprowadzenie do dalszych badań, które mogłyby zostać zrealizowane.

2. Symulator rozgrywki

Repozytorium kodu znajduje się pod adresem:
`https://github.com/andrzej-sawicki-stud/Scrabble`

2.1. Wprowadzenie

W celu przeprowadzenia analizy porównawczej algorytmów wykorzystywanych w grze Scrabble przygotowano program prowadzący rozgrywkę w wersji turniejowej, czyli dla dwóch graczy. W ramach programu zaimplementowano optymalne algorytmy służące do generacji ruchu, obliczania zestawów skrośnych redukujących problem gry do jednego wymiaru, wyszukiwania pól kotwiczących planszy często opisywanych w literaturze jako kandydujących, a także zgodną ze wszystkimi zasadami gry funkcję obliczania wartości ruchu gracza. Niektóre fragmenty implementacji zawierają unikalne autorskie optymalizacje istniejących już w literaturze algorytmów. Przytoczone techniki zostały szczegółowo opisane w poniższym rozdziale.

2.2. Wykorzystane narzędzia

Program został zaimplementowany jako skrypty języka Python. Język Python został wybrany ze względu na popularność w momencie pisania pracy, co bezpośrednio przekłada się na możliwość wykorzystania implementacji przez osoby trzecie, które miałyby potrzebę wykorzystania implementacji do celów własnych badań. W zasobach internetowych brakuje implementacji optymalnego silnika gry Scrabble w tym języku [16][17] lub jest ona napisana w innym języku, który w momencie pisania był popularny, ale obecnie już nie jest [18]. W trakcie produkcji oprogramowania wykorzystano środowisko PyCharm, które uchodzi za jedno z najpopularniejszych środowisk programistycznych do edycji kodu źródłowego z wbudowanym debugerem. Symulacje rozgrywki przeprowadzono na urządzeniu z wbudowanym procesorem AMD Ryzen 5 PRO 6650U. Mimo stosunkowo niewielkiej mocy obliczeniowej modelu, dzięki zastosowaniu optymalnych algorytmów, wspomniana jednostka okazała się wystarczająca, a symulacje wykonywały się w krótkim czasie, wynoszącym średnio 0,415 sekundy na rozgrywkę. Urządzenie wykorzystane do celów symulacji dysponowało 8 GB pamięci podręcznej, która okazała się być wystarczająca, gdyż program w trakcie wykonywania wykorzystywał nie więcej niż 40 MB pamięci podręcznej, głównie do przechowywania wczytanego częściowo zminimalizowanego grafu GADDAG.

2.3. Klasy i struktury danych

Gra Scrabble sprowadza się do zaimplementowania następujących klas, struktur danych oraz funkcji:

- Gracz - przechowujący swoją nazwę, wynik, stojak, strategię wykorzystywaną do podejmowania decyzji, który ruch z zestawu dostępnych ruchów w grze wybrać
- Plansza - przechowująca stan planszy oraz o stan zestawów skrośnych pól na planszy, z funkcją układania słowa
- Worek - przechowujący listę liter w worku, z funkcjami układania liter w losowej kolejności oraz uzupełniania stojaka gracza
- Stojak - przechowujący listę liter na stojaku, z funkcją liczenia wartości liter na stojaku

Przedstawione powyżej klasy znajdują się w katalogu *./structures* projektu.

- Silnik - przechowujący wybrany graf oraz algorytmy z nim związne, które pozwalają na generację listy wszystkich możliwych ruchów, które dla zadanego stanu planszy można wykonać

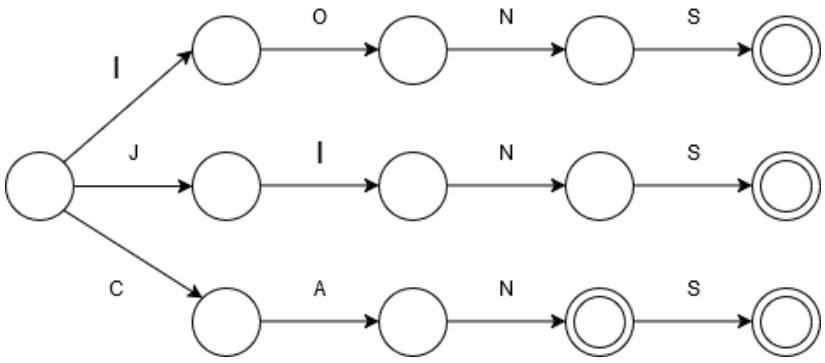
Powyższa klasa znajduje się w katalogu *./algorithms* projektu.

- Ballard - funkcja strategii ewaluacji stojaka bazująca na tabeli wartości z artykułu Ballarda
- Gordon_h2 - funkcja strategii ewaluacji stojaka bazująca na tabeli wartości z artykułu Gordona opierająca się jedynie na wartości liter
- Gordon_h3 - zmodyfikowana strategia gordona_h2, w której dodano dodatkową tabelę w celu zachowania balansu spółgłosek i samogłosek w stojaku
- Asawicki_h3 - autorska strategia nawiązująca do strategii gordon_h3, w której zaimplementowano szybkie zakończenie rozgrywki
- Asawicki_h4 - autorska strategia, w której zaimplementowano drzewo minimax stosowane w fazie deterministycznej rozgrywki

Powyższe funkcje znajdują się w katalogu *./strategies* projektu.

2.4. Przegląd grafów

Gra Scrabble wymaga układania poprawnych językowo słów spełniających reguły prawidłowego ułożenia na planszy. Metoda brutalna polegająca na iteracji po każdym polu planszy do gry oraz układaniu wszystkich istniejących w słowniku do gry słów, a następnie weryfikacji poprawności ułożenia przez sprawdzenie istnienia użytych w ruchu liter na stojaku gracza oraz sprawdzenia poprawności ruchu zgodnie ze stanem planszy, jest wysoko nieefektywna. W celu optymalizacji układania poprawnych słów stosuje się grafy słowne, które są grafami acyklicznymi i skierowanymi. Składają się z węzłów oraz krawędzi, w których każda z krawędzi odpowiada jednej literze z alfabetu. Pojawiającymi się w literaturze grafami wykorzystywanymi w tym celu są drzewo trie, DAWG oraz GADDAG.



Rysunek 2.1. Drzewo Trie dla leksykonu: ions, jins, cans, can.

2.4.1. Drzewo trie

Drzewo trie jest najprostszym słownym grafem popularnie wykorzystywanym w pracach związanych z grą Scrabble. Przykładowe drzewo przedstawiono na rysunku 2.1. Badacze często decydują się na użycie tego grafu w projekcie z powodu prostoty implementacji [10], uznając implementację innych grafów za czasochłonną [11], lub nie wspominając wcale o ich istnieniu [9][8]. Drzewo grafu składa się z węzłów terminalnych i nieterminalnych oraz krawędzi o wartości jednej litery z alfabetu. Graf pozwala na wydajną weryfikację słowa wyłącznie w kolejności od początkowego wyrazu słowa do końcowego. Nie umożliwia optymalnej weryfikacji słowa, zaczynając od środkowej części słowa lub w odwróconej kolejności liter w słowie. Weryfikacja słowa z wykorzystaniem tego drzewa polega na iteracji po każdej literze słowa w kolejności liter w słowie, gdzie w każdej iteracji przechodzi się po krawędzi w grafie odpowiadającej wartości litery do węzła. Poprawność słowa jest spełniona tylko i wyłącznie wtedy, gdy podczas przechodzenia po grafie w każdej iteracji występowała krawędź o wartości litery oraz otrzymany węzeł po iteracjach jest terminalny. Z powodu możliwości weryfikacji słowa jedynie od początkowego wyrazu do końcowego, drzewo trie nadaje się wyłącznie do wyszukiwania poprawnych słów jedynie jako wariant metody brutalnej, ponieważ nie zawęża obszaru poszukiwania do pozycji na planszy z już ułożonymi słowami lub bezpośrednio sąsiadującymi z nimi. Przeszukiwanie za pomocą drzewa trie jest opisane poniższym pseudokodem.

zbiór rezultatów <- pusty zbiór

dla każdego pola na planszy p
    rezultat <- pusty ciąg znaków
    dla kierunku k ze zbioru poziomy, pionowy
        wywołaj funkcję go(p, korzeń drzewa trie, stojak gracza, rezultat, k)

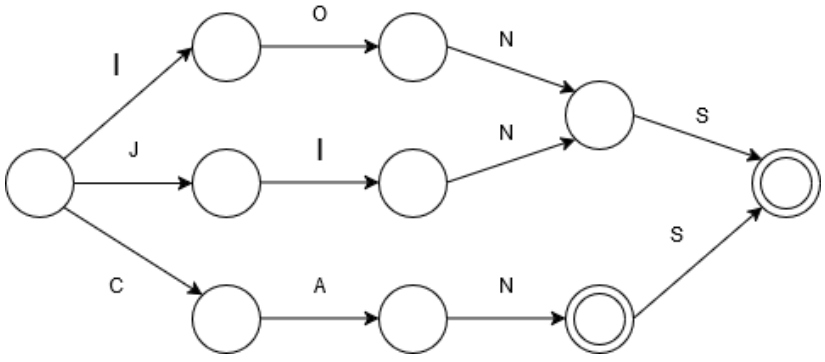
go(p, w, s, r, k)

jeśli pole p jest wolne, wtedy
dla każdej litery l w stojaku s
jeśli litera l jest mydełkiem
dla każdej litery m, ze zbioru wszystkich liter alfabetu
wywołaj funkcję goOn(p, w, s - m, m, r, k)
w przeciwnym wypadku
wywołaj funkcję goOn(p, w, s - l, l, r, k)
jeśli pole p jest zajęte
l <- litera z pola p
wywołaj funkcję goOn(p, w, s, l, r, k)

goOn(p, w, s, l, r, k)
jeśli istnieje krawędź k z węzła w dla litery l
w <- węzeł po przejściu po k
r <- r + l
jeśli węzeł w jest terminalny
zapisz r
p <- kolejna pozycja p zgodna z kierunkiem k
wywołaj funkcję go(p, w, s, r, k)

dla każdego rezultatu r ze zbioru rezultatów
ułóż rezultat r na planszy
zweryfikuj poprawność słów na planszy

2.4.2. DAWG

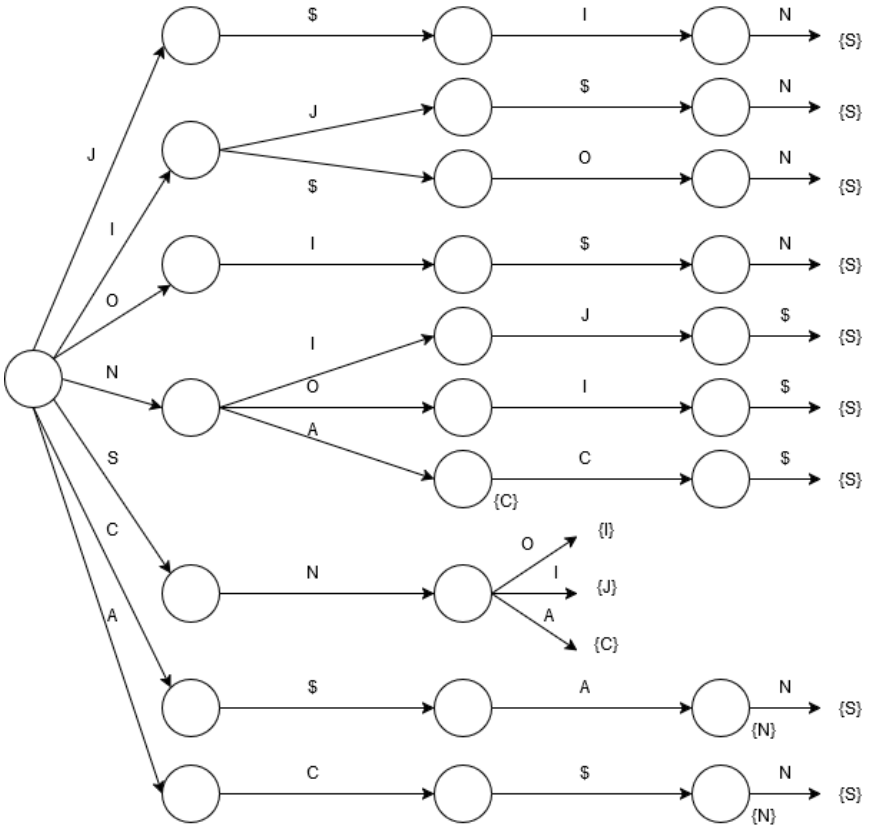


Rysunek 2.2. DAWG dla leksykonu: ions, jins, cans, can.

Graf DAWG jest wariantem drzewa trie, które poddano minimalizacji. Zaprezentowano je w 1988 roku [12]. Przykładowy graf przedstawiono na rysunku 2.2. Sposób użycia grafu DAWG w grze Scrabble jest identyczny jak w przypadku drzewa trie i stosuje się w tym celu ten sam algorytm, który został wcześniej opisany pseudokodem. Rok zaprezento-

wania grafu został celowo przytoczony, by zwrócić uwagę na możliwości obliczeniowe urządzeń w czasie wydania oraz pojemność pamięci podręcznej. Drzewo trie nie mogło się zmieścić w pamięci podręcznej urządzeń tamtych czasów. W artykule związanym z DAWG zaprezentowano technikę wchłaniania lewych części słowa od wskazanej pozycji, co ogranicza ilość pozycji kandydujących do generacji ruchu oraz generację zestawów skrośnych, jednakże jest jedynie optymalizacją metody brutalnej i nie będzie omawiane w tej pracy, gdyż zasada pracy pozostaje taka sama.

2.4.3. GADDAG



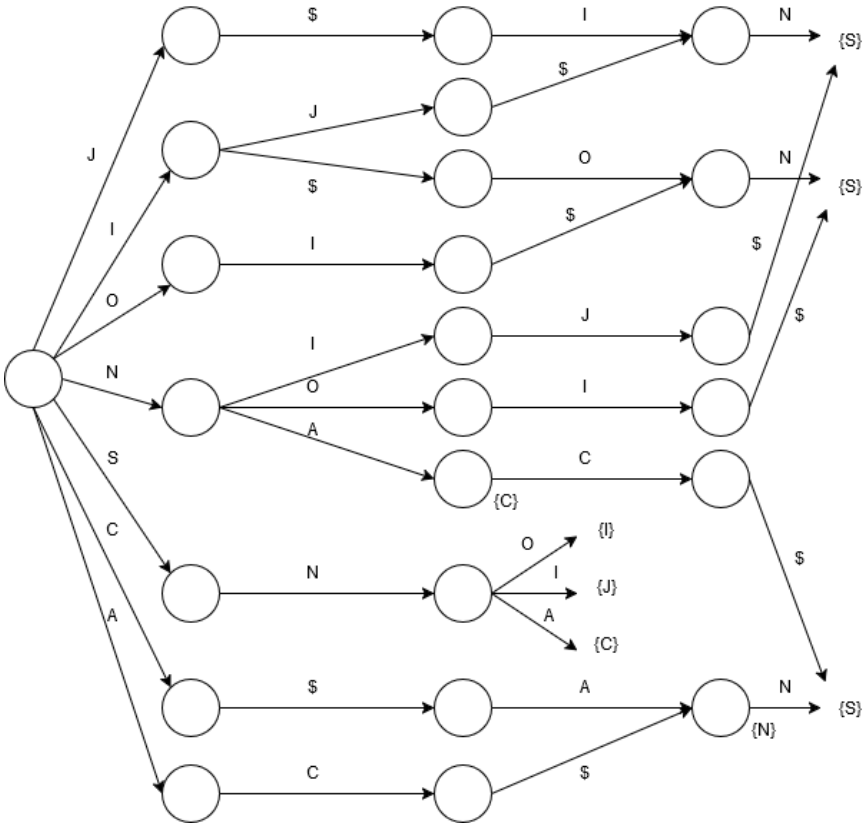
Rysunek 2.3. Niezminimalizowany GADDAG dla leksykonu: ions, jins, cans, can.

GADDAG został zaprezentowany w 1994 roku [13]. Przykładowy graf w postaci niezminimalizowanej przedstawiono na rysunku 2.3. Graf różni się od drzewa trie oraz DAWG. Ilość węzłów oraz krawędzi w tym grafie w postaci zoptymalizowanej jest 5-krotnie większa. W odróżnieniu od poprzednio opisywanych grafów, nie zawiera węzłów terminalnych. Każdy z węzłów zawiera zestaw liter kończących, z zastrzeżeniem, że może to być zbiór pusty. Każde ze słów wchodzących w skład leksykonu jest dzielone na pół w każdej kolejnej pozycji, która znajduje się między dwoma literami słowa. Następnie lewa część podzielonego słowa jest odwracana, oraz między odwróconą lewą część i prawą część umieszcza się symbol specjalny. Generację ruchu opisano w dalszej części pracy. Graf ulega częściowej minimalizacji za pomocą algorytmu, który opisano poniższym listingiem, który

wprost odpowiada zaprezentowanemu w artykule pseudokodowi. Rezultat minimalizacji przykładowego grafu przedstawiono na rysunku 2.4.

```
def add_semi_minimized(word):
    def add_arc(st, ch):
        if ch not in st.edges:
            st.edges[ch] = Node()
        return st.edges[ch]

    def add_final_arc(st, c1, c2):
        st = add_arc(st, c1)
        st.set.append(c2)
        return st

    def force_arc(st, ch, fst):
        if ch in st.edges and st.edges[ch] != fst:
            raise ValueError("an error occurs if an arc from st
            for ch already exists going to any other state")
        elif ch not in st.edges:
            st.edges[ch] = fst

    st = self.root
    for i in range(len(word) - 1, 1, -1):
        st = add_arc(st, word[i])
    st = add_final_arc(st, word[1], word[0])

    st = self.root
    for i in range(len(word) - 2, -1, -1):
        st = add_arc(st, word[i])
    st = add_final_arc(st, "|", word[len(word) - 1])

    for m in range(len(word) - 3, -1, -1):
        forceSt = st
        st = self.root
        for i in range(m, -1, -1):
            st = add_arc(st, word[i])
        st = add_arc(st, "|")
        force_arc(st, word[m + 1], forceSt)
```

Graf GADDAG jest optymalny do wyszukiwania słów w grze Scrabble dzięki możliwości budowania słowa zaczynając od dowolnej pozycji. Generacja słowa odbywa się ponad 2 razy szybciej niż w przypadku grafu DAWG według badań w artykule. Do wygenerowania



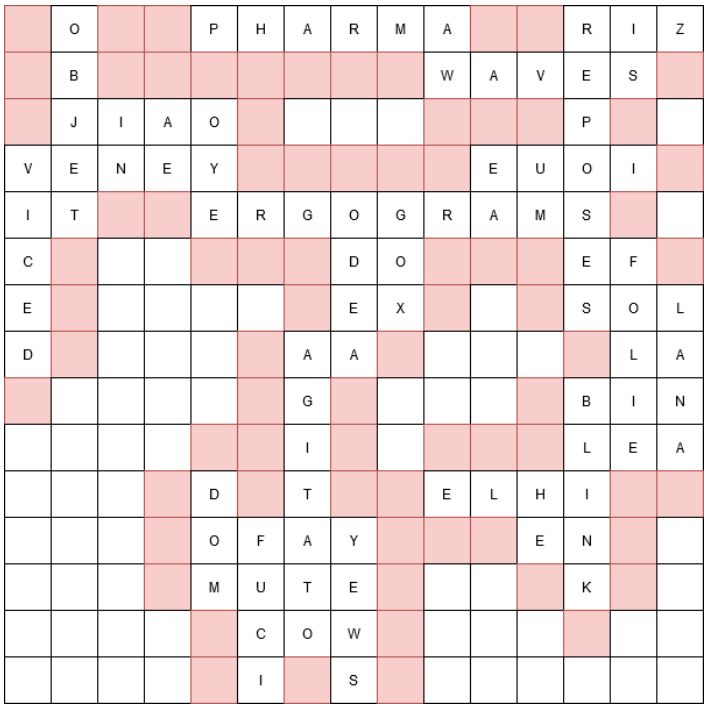
Rysunek 2.4. Częściowo zminimalizowany GADDAG dla leksykonu: ions, jins, cans, can.

ruchów graf przechodzi przez ponad 2 razy mniej krawędzi niż w przypadku grafu DAWG. Ze względu na jakościowe podejście do pracy zdecydowano się na wdrożenie do projektu tego grafu oraz związanych z nim funkcji. W pozostałej części rozdziału szczegółowo zaprezentowano techniki, które odnoszą się do użycia tego grafu.

2.5. Pozycje kandydatów

Pozycje kandydatów to pozycje na planszy, od których należy próbować budować słowa. Budowa słów odbywa się jedynie w kierunku poziomym, o czym więcej wspomniano w dalszym podrozdziale. Znalezienie tych pozycji na planszy jest kluczowe dla zawężania obszaru poszukiwań. Układane słowa na planszy muszą przylegać do sąsiadujących obok słów lub stanowić prefiks i/lub sufiks istniejącego już słowa. W artykule dotyczącym DAWG przytacza się, że pozycjami kandydatów są pozycje, które są sąsiadami zajętych pozycji na planszy zarówno horyzontalnie, jak i wertykalnie 2.5.

"It is easy to decide which squares are potential anchor squares: they are the empty squares that are adjacent (vertically or horizontally) to filled squares. We call these candidate anchor squares the anchors of the row."



Rysunek 2.5. Przykładowa wygenerowana plansza Scrabble za pomocą gotowego symulatora. Pozycje kandydatów wg. autorów artykułu o grafie DAWG.

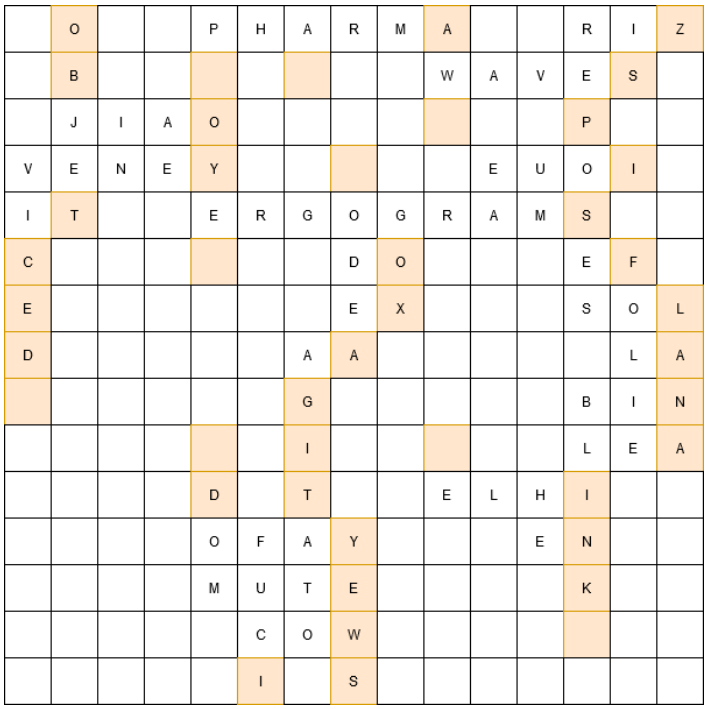
W artykule dotyczącym GADDAG wspomina się o pozycjach kandydatów jako jednej z liter wewnątrz ułożonego słowa, nie zostało opisane, która jest to litera.

"There is no need to generate plays from every other internal anchor square of a sequence of contiguous anchor squares (e.g. the square left or right of the B in Figure 2), since every play from a given anchor square would be generated from the adjacent anchor square either to the right (above) or to the left (below)"

W artykule związanym z GADDAG przytoczony został pseudokod algorytmu generacji ruchu. W wyniku badań przeprowadzonych w ramach tej pracy odkryto, że w wyniku implementacji algorytmu założono, lecz nie opisano w artykule, że zakłada się istnienie pustej pozycji po prawej stronie od pozycji startowej. Poniżej został przytoczony fragment pseudokodu z artykułu.

```
1   GoOn(pos, L, word, rack, NewArc, OldArc):
2       IF pos <= 0 THEN
3           word <- L + word
4           IF L on OldArc and no letter directly left THEN RecordPlay
5           IF NewArc != 0 THEN
6               IF room to the left THEN Gen(pos-1, word, rack, NewArc)
7               NewArc <- NextArc(NewArc, ")
```

```
8               IF NewArc != 0, no letter directly left and room to the right THEN
9                   Gen(1, word, rack, NewArc)
10          ELSE IF pos > 0 THEN
11              word <- word + L
12              IF L on OldArc and no letter directly right THEN RecordPlay
13              IF NewArc != 0 and room to the right THEN
14                  Gen(pos + 1, word, rack, NewArc)
```



Rysunek 2.6. Przykładowa wygenerowana plansza Scrabble za pomocą gotowego symulatora. Pozycje kandydatów dla GADDAG, które są rezultatem badań.

W wyniku badań ustalono, że z powodu braku warunku sprawdzającego istnienie litery na pozycji po prawej stronie od początkowej pozycji pos w linii 4, pozycją kandydata do wyszukiwania w słowie jest skrajnie prawa pozycja, czyli ostatnia litera w słowie. W ramach ustalono też dodatkowo, że pozycją kandydata do wyszukiwania może być tylko taka pozycja nad lub pod słowem, która nie posiada sąsiada bezpośrednio po prawej i po lewej stronie. Dodatkowo ustalono ze względu na leksykon oparty na słowniku SOWPODS, w którym nie istnieją żadne 1-literowe słowa, że pozycja wertykalnego kandydata powinna mieć przynajmniej dwie pozycje na planszy wolne od lewej strony. Rezultat badań przedstawiono na rysunku 2.6.

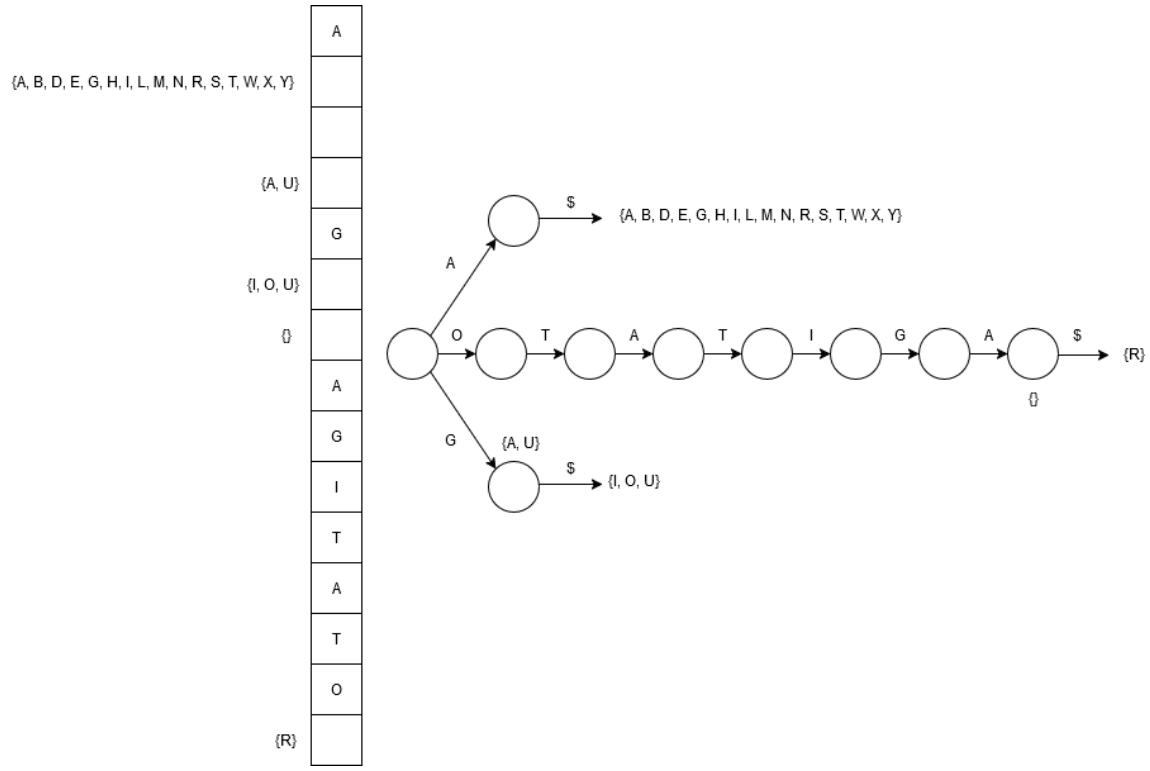
Znalezienie wszystkich pozycji kandydatów dodatkowo jest opisywane poniższym listingiem kodu.

```
def get_candidate_positions(board):
    positions = []
    for y in range(15):
        for x in range(15):
            if (
                board[x][y] and x == 15 or
                board[x][y] and x + 1 < 15 and not board[x + 1][y] or
                (not board[x][y] and y + 1 < 15 and board[x][y + 1] or
                not board[x][y] and y - 1 >= 0 and board[x][y - 1]) and
                (x + 1 < 15 and not board[x + 1][y] or x + 1 >= 15) and
                (
                    (x - 1 >= 0 and not board[x - 1][y]) and
                    (x - 2 >= 0 and not board[x - 2][y] or x - 2 < 0) or
                    (x - 1 >= 0 and not board[x - 1][y] or x - 1 < 0) and
                    (x + 2 < 15 and not board[x + 1][y] or x + 2 >= 15)
                )
            ):
                if (x - 1, y) not in positions:
                    positions.append((x, y))
     return positions
```

2.6. Redukcja problemu do jednego wymiaru

Plansza do gry Scrabble jest dwuwymiarowa i zasady gry umożliwiają układanie słów zarówno horyzontalnie, jak i wertykalnie. Aby umożliwić układanie słów w wybranym jednym kierunku (w literaturze dla prostoty wybranym kierunkiem jest horyzontalny), wykonuje się transponowanie planszy. Aby uprościć układanie słów do jednego rzędu w sposób poprawny, potrzebne są informacje o ułożonych wertykalnie słowach, które rozpoczynają się tuż pod pozycją lub kończą się tuż nad pozycją, na której chcielibyśmy ułożyć literę. W tym celu obliczane są tzw. zestawy skrośny liter, czyli jednoliterowe prefiksy i sufiksy ułożonych wertykalnie słów. Prefiksy i sufiksy są jednoliterowe z powodu podejścia do modyfikacji wyłącznie wybranego jednego rzędu, co oznacza, że wertykalne słowo może zostać przedłużone tylko o jedną literę. Pomocniczo obliczanie zestawów skrośnych przedstawiono na rysunku 2.7.

W artykule o grafie GADDAG zostało wskazane, że obliczanie tych zestawów jest realizowane przez przechodzenie po planszy od dołu do góry w każdym kolejnym rzędzie, jednocześnie przechodząc po grafie do kolejnych węzłów. Po napotkaniu pierwszej wolnej pozycji od ułożonego wertykalnie słowa, zestawem skrośnym dla tej pozycji jest zestaw liter zapisany na tym węźle, zaś zestawem skrośnym dla pozycji tuż przed napotkaniem



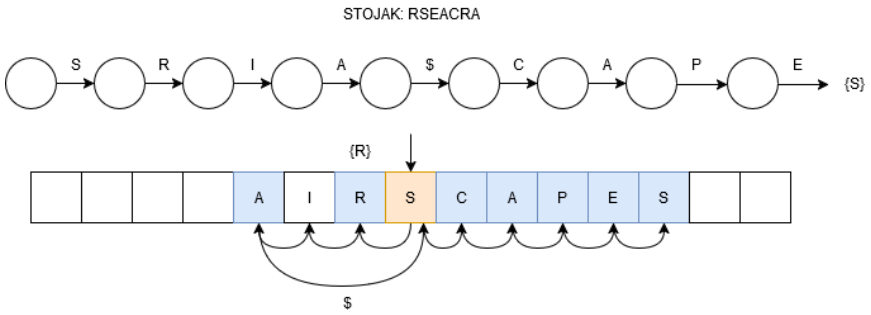
Rysunek 2.7. Przykładowa kolumna z wygenerowanej plansza Scrabble za pomocą gotowego symulatora. Ilustracja przedstawiająca uproszczony graf GADDAG jedynie z tymi węzłami i krawędziami, które zostały wykorzystane do obliczenia zestawów skrośnych.

ostatniej litery słowa jest zestaw liter dla węzła po przejściu przez znak specjalny.

Po obliczeniu i zapisaniu wszystkich zestawów skrośnych podczas układania liter w rzędzie wykonuje się sprawdzenie, czy zgodnie z zestawem skrośnym zadana litera może być ułożona na pozycji.

2.7. Generacja ruchu

Po uproszczeniu problemu gry Scrabble do wybranego jednego rzędu planszy oraz obliczeniu zestawów skrośnych i wyznaczeniu pól kandydatów, dla każdego pola kandydata wykonuje się generację wszystkich możliwych słów, które mogą zostać ułożone ze stojaka gracza i zahaczają o pole kandydata. Generacja rozpoczyna się od wskazanego pola i wchłania kolejne litery na planszy, przesuwając się w rzędzie w lewą stronę. Po napotkaniu pierwszego wolnego pola rozpoczyna się iteracja po wszystkich literach w stojaku, próbując każdą z liter ułożyć na zadanej wolnej pozycji i sprawdzając poprawność słowa. W sytuacji, w której dalsze układanie słowa w lewą stronę nie jest możliwe, sprawdzane jest występowanie symbolu specjalnego. W przypadku wystąpienia następuje dalsza próba generacji słowa w prawą stronę od początkowej pozycji. Przykładowa generacja została przedstawiona na rysunku 2.8. Autor w artykule stwierdza, że podany w artykule



Rysunek 2.8. Przykładowy wiersz z wygenerowanej plansza Scrabble za pomocą gotowego symulatora. Ilustracja przedstawiająca generowanie ruchu "AIRCAPES"ze stojaka "RSEACRA", od pola kandydata "S", z zestawem skrośnym "R"dla pola bezpośrednio po lewej, wraz z podaną ścieżką w grafie, którą generator przeszedł.

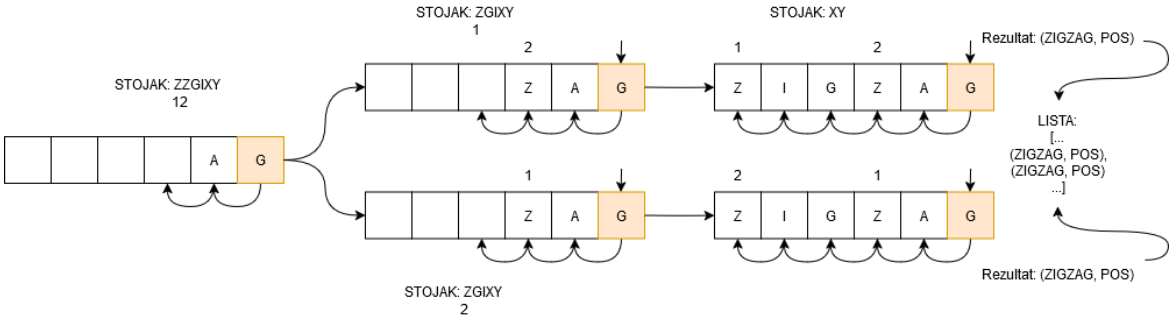
algorytm zapobiega generowaniu duplikatów słów przez przechodzenie w lewą stronę bez możliwości powrotu, a następnie w prawą stronę.

"In order to avoid generating the same move twice, the GADDAG algorithm was implemented with a parameter to prevent leftward movement to the previously used anchor square."

Poniżej przytoczony został fragment pseudokodu z artykułu, który dotyczy generacji ruchu.

```
1  Gen(pos, word, rack, arc):
2      IF a letter, L, is already on this square THEN
3          GoOn(pos, L, word, rack, NextArc(arc, L), arc)
4      ELSE IF letters remain on the rack THEN
5          FOR each letter on the rack, L, allowed on this square
6              GoOn(pos, L, word, rack - L, NextArc(arc, L), arc)
7          IF the rack contains a BLANK THEN
8              FOR each letter the BLANK could be, L, allowed on this square
9                  GoOn(pos, L, word, rack - BLANK, NextArc(arc, L), arc)
```

Algorytm przytoczony przez autora nie rozróżnia tych samych liter znajdujących się w stojaku gracza. Zostało to opisane w linii 5 pseudokodu. Z tego powodu w wyniku generacji listy poprawnych ruchów będą się pojawiały duplikaty słów zawierających więcej niż jedną taką samą literę, ponieważ algorytm wygeneruje nierozróżnialne kombinacje liter składające się na ten wyraz. Dla wyjaśnienia na rysunku 2.9 przedstawiono przykład wspomnianego działania. W ramach badań wykonano autorską modyfikację algorytmu,



Rysunek 2.9. Przykładowa generacja ruchu z duplikatem. Działanie algorytmu jest zgodne z treścią artykułu. W momencie przejścia automatu przez A oraz G, następuje iteracja po pierwszym oraz drugim Z w stojaku zgodnie z linią 5 pseudokodu. W rezultacie algorytm znajduje dwa takie same słowa do ułożenia.

która sprowadza się do zmiany 5 linii w podanym pseudokodzie, by brany był pod uwagę zestaw liter w stojaku bez powtórzeń.

*5 FOR each **unique** letter on the rack, L, allowed on this square*

Dodatkowo, zadany algorytm generuje duplikaty w sytuacji, w której na stojaku gracza znajduje się mydełko. Zgodnie z linią 8 pseudokodu następuje iteracja po wszystkich możliwych literach z alfabetu. Powoduje to duplikację wygenerowanych słów dla liter ze stojaka, dla których nastąpiła już próba ułożenia na zadanym polu zgodnie z liniami 5 oraz 6 pseudokodu. Aby zapobiec duplikacji słów, mydełko powinno przybierać tylko takie litery, które nie znajdują się w danym momencie wykonywania algorytmu na stojaku. Linia 8 pseudokodu została zmodyfikowana i zaprezentowana poniżej.

*8 FOR each letter the BLANK could be **not on rack**, L, allowed on this square*

Ponadto, algorytm nie uwzględnia użycia litery, wywołując funkcję goOn bez rozróżnienia czy wcześniej została użyta jakakolwiek litera ze stojaka. W sytuacji, w której nie została użyta jakakolwiek litera ze stojaka, ruch nie powinien być zapisywany, ponieważ jest to słowo, które wcześniej zostało ułożone na planszy do gry. Z tego względu pseudokod powinien zostać zmodyfikowany następująco.

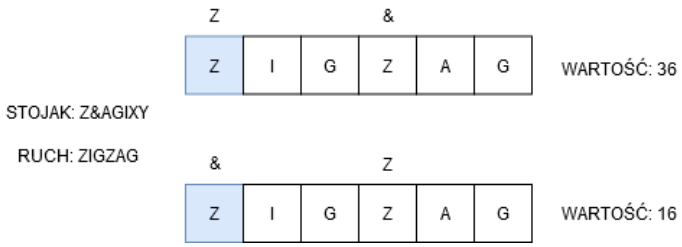
*1 Gen(pos, word, rack, arc**, save=False**):*
...
*3 GoOn(pos, L, word, rack, NextArc(arc, L), arc**, save**)*
...
*6 GoOn(pos, L, word, rack - L, NextArc(arc, L), arc**, True**)*
...

9 *GoOn(pos, L, word, rack - BLANK, NextArc(arc, L), arc,* ***True****)*

Dzięki wprowadzonym zmianom algorytm przestał generować duplikaty słów. Wygenerowana lista wszystkich ruchów zawiera tylko i wyłącznie unikalne wpisy. Każdy wpis składa się z wyrazu, pozycji oraz kierunku. Dodatkowo, algorytm przestał generować wpisy ruchów, które już zostały ułożone na planszy do gry.

2.8. Wartość ruchu

W artykułach odnoszących się do grafów, które mają zastosowanie w generacji ruchu w Scrabble, nie jest wspomniane w żaden sposób o tym, w jaki sposób policzyć jego wartość. Rezultatem działania algorytmu z artykułu z GADDAG jest poprawny wyraz jako string i nie jest wiadome, na której pozycji zostało ułożone mydełko, ponieważ algorytm nie podaje tych informacji w rekurencyjnych wywołaniach funkcji. Ułożone mydełko na planszy ma wartość punktową równą zero.



Rysunek 2.10. Przykładowy ruch do obliczenia wartości oraz stojak z literami. Niebieskie pole jest 3-krotną premią literową. Znakiem specjalnym oznaczono mydełko. Wartość ruchu jest zależna od miejsca, w którym mydełko zostałoby ułożone.

W sytuacji, w której w wyrazie do ułożenia znajdują się dwie takie same litery i jedna z tych liter pojawiła się w wyrazie na skutek użycia mydełka, wówczas kolejność ułożenia tych liter wyrazu może zmienić wartość punktową ruchu. Mydełko w takim przypadku powinno być układane zawsze na pozycji, która nie zawiera premii literowej. Nie musimy rozróżniać premii literowych między taką, która daje współczynnik 2 lub 3, z tego powodu, że premie literowe w tych samych rzędach i tych samych kolumnach planszy do gry mają tę samą wartość. Dla wyjaśnienia przedstawiono rysunek 2.10, w którym występuje wspomniany problem. Wartości słów z rysunku zostały obliczone zgodnie z oficjalnymi zasadami gry [19].

2.9. Rozgrywka między graczami

Przeprowadzenie rozgrywki między graczami w wersji turniejowej Scrabble sprowadza się do zainicjalizowania worka, planszy, graczy wraz ze strategiami oraz ustalenia, który z graczy rozpoczyna rozgrywkę. Gra iterowana jest do momentu, gdy gracz, który ostatni wykonywał ruch, ma pusty stojak i nie ma już liter, którymi można uzupełnić jego stojak z

worka, lub sytuacji, w której żaden z graczy nie może wykonać ruchu. Na każdą iterację składa się skorzystanie z silnika gry, by podać listę możliwych słów, w przypadku braku wpisów zaznaczenie, że gracz nie może wykonać ruchu, w przeciwnym wypadku wybór najlepszego ruchu w ramach strategii, którą gracz stosuje, i aktualizacja planszy, wyniku gracza oraz stojaka. Poniżej podano listing stanowiący reprezentację tego opisu.

```
bag = Bag()
board = Board()
player1, player2 = Player("1", greedy), Player("2", greedy)
player1.rack.letters, player2.rack.letters
    = bag.fill_rack(player1.rack.get()),
        bag.fill_rack(player2.rack.get())
player = player2
while (not (player.rack.empty() and bag.empty()) and
       not (player1.cant_move and player2.cant_move)):
    turn = turn + 1
    if player == player1: player = player2
    else: player = player1
    words_and_positions
        = scrabble.find_all_moves(board.get(), player.rack.get())
    if not words_and_positions:
        player.cant_move = True
         continue
    player.cant_move = False
    play, score, player.rack.letters
        = player.strategy(player, board.get(), words_and_positions)
    board.put_word(*play)
    player.score = player.score + score
    player.rack.letters = bag.fill_rack(player.rack.get())
```

2.10. Podsumowanie

Zaprezentowany szczegółowo graf GADDAG oraz techniki towarzyszące optymalizacji wyszukiwania poprawnych słów są fundamentami dobrze działającego symulatora gry Scrabble. Modyfikacje i rozwinięcia do artykułu o zastosowaniu tego grafu w obszarze wyszukiwania pól kandydatów, generacji ruchu oraz wyznaczania wartości ruchu mogą być cenne dla każdej osoby, która będzie podejmowała się badań w obszarze tej gry. Wyprodukowany w ten sposób symulator stanowi bazę dla dalszej części pracy, w której dokonywana będzie analiza porównawcza strategii, które mogą stosować gracze podczas rozgrywki.

3. Analiza rozgrywek

3.1. Wprowadzenie

Po ukończeniu pracy nad symulatorem oraz strategią zachłanną 4.3, przeprowadzono 25 tysięcy rozgrywek w Scrabble między dwoma graczami. Każdy z nich stosował strategię zachłanną. Z każdej rozgrywki zapisane zostały:

- całkowity czas rozgrywki
- finalny stan planszy
- każdy ruch, który został wykonany w rozgrywce, z wyszczególnieniem
  - stanu stojaka przed wykonaniem ruchu
  - stanu stojaka po wykonaniu ruchu
  - słowa, które zostało ułożone
  - pozycji słowa, które zostało ułożone
  - kierunku, w którym słowo zostało ułożone
  - wartości ruchu

Plik z danymi ze wszystkich rozgrywek zapisano w projekcie w pliku *./data/games.json*. Finalny rozmiar pliku wynosi 294 MB.

Na bazie zebranych w ten sposób danych przeprowadzono analizy, których rezultatem są przedstawione poniżej dane, które ułatwiają zrozumienie natury gry Scrabble.

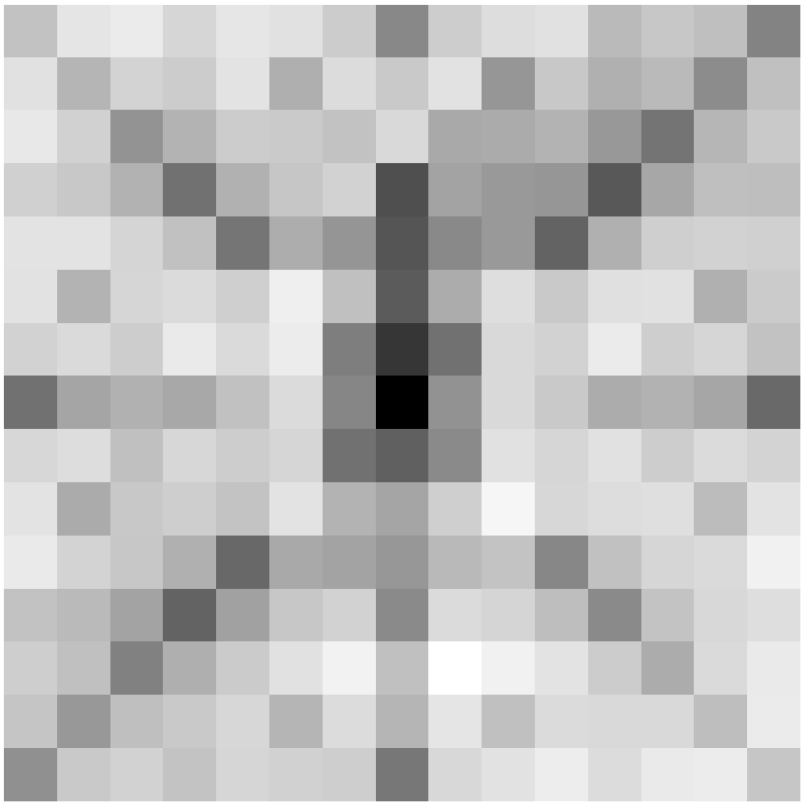
3.2. Zdobycz punktowa jako funkcja pozycji

Na rysunku 3.1 znajduje się dwuwymiarowy widok planszy do gry. Rysunek wskazuje, jak często każde z pól na planszy było wykorzystywane podczas rozgrywek. Pierwszy ruch symulatora był zawsze pionowy, stąd wyższa częstotliwość użycia w środkowej części planszy. Warto zauważyć, w jaki sposób gra rozwija się wokół przekątnych planszy, na których znajdują się podwójne premie słowne. Można także zauważyć, jak gra jest rozciągana po planszy w celu pokrycia pól premiowanych. Dodatkowo, pola trzykrotnej premii słownej częściej są pokrywane w 2. i 3. ćwiartce planszy niż w 1. oraz 4., czyli lewym górnym oraz prawym dolnym narożniku.

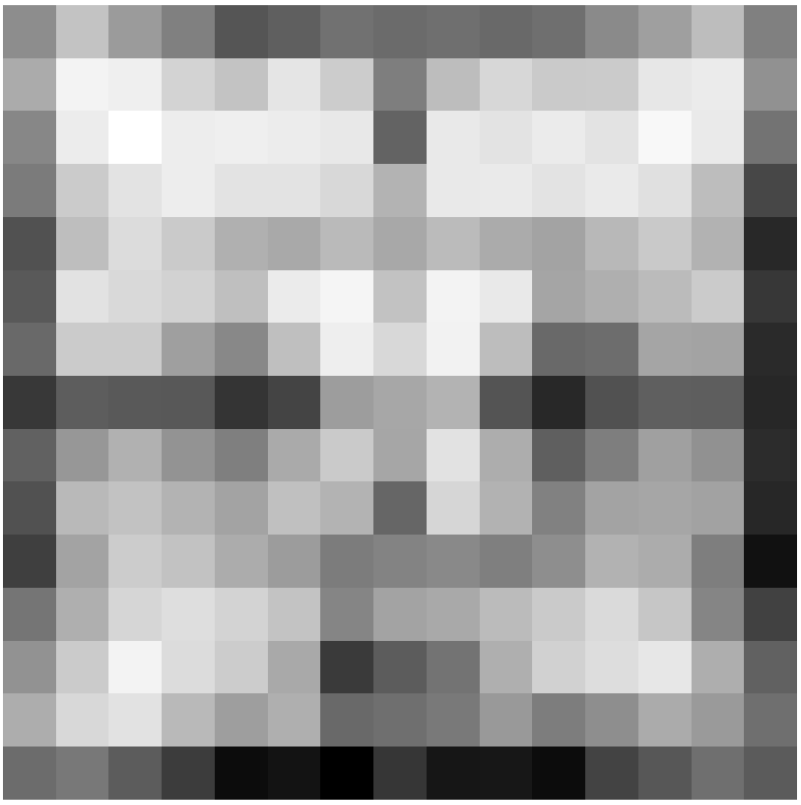
Rysunek 3.2 przedstawia średnią wartość ruchu pokrywającego zadane pole. Rysunek pokazuje jasno, że pola zawierające trzykrotne premie słowne oraz ich wertykalne i horyzontalne okolice dominują w zdobywaniu punktów. Pozostała część planszy jest dość jasna, co wskazuje na to, że dwukrotne premie słowne nie mają tak dużego wpływu na zdobycz punktową.

3.3. Zdobycz punktowa jako funkcja etapu gry

Tabela 3.1 informuje, w jaki sposób zdobycz punktowa zmienia się wraz ze zmieniającym się etapem rozgrywki. W pierwszej turze gracz zdobywa średnio 33,7 punktów, zaś w drugiej turze (czyli pierwszej turze drugiego gracza) średnio 39,8 punktów. Zwiększona



Rysunek 3.1. Częstotliwość użycia pozycji na planszy w rozgrywkach.



Rysunek 3.2. Wartość ruchu w zależności od pozycji na planszy, na którym jest układane.

średnia drugiego ruchu jest spowodowana możliwościami zagrania bingo na otwartej planszy do gry. Zdobycz punktowa jest wysoka przez kilkanaście następnych tur gry, a następnie zaczyna się konsekwentnie zmniejszać wraz ze stopniem zapełnienia planszy do gry.

Tura	Średnia	Bingo %	Bingo	Inne
1	33.7	13.532%	76.6	26.9
2	39.8	20.972%	76.9	30.0
3	37.9	14.724%	76.5	31.3
4	38.6	13.492%	77.1	32.6
5	38.3	11.912%	77.6	33.0
6	38.3	11.252%	77.8	33.3
7	37.6	10.236%	78.2	33.0
8	37.2	9.824%	77.9	32.8
9	36.8	9.160%	78.0	32.7
10	36.7	8.896%	77.9	32.7
11	36.3	8.536%	77.7	32.4
12	36.2	8.305%	78.8	32.4
13	35.7	7.873%	78.0	32.1
14	35.4	7.581%	77.9	32.0
15	35.0	7.145%	77.5	31.7
16	34.6	6.890%	78.1	31.4
17	34.1	6.280%	77.2	31.2
18	32.7	5.816%	77.5	30.0
19	30.2	4.890%	77.5	27.7
20	25.9	3.224%	77.4	24.2
21	21.6	2.117%	79.1	20.3
22	17.4	1.260%	75.9	16.6
23	14.2	0.591%	77.0	13.9
24	12.0	0.337%	72.1	11.8
25	10.2	0.186%	68.1	10.1
26	9.0	0.087%	100.0	8.9
27	7.8	0.274%	81.0	7.6
28	7.0	0.000%	0.0	7.0
29	6.4	0.000%	0.0	6.4
30	6.6	0.000%	0.0	6.6

Tabela 3.1. Wynik jako funkcja tury gry.

3.4. Zdobycz punktowa jako funkcja płytek

Zdobywanie wysokiej liczby punktów w ruchu jest uzależnione od płytek, które gracz posiada. Tabela 3.2 przedstawia średnią wartość ruchu w zależności od płytek, które gracz posiadał oraz użył. Kolumna "Jedna w stojaku"przedstawia średnią wartość ruchu w sytuacji, gdy przed wykonaniem ruchu płytka o wskazanej literze znajdowała się w stojaku.

Zauważamy, że dla liter, których jest najwięcej w worku przy rozpoczęciu gry, takich jak "AEIONRT", oscyluje wokół średniej wartości ruchu ogółem, a litery, których jest najmniej w worku, takie jak mydełko oraz "Q"w sposób znaczący wpływają na wartość ruchu. Jest to intuicyjne, ponieważ litery, których jest najwięcej w worku często występują na stojaku gracza w trakcie rozgrywki, stąd wartość ruchu z ich udziałem jest bliska średniej wartości ruchu w grze ogółem, zaś litery, których jest najmniej w worku, tworzą pojedyncze ruchy w trakcie rozgrywki, stąd ich zróżnicowanie jest większe.

Litera	Jedna w stojaku	Dwie w stojaku	Użycie w ruchu
	53.2	72.5	34.3
A	34.0	30.4	36.1
B	34.3	32.0	35.8
C	33.8	29.0	37.8
D	37.0	31.9	38.1
E	34.7	31.5	36.5
F	35.5	35.7	35.3
G	33.2	29.6	35.1
H	39.4	36.1	37.5
I	31.3	27.0	36.1
J	34.6	-	36.3
K	38.9	-	36.6
L	33.5	29.9	37.9
M	37.2	34.3	37.4
N	34.8	30.1	39.6
O	31.8	28.6	35.4
P	34.8	33.5	36.8
Q	30.4	-	37.0
R	34.5	29.9	39.4
S	42.8	41.5	42.3
T	34.6	30.9	39.0
U	29.6	25.5	36.0
V	31.8	27.7	34.0
W	34.1	30.0	34.2
X	39.2	-	39.0
Y	38.7	33.1	36.4
Z	42.7	-	41.6

Tabela 3.2. Wynik ruchu w zależności od zawartości stojaka oraz ruchu.

Kolejną rzeczą, którą można zauważyć z kolumny "Jedna w stojaku"jest szacowana liczba liter w grze. Wyróżniają się litery "SXYZ", zaś "IOQUV"są pułapkami dla zdobyczy punktowej.

Kolumna "Dwie w stojaku"przedstawia, w jaki sposób przechowywanie dwóch liter wpływa na wartość ruchu. Oczywiście przechowywanie dwóch mydełek bardzo pozytywnie wpływa na wynik rozgrywki. Zauważamy, że przechowywanie dwóch liter "FS"ma

podobny wpływ jak przechowywanie jednej z nich, zaś w pozostałych przypadkach przechowywanie dwóch takich samych liter w sposób znaczący redukuje wartość ruchu.

Kolumna "Użycie w ruchu"wskazuje, w jaki sposób wartość ruchu zależy od płytek wykorzystanych do wykonania ruchu. Zauważamy, że ruchy, w których wykorzystano litery "SXZ"mają wyniki powyżej średniej, zaś ruchy wykorzystujące "VW"mają niższą od średniej wartość punktową. Dodatkowo warto zauważyć różnice między kolumnami "Użycie w ruchu"i "Jedna w stojaku". Litery takie jak "INQRTU"cechują się wysoką wartością ruchu, gdy są w nim używane, zaś przeciętną, gdy jedynie znajdują się w stojaku. Zauważamy, że mydełko w przypadku strategii zachłannej jest używane do wykonywania przeciętnych ruchów.

Warto też zwrócić uwagę na to, w jaki sposób liczba użytych płytek w poprzednim ruchu wpływa na wartość następnego ruchu. W tym celu przygotowano tabelę 3.3, która prezentuje średnią wartość punktową następnego ruchu w zależności od liczby płytek, które pozostały na stojaku po wykonaniu poprzedniego ruchu.

0	1	2	3	4	5	6
40.2	29.1	33.2	33.9	33.3	32.1	29.6

Tabela 3.3. Średnia wartość ruchu od liczby płytek, które zostały w stojaku po poprzednim ruchu w całej grze.

Dodatkowo wygenerowano taką samą tabelę 3.4, ograniczając się jedynie do tur 10. rozgrywek. Warto zauważyć, że gra zmienia się wraz z turami i analizowanie gry oraz zachodzących w niej zmian nie powinno się odbywać z punktu widzenia całości rozgrywki. Końcowy etap jest wyjątkowy i różni się znacząco.

0	1	2	3	4	5	6
39.3	37.1	36.9	36.2	35.4	33.9	30.7

Tabela 3.4. Średnia wartość ruchu od liczby płytek, które zostały w stojaku po poprzednim ruchu w turze 10. gry.

W tabeli 3.5 umieszczono rozkład spółgłosek (w kolejnych kolumnach) oraz samogłosek (w kolejnych wierszach) pozostałych po poprzednim ruchu na stojaku oraz wartości punktowe osiągane w następnym ruchu. Warto zauważyć, pozostawienie w stojaku samych spółgłosek lub samogłosek niekorzystnie wpływa na korzyść punktową w następnej turze. Najlepsze wyniki otrzymywane są po pozostawieniu w stojaku odpowiednio:

- 3 samogłosek,
- 2 samogłosek i 1 spółgłoski
- 1 samogłoski i 2 spółgłosek
- 3 spółgłosek.

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	40.2	29.3	32.9	33.4	32.4	31.5	28.5
1	28.7	34.1	34.0	33.1	31.5	28.2	
2	32.8	34.2	33.1	31.6	29.0		
3	34.1	33.7	32.0	29.4			
4	33.9	32.3	30.1				
5	33.1	30.5					
6	30.7						

Tabela 3.5. Średnia wartość zdobyczy punktowej od liczby spółgłosek i samogłosek pozostałych po poprzednim ruchu w stojaku.

3.5. Zdobycz punktowa jako funkcja słownika

Tabele w tej sekcji przedstawiają, jak zdobycz punktowa jest zależna od słów użytych w grze. Niektóre ze słów są ważniejsze niż inne i z tego powodu częściej używane i/lub wyżej punktowane.

Tabela 3.6 przedstawia, jak wartość punktowa jest rozkładana w zależności od liczby użytych płytek do wykonania ruchu. Najwyższy wiersz w tabeli informuje o użyciu wszystkich 7 płytek, czyli zagraniu bingo. Mimo że bingo było grane jedynie 8,615% ze wszystkich ruchów, odpowiada za 19,968% całej zdobyczy punktowej ze wszystkich rozgrywek. To potwierdza, jak ważne są ruchy z premią bingo.

Tabela informuje, że średnia liczba płytek na ruch wynosi około 3,919 płytek. To wskazuje, że przeciętna gra kończy się po 84 / 3,919 = 21,4 ruchach, podczas gdy nieco mniej ruchów jest potrzebnych, ponieważ na końcu gry jeden z graczy zostaje z przynajmniej jedną literą w stojaku.

Płytki	**Częstotliwość**	**Wynik**
7	8,615%	77,5
6	7,830%	34,7
5	17,850%	33,6
4	23,950%	31,8
3	22,055%	29,3
2	13,692%	23,4
1	6,006%	11,9

Tabela 3.6. Wynik w zależności od liczby użytych płytek.

Tabela 3.7 informuje o tym, w jaki sposób ruch zależy od długości słowa wraz z charakterystyką zdobyczy punktowej. Warto zauważyć, że słowa o długości nie większej niż 5 liter stanowią 74,24% wszystkich zagranych słów, pomimo że słownik użyty w symulacji, czyli SOWPODS, zawiera jedynie 34488 takich słów z około 267 tysięcy wszystkich słów.

Słowa zawierające literę z zestawu "JQXZ"są niezwykle ważne. Przeciętna rozgrywka zawiera około 4 ruchów zawierających wspomniane litery. Tabela 3.8 przedstawia najczę-

Długość	Częstotliwość	Wynik	Na grę
2	9.405%	20.6	21.6
3	19.503%	25.8	56.5
4	23.081%	29.7	77.1
5	22.251%	32.3	84.9
6	12.511%	32.7	48.5
7	7.669%	57.0	51.3
8	5.254%	71.4	38.2
>= 9	0.323%	52.6	1.9

Tabela 3.7. Wynik w zależności od długości słowa.

ściej występujące słowa z literą ze wspomnianego zestawu wraz z ich przeciętną zdobyczą punktową. Warto zauważyć, że słowo "QI"odpowiada za ponad 1% ruchów w grze, zaś wszystkie podane słowa za 2,756% wszystkich słów użytych w grze.

Słowo	Częstotliwość	Wynik
QI	1,028%	25,4
QIS	0,311%	32,1
XI	0,215%	31,7
QAT	0,214%	31,6
EX	0,198%	36,8
ZA	0,174%	37,5
JA	0,162%	31,9
JO	0,159%	28,3
AX	0,151%	37,4
ZO	0,144%	34,2

Tabela 3.8. Dziesięć najczęściej występujących słów "JQXZ".

Ostatnia z tabel przedstawia, jak kolejne fragmenty słownika wpływają na zdobycz punktową. Kolejną cechą jest użyteczność. Niektóre ze słów są używane częściej niż inne. Tabela 3.9 informuje o tym, w jaki sposób zdobycz punktowa z rozgrywek jest rozdzielona między słowami w słowniku SOWPODS. Najczęściej używany był zestaw 7046 słów, który odpowiadał za 64,475% wykonanych ruchów w grze.

Większość najbardziej użytecznych słów to krótkie słowa oraz często używane słowa z premią bingo. Najmniej użyteczne słowa to w większości rzadko występujące słowa, używane do uzyskania premii bingo.

3.6. Wpływ niedeterminizmu na przebieg rozgrywki

Losowość w grze odgrywa znaczną rolę. W zależności od zestawu płytek, które gracz otrzyma będzie mógł układać inne słowa, o wyższej lub niższej premii punktowej. W tym podrozdziale zbadano wpływ losowości na przebieg rozgrywki oraz skuteczność, czyli liczbę zwycięstw.

Użyteczność	Liczba słów	Częstotliwość	Udział punktowy w rozgrywkach
Najlepsze	7046	64.475%	54.416%
90%	14092	77.202%	67.463%
80%	21138	84.157%	75.876%
70%	28184	88.653%	81.945%
60%	35230	91.929%	86.770%
50%	42276	94.446%	90.674%
40%	49322	96.223%	93.545%
30%	56368	97.481%	95.703%
20%	63414	98.740%	97.876%
Najgorsze	70460	100%	100%

Tabela 3.9. Wynik w zależności od częstotliwości słów.

3.6.1. Mydełka

Po przeprowadzeniu analiz wygenerowanych wcześniej rozgrywek zauważono, że otrzymanie dwóch pustych płytek w znacznym stopniu przekłada się na zwycięstwa w rozgrywce. Sytuacja otrzymania przez jednego z graczy dwóch mydełek występuje w 51,64% rozgrywek. We wspomnianych rozgrywkach zwycięski okazał się gracz, który otrzymał dwie puste płytki, zaś jego skuteczność wyniosła 72,73%. Gracz, który otrzyma dwie puste płytki, może układać lepsze, wyżej punktowane słowa.

W tabeli 3.10 przedstawiono rozkład użycia mydełek w zależności od tury. Puste płytki zwykle są używane do ruchów wykorzystujących wszystkie płytki na stojaku. W pierwszych dwóch turach używanych jest 15% wszystkich mydełek, po pierwszych 4 turach 25%, zaś do tury 19. użycie mydełek jest na poziomie powyżej 4%.

3.6.2. Litery

Na podstawie zestawu danych powstałych w rezultacie zasymulowanych rozgrywek przeanalizowano, które litery najczęściej pozostają w stojaku gracza po ukończonej rozgrywce 3.11. W pierwszej kolumnie tabeli znajduje się litera, która znajdowała się w stojaku, zaś w drugiej suma liczby wystąpień. Interesujące jest to, że w 71 rozgrywkach nie został wykorzystany blank przez jednego z graczy. Powstało to w sytuacji, w której jeden z graczy po wykonaniu ruchu uzupełnił płytki, opróżniając worek, zaś drugi wykonał ruch, wykorzystując wszystkie płytki w swoim stojaku. Zaskakujące jest to, że płytki wysokopunktowane, takie jak "JXZ"zazwyczaj były wykorzystywane w rozgrywce, jedynie płytka z literą "Q"częściej pozostawała w stojaku po zakończeniu rozgrywki, co miało miejsce w 5,68% przypadków. W wyniku analizy można zauważyć zasadność stosowania techniki Q-sticking, która polega na wymianie litery Q ze stojaka gracza, tak by przeciwnik pozostał z tą literą do końca rozgrywki. Implementacja i badanie techniki Q-sticking nie zostało jednak objęte w ramach pracy. Najczęściej pozostawianymi w stojaku literami po zakończeniu

Tura	Użycie mydełka
1	7.526%
2	7.642%
3	5.704%
4	5.292%
5	4.900%
6	4.984%
7	4.590%
8	4.723%
9	4.604%
10	4.564%
11	4.536%
12	4.424%
13	4.396%
14	4.358%
15	4.430%
16	4.204%
17	4.268%
18	4.084%
19	3.663%
20	2.828%
21	1.952%
22	1.160%
23	0.496%
24	0.260%
25	0.086%

Tabela 3.10. Użycie mydełka w zależności od tury.

rozgrywki "IUOEALR"to płytki punktowane najniższą wartością 1-punktową. Stosunkowo często w stojaku pozostawały litery 4-punktowe "W"oraz "V", odpowiednio w 8,16% oraz 2,27% wystąpień. Możliwe, w ramach badań warto by było zbadać technikę QVW-sticking lub QV-sticking, która być może mogłaby odnieść lepszy rezultat niż Q-sticking.

Na bazie wykonanych rozgrywek zauważono, że gracze nie mogli wykonać ruchu 1855 razy. Dokonano analizy zawartości stojaków graczy w tych sytuacjach i wyniki zamieszczono w tabeli 3.12. W pierwszej kolumnie tabeli znajduje się litera, która znajdowała się w stojaku gracza, zaś w drugiej kolumnie liczba wystąpień tej litery. Zauważono, że posiadanie liter takich jak "Q"oraz "V"w największym stopniu wpływało na brak możliwości wykonania ruchu, co potwierdza zasadność stosowania techniki Q-sticking lub QV-sticking. Technika oparta na wymianie litery "W"prawdopodobnie nie sprawdziłaby się tak, jak technika oparta na wymianie litery "V"oraz "Q"z powodu 10-krotnie mniejszej liczby wystąpień braku możliwości wykonania ruchu w sytuacji posiadania tej litery. W znacznym stopniu litery takie jak "U", "I"oraz "C"uczestniczyły w sytuacjach braku

Litera	Liczba wystąpień
I	10904
U	8894
O	4157
E	4128
A	2876
L	2597
R	2196
V	2039
G	1905
N	1524
T	1502
Q	1419
C	1150
D	667
W	567
B	550
P	434
F	335
S	326
M	307
J	260
Y	198
H	188
K	102
Z	75
	71
X	67

Tabela 3.11. Litery znajdujące się w stojakach graczy po ukończonej rozgrywce.

możliwości wykonania ruchu, co mogło mieć wpływ na wartościowanie liter w tabeli przedstawionych przez Gordona 4.5.

W tabeli 3.13 przedstawiono wpływ braku możliwości wykonania ruchu na porażkę w rozgrywce. W pierwszej kolumnie znajduje się liczba rozgrywek, w których wystąpiła sytuacja, w której gracz nie mógł wykonać ruchu, zaś w drugiej kolumnie liczba porażek w tych rozgrywkach. W pierwszym wierszu przedstawiono odpowiednio rozgrywki, w których wystąpiła dowolna liczba braków możliwości ruchu przez każdego z graczy. W drugim wierszu zostały opisane rozgrywki, w których tylko jeden z graczy nie mógł wykonać ruchu. Trzeci wiersz tabeli to różnica między pierwszym i drugim wierszem, czyli wszystkimi rozgrywkami z brakiem możliwości ruchu przez któregokolwiek z graczy a rozgrywkami, w których tylko jeden z nich nie mógł wykonać ruchu. Gracz, który jako jedyny w rozgrywce nie mógł wykonać ruchu, przegrał 56,08% rozgrywek. Rozgrywki, w których każdy z graczy

Litera	Liczba wystąpień
U	557
Q	498
V	496
I	375
C	165
G	80
L	78
R	76
J	73
N	66
S	55
W	49
D	46
F	40
B	37
M	37
Z	31
E	30
P	29
T	27
A	21
X	21
K	18
O	17
H	6
Y	3

Tabela 3.12. Litery znajdujące się w stojaku gracza w przypadku braku możliwości wykonaniu ruchu.

nie mógł wykonać ruchu, jedynie w 17,81% przekładały się na porażkę w rozgrywce. Z danych można wywnioskować, że brak możliwości wykonania ruchu może mieć znaczny wpływ na skuteczność gracza, ale tylko w sytuacji, w której przeciwnik będzie korzystał ze strategii, która zapobiegnie lub ograniczy występowanie takich przypadków. Takimi strategiami są opisane w dalszej części pracy strategie stojakowe Ballarda i Gordona.

Następnie wykonano analizę pojedynczych liter występujących w grze, takich jak "JQXZ". Zauważono, że otrzymanie tych liter miało niewielki pozytywny wpływ na rezultat rozgrywki. Wyniki przedstawiono w tabeli 3.14, w której pierwsza kolumna odpowiada jednej z liter "JQXZ", zaś druga kolumna to skuteczność w wygrywaniu rozgrywki przez gracza, który otrzymał daną literę. Warto zauważyć, że otrzymanie przez gracza litery "Z"jest najkorzystniejsze ze wszystkich zbadanych liter i zwiększa szanse na zwycięstwo w rozgrywce o 6,248%.

Liczba rozgrywek	Liczba porażek
1151	523
831	466
320	57

Tabela 3.13. Wpływ braku możliwości ruchu na zwycięstwo w rozgrywce.

Litera otrzymana w trakcie rozgrywki	Procent wygranych rozgrywek
J	53.284%
Q	51.804%
X	54.900%
Z	56.248%

Tabela 3.14. Procent wygranych gier w zależności od otrzymanej litery.

3.7. Podsumowanie

Ten rozdział przedstawił dobry przegląd gry, w którym przedstawione zostało, w jaki sposób tura gry wpływa na zdobycz punktową, w jaki sposób gra zmienia się wraz z kolejnymi turami gry, w jaki sposób zdobycz punktowa zależy od posiadanych płytek, w jaki sposób słowa oraz ich długość wpływają na zdobycz punktową oraz jak pola premiowane wpływają na rozgrywkę.

Warto zauważyć, że analiza rozgrywek, szczególnie zależności posiadanych liter w stojaku od wartości ruchu i zwycięstwa w rozgrywce, jest źródłem inspiracji do badania poddziedziny ewaluacji stojaka w grze [15], w którym badacze aplikują dodatkowe wartościowanie stanu stojaka dla optymalizacji wyboru najlepszego ruchu. Steven Gordon wskazuje w artykule [14], że zbudowanie sieci neuronowej do tego celu mogłoby się okazać ciekawym rozwiązaniem, podając, że tabela przedstawiona w jego artykule mogłaby zostać użyta do tego celu.

4. Ewaluacja stojaka

4.1. Wprowadzenie

Ewaluacja stojaka jest jedną ze strategii wykorzystywanych w procesie decyzyjnym nad optymalnym ruchem w grze Scrabble. Strategia ewaluacji skupia się na literach, które zostają w stojaku gracza po wykonaniu ruchu. Inspiracją do stosowania takich strategii była analiza wpływu liter na wartość ruchu. Jedną z takich analiz wykonano w ramach tej pracy w podrozdziale 3.4. Dodatkowo, strategie ewaluacji stojaka znajdują swoje uzasadnienie na bazie wykonanych analiz liter uniemożliwiających wykonanie ruchu oraz liter pozostałych w stojaku po zakończeniu rozgrywki, które zostały przedstawione w podrozdziale 3.6.2.

Strategia	Liczba wygranych rozgrywek	Średni wynik punktowy
Zachłanna gracz 1.	51,2%	375,7
Zachłanna gracz 2.	48,8%	374,0
Zachłanna (uśrednione)	50%	374,8

Tabela 4.1. Rezultat rozgrywek graczy korzystających ze strategii zachłannej

4.2. Metodyka

W ramach prowadzonych badań wykonywano rozgrywki między dwoma graczami stosującymi różne strategie. Do tego celu wykorzystano symulator rozgrywki opisany w rozdziale 2. Między graczami rozgrywano 1000 rozgrywek. Gracze przyjmowali strategie, które zostały zaimplementowane i znajdują się w katalogu *./strategies* projektu. Dodatkowo zmodyfikowano kod symulacji tak, by w 1000 iteracji raz zaczynał gracz 1, a raz gracz 2, by każdy z nich rozegrał taką samą liczbę rozgrywek zaczynając grę.

4.3. Strategia zachłanna

Strategia zachłanna jest najczęściej używana w badaniach celem porównywania strategii między sobą. Działanie strategii zachłannej sprowadza się do wybrania ruchu, który po ułożeniu słowa uzyska największą korzyść punktową dla gracza w zadanej turze. Strategia skupia się wyłącznie na bieżącej turze, nie bierze pod uwagę sytuacji, w jakiej znajdzie się gracz w następnej turze. W przeciwieństwie do strategii opisywanych w tym rozdziale, podana strategia może skutkować niekorzystnymi rozkładami zawartości stojaka w następnej turze, co będzie powodowało przypadki braku możliwości wykonania ruchu i/lub pozostawienie w stojaku wysokopunktowanych liter po zakończeniu rozgrywki, które będą bonusem punktowym przeciwnika.

Wyniki dwóch graczy grających według strategii zachłannej zostały opisane tabelą 4.1. Każdy z graczy powinien wygrać dokładnie 50% rozgrywek, jednakże ze względu na wysoki niedeterminizm gry Scrabble oraz stosunkowo niewielką liczbę przeprowadzonych

Litera	Wartość
A	0,5
B	-3,5
C	-0,5
D	-1
E	4
F	-3
G	-3,5
H	0,5
I	1,5
J	-2,5
K	-1,5
L	-1,5
M	-0,5
N	0
O	-2,5
P	-1,5
Q	-11,5
R	1
S	7,5
T	-1
U	-4,5
V	-6,5
W	-4
X	3,5
Y	-2,5
Z	3
Mydełko	24,5

Tabela 4.2. Tabela wartości liter pozostających w stojaku z artykułu Ballarda

rozgrywek (1000), liczba zwycięstw gracza 1. wyniosła 51,2%, zaś gracza 2. 48,8%. Średnia liczba uzyskanych punktów przez graczy nie różniła się znacząco i wyniosła około 374,8 punktów.

4.4. Strategia Ballarda

Badacze na podstawie analiz wpływu liter na stojaku na wynik punktowy zaczęli się zastanawiać nad korzyścią punktową płynącą z wybierania nieco mniej korzystnych ruchów w danej turze, ale pozostawiających dobre litery na stojaku do wykonania ruchu w następnej turze. Początkowo temat został rozwinięty przez Ballard w artykule [15], a następnie podejmowany przez kolejnych badaczy.

Metoda Ballarda, która jest heurystyką, sprowadza się do sumacji wartości liter pozostających na stojaku po wykonaniu ruchu zgodnie z tabelą 4.2.

Strategia	Liczba wygranych rozgrywek	Średni wynik punktowy
Ballarda gracz 1.	53,5%	387,9
Zachłanna gracz 2.	46,5%	382,5

Tabela 4.3. Wyniki własnej symulacji dla strategii literowej Ballarda oraz zachłannej

Strategia	Liczba wygranych rozgrywek	Średni wynik punktowy
Gordona gracz 1.	60,3%	393,6
Zachłanna gracz 2.	39,7%	375,9

Tabela 4.4. Wyniki własnej symulacji dla strategii literowej Gordona oraz zachłannej

Wyniki przeprowadzonych rozgrywek przedstawiono w tabeli 4.3. Heurystyka zaproponowana przez Ballarda uzyskała o 3,5% wyższą skuteczność w wygrywaniu rozgrywek, a także zdobywała około 10 punktów więcej niż strategie zachłanne w rozgrywkach między sobą. Zauważono także większą średnią liczbę punktów zdobywaną przez gracza zachłannego. Mogła ona wynikać z odpuszczania przez gracza posługującego się strategią Ballarda korzystnych ruchów zajmujących premiowane pola, które w następnej turze były wykorzystywane przez gracza stosującego strategię zachłanną.

Implementacja wspomnianej strategii znajduje się w pliku *./strategies/ballard.py* projektu.

4.5. Strategie Gordona

Gordon w swoich badaniach, które przedstawił w artykule [8], zaproponował rozwinięcie dla heurystyki Ballarda oraz jedną własną nową, która opiera się na balansie samogłosek oraz spółgłosek. Zaproponowane heurystyki opierają na statystycznej analizie stanu stojaka oraz wpływie liter na korzyść punktową.

4.5.1. Balans liter

Gordon zauważył, że heurystyka Ballarda w taki sam sposób oblicza wartość stojaka, w którym się znajdują duplikaty liter. Zgodnie z regułami Ballarda, stojak z literami IISSS ma wartość punktową równą 25,5 mimo, że nic sensownego z takiej kombinacji liter nie można by było ułożyć. W swoim badaniu zaproponował nową tabelę 4.5 z wartościami liter, która uwzględnia nowe wartościowanie duplikatów.

Wyniki przeprowadzonych rozgrywek z zastosowaniem heurystyki literowej Gordona znajdują się w tabeli 4.4. Zauważalny jest znaczny wzrost wygranych rozgrywek w stosunku do heurystyki Ballarda. Heurystyka wygrała 60,3% rozgrywek przeciwko strategii zachłannej, osiągając wynik na poziomie 393,6 punktów. Zauważalny jest też delikatny spadek finalnej korzyści punktowej strategii zachłannej.

Implementacja wspomnianej strategii znajduje się w pliku *./strategies/gordon_h2.py* projektu.

Litera	Wartość (pierwsza w stojaku)	Wartość (każda kolejna)
A	1	-3
B	-3,5	-3
C	-0,5	-3,5
D	0	-2,5
E	4	-2,5
F	-2	-2
G	-2	-2,5
H	0,5	-3,5
I	-0,5	-4
J	-3	
K	-2,5	
L	-1	-2
M	-1	-2
N	0,5	-2,5
O	-1,5	
P	-1,5	-2,5
Q	-11,5	
R	1,5	-3,5
S	7,5	-4
T	0	-2,5
U	-3	-3
V	-5,5	-3,5
W	-4	-4,5
X	3,5	
Y	-2	-4,5
Z	2	
Mydełko	24,5	-15

Tabela 4.5. Tabela wartości liter pozostających w stojaku z artykułu Gordona

4.5.2. Balans samogłosek i spółgłosek

Dodatkowo Gordon zauważył, że do optymalnego budowania słów wymagane jest zachowanie balansu spółgłosek i samogłosek. Wartości w tabeli 4.6 wynikają ze statystycznej analizy słów, w której stwierdzono, że słowa składają się w 60% ze spółgłosek. Zachowanie odpowiedniego balansu spółgłosek i samogłosek powinno pozwolić na układanie dłuższych słów.

Gordon w artykule osiągnął rezultat na poziomie 63,4% wygranych rozgrywek. Za pomocą symulatora nie udało się odtworzyć rezultatów osiągniętych w artykule. Wyniki symulacji dla kombinacji strategii literowej oraz balansu samogłosek i spółgłosek znajdują się w tabeli 4.7.

Implementacja wspomnianej strategii znajduje się w pliku *./strategies/gordon_h3.py* projektu.

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	-1	-2	-3	-4	-5
1	-1	1	1	0	-1	-2	
2	-2	0	2	2	1		
3	-3	-1	1	3			
4	-4	-2	0				
5	-5	-3					
6	-6						

Tabela 4.6. Tabela wartości balansu spółgłosek i samogłosek z artykułu Gordona

Strategia	**Liczba wygranych rozgrywek**	**Średni wynik punktowy**
Gordona gracz 1.	60,7%	396,4
Zachłanna gracz 2.	39,3%	377,9

Tabela 4.7. Wyniki własnej symulacji dla połączonej strategii literowej oraz balansu spółgłosek i samogłosek Gordona oraz zachłannej.

4.6. Podsumowanie

Przeprowadzone badania wskazują, że ewaluacja stojaka może być użyteczna w poszukiwaniu najlepszej strategii rozgrywki w grze. W rozgrywkach przeciwko strategii zachłannej najlepsza z metod uzyskała wynik na poziomie ponad 60%. Jest to znaczący wzrost, szczególnie biorąc pod uwagę fakt, że Scrabble jest grą, w której w etapie startowym każdy z graczy otrzymuje płytki w sposób losowy, co ma znaczny wpływ na przebieg rozgrywki.

5. Końcowy etap rozgrywki

5.1. Wprowadzenie

Rozpoczęciem końcowego etapu rozgrywki jest pierwsza tura, w której worek do gry jest pusty. Jest to etap obowiązujący do końca gry. Charakteryzuje się zmianą determinizmu gry, ponieważ od wskazanej tury gra staje się deterministyczna. Z powodu braku płytek w worku, jednocześnie znając wszystkie płytki uczestniczące w rozgrywce, możliwe jest obliczenie zawartości stojaka przeciwnika. Wykonywane jest to przez odjęcie płytek na planszy oraz gracza od sumy wszystkich płytek biorących udział w grze.

Wynika z tego korzyść w postaci możliwości stosowania popularnych algorytmów w grach deterministycznych, tj. szachy lub kółko i krzyżyk. W tym rozdziale zaprezentowane zostały niektóre z nich, tj. drzewo minimax i jego wariancja z założeniem gry przeciwnika w sposób zachłanny.

5.2. Szybkie zakończenie

Strategie opisywane w rozdziale 4 zawierały założenie, że gra jest niedeterministyczna w całej długości rozgrywki. Nie skupiano się na tym w jaki sposób grę najkorzystniej by było kończyć. Gracz, który zakończy rozgrywkę jako pierwszy uzyskuje korzyść w postaci podwojonej wartości liter na stojaku przeciwnika. Z tego powodu w ramach prowadzonych badań zmodyfikowano strategię Gordona opisywaną w podrozdziale 4.5.2 według własnego pomysłu, który polega na tym by w końcowej rozgrywce nałożyć modyfikację w postaci wyboru słów kończących rozgrywkę zamiast najkorzystniejszych wg obliczeń z tabel 4.5 oraz 4.6. W rezultacie otrzymano następujące wyniki zaprezentowane w tabeli 5.1.

Wybór słowa kończącego rozgrywkę oznacza wybór takiego ruchu, który wykorzysta wszystkie płytki, które pozostały na stojaku gracza. Po wykorzystaniu wszystkich płytek, przy założeniu pustego worka, gra natychmiast kończy się.

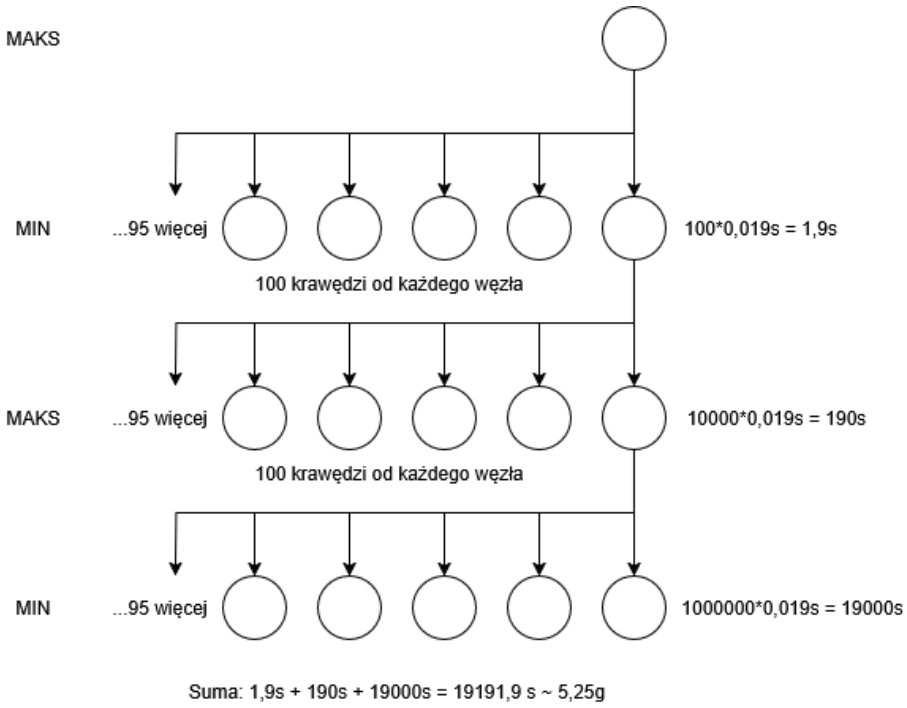
Strategia	Liczba wygranych rozgrywek	Średni wynik punktowy
H3 + SZ gracz 1.	63,0%	393,9
Zachłanna gracz 2.	37,0%	367,5

Tabela 5.1. Wyniki własnej symulacji dla połączonej strategii literowej oraz balansu spółgłosek i samogłosek Gordona z dodatkiem szybkiego zakończenia oraz zachłannej.

Warto zauważyć, że uzyskano rezultat o 2,7% lepszy od wariantu strategii Gordona bez szybkiego zakończenia. Gracz stosujący tę strategię zdobył, w granicach błędu statystycznego, więcej o 0,3 punktu, zaś liczba punktów przeciwnika spadła o 8,4 punktu. Jest to zgodne z przewidywaniami, ponieważ stosując strategię szybkiego zakończenia, przeciwnik traci turę lub dwie, które zwykle miał na końcu rozgrywki.

Implementacja wspomnianej strategii znajduje się w pliku *./strategies/asawicki_h3.py* projektu.

5.3. Drzewo Minimax



Rysunek 5.1. Graf reprezentujący drzewo Minimax przy założeniu, że gracz ma do wyboru 100 ruchów w każdej turze.

Drzewo Minimax w przypadku gry Scrabble sprowadza się do grafu, w którym każdy z węzłów odpowiada jednemu z stanów gry, złożonego z:

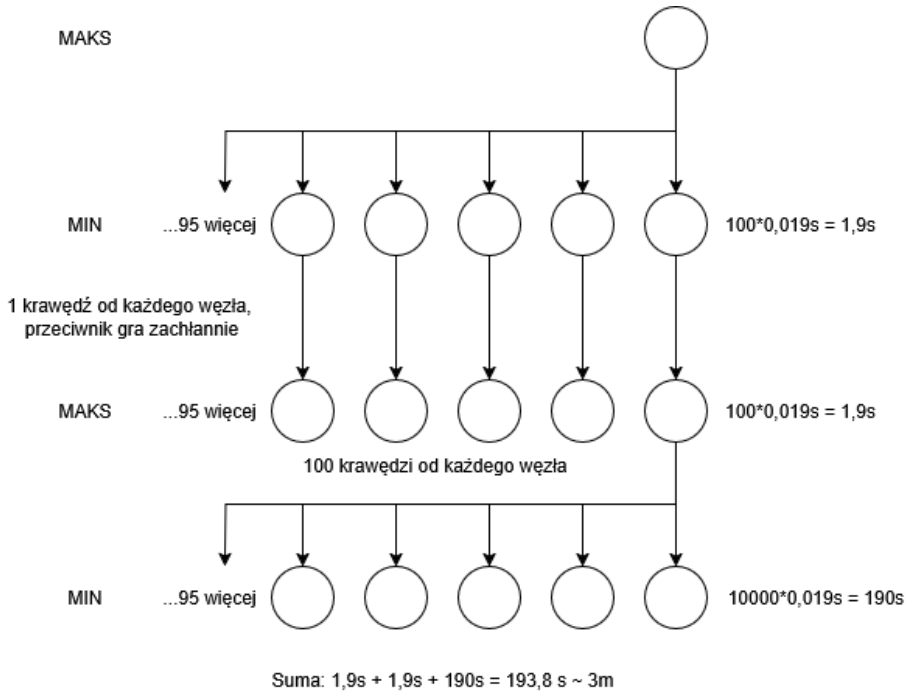
- planszy,
- graczy biorących udział w rozgrywce,
- stojaków graczy,
- informacji, który z graczy wykonuje ruch.

Każda z krawędzi grafu reprezentuje jeden z ruchów, który gracz mógłby wykonać w stanie, z którego krawędź wychodzi. Graf jest skierowany i acykliczny, dzięki czemu sytuacja zapętlenia jest wykluczona. Drzewo maksymalnie może mieć wysokość równą 14, w sytuacji gdy w węźle startowym każdy z graczy ma pełny stojak (7 płytek), zaś w każdym ruchu każdy z nich dokłada do planszy jedną literkę.

Celem produkcji drzewami Minimax jest znalezienie optymalnego ruchu dla zadanego stanu gry. Zakłada się, że przeciwnik będzie również grał optymalnie, w przeciwieństwie do sytuacji opisywanej w następnym rozdziale 5.4.

Głównym problemem tworzenia drzewa Minimax dla gry Scrabble jest czas tworzenia. Dla każdego węzła drzewa Minimax, które jest pozbawione wychodzących krawędzi,

przeprowadzane jest wyszukiwanie na planszy wszystkich możliwych ruchów, które gracz może wykonać przy zadanym stanie stojaka. Ta operacja pochłania większość czasu pracy programu. Średni czas wykonywania tej operacji wynosi iloraz z średniego czasu rozgrywki przez średnią liczbę tur w rozgrywce, co wynosi 0,415 s / 21,4 = 0,019 s. Przeciętnie w trakcie każdej tury gracz ma do wyboru ok. 100 ruchów. Drzewo rozrasta się wraz z każdym poziomem głębi 100-krotnie, zaś liczbę liści w grafie na każdym poziomie opisuje się liczbą 100^n, gdzie n to poziom głębokości drzewa. W sytuacji, gdyby ograniczyć się do symulowania jedynie trzech następnych ruchów graczy, wymagany czas na utworzenie drzewa wynosiłby 100*100*100*0,19 s, czyli około 5,25 godziny. Gracz w rozgrywce turniejowej dysponuje 30 sekundami, więc to znacząco wykracza poza możliwości obliczeniowe komputerów. Dla przejrzystości dodano ilustrację 5.1 reprezentującą drzewo Minimax.



Rysunek 5.2. Graf reprezentujący drzewo Minimax przy założeniu, że gracz ma do wyboru 100 ruchów w każdej turze oraz przeciwnik gra zachłannie.

Implementacja wspomnianej strategii znajduje się w pliku *./strategies/asawicki_h4.py* projektu.

5.4. Strategia zachłanna przeciwnika

Dla ułatwienia można przyjąć założenie, że przeciwnik grałby w sposób zachłanny. Wówczas drzewo minimax dla takiego wariantu prezentowałoby się tak, jak na ilustracji 5.2. Dla osiągnięcia jakiejkolwiek korzyści należałoby symulować rozgrywkę w przód przynajmniej dla 4 następnych ruchów graczy. Wówczas wymagany czas na utworzenie tak skróconego drzewa wynosiłby 100*100*0,19s, czyli około 190 sekund. To także wykracza

poza możliwości turniejowe, w których na turę przypada 30 sekund. Z powodu braku użyteczności wspomnianego algorytmu, na bazie ograniczeń czasowych zrezygnowano z implementacji tego rozwiązania oraz dalszych badań.

5.5. Podsumowanie

Na bazie strategii szybkiego końca rozgrywki, której wynik opierał się na 63% zwycięstw przeciwko strategii zachłannej, można stwierdzić, że zastosowanie odpowiedniego algorytmu w końcowym etapie rozgrywki może mieć kluczowe znaczenie dla efektywności, czyli liczby zwycięstw w grze. Jednakże stwierdzono także, że algorytmy takie jak Minimax i jego wariancje nie mają obecnie zastosowania w rozgrywce turniejowej z powodu złożoności obliczeniowej, która przekłada się na czas wykonywania przekraczający ustalone 30 sekund na turę dla gracza.

6. Podsumowanie

W pracy ujęto najważniejsze aspekty gry Scrabble, zaczynając od historycznego wstępu do gier i sztucznej inteligencji, na analizie ostatniego etapu rozgrywki kończąc. Największym, a jednocześnie autorskim, wkładem w tematykę gry była z części inżynieryjnej:

- implementacja symulatora do gry Scrabble w języku Python
- modyfikacja zapobiegająca duplikatom i implementacja zaproponowanego przez Gordona algorytmu grafowego
- algorytm i implementacja znajdowania pozycji kotwiczących na planszy
- algorytm i implementacja obliczania zestawów skrośnych na planszy
- przeprowadzenie kilkudziesięciu tysięcy symulacji rozgrywek gry

W części badawczej pracy udało się:

- przeanalizować rozgrywki skupiając się na analizie planszy do gry
- przeprowadzić analizę gry z wyszczególnieniem używanych przez graczy płytek
- sprawdzić w jakim stopniu oraz w jaki sposób wykorzystywany jest słownik do gry
- przeprowadzić badania na algorytmach ewaluacji stojaka
- zbadać algorytmy służące do gier deterministycznych, które miałyby zastosowanie do końcowego etapu gry Scrabble

Niektóre obszary mogły zostać zbadane bardziej obszernie, jednakże z powodu ograniczeń nie udało się zrealizować wszystkiego.

6.1. Ograniczenia

Jednym z ograniczeń był sprzęt oraz dobór technologii. Pierwotnie wybrano język Python, aby implementacja w sieci była łatwiej dostępna dla odbiorców, testerów i badaczy. Python jest językiem, w którym wykorzystywany jest interpreter, kod źródłowy nie jest kompilowany przed wykonywaniem, stąd działanie programu mogło być wolniejsze niż w przypadku języka tj. C++. Dodatkowo badania były realizowane na sprzęcie, który w obecnej chwili nie charakteryzuje się wyjątkowo wysoką mocą obliczeniową. Cała praca, z charakteru magisterskiego, nie skupiała się na wyprodukowaniu symulatora rozgrywki, który zawierałby w sobie optymalizacje takie jak wykorzystanie wielu wątków procesora do przyspieszenia najbardziej czasochłonnych elementów kodu, np. wyszukiwania wszystkich poprawnych ruchów, które gracz może wykonać. Mogło to przyciąć skrzydła badaniom tj. opisywany wariant drzewa Minimax z podrozdziału 5.4.

Kolejnym ograniczeniem był czas, który w dużej mierze był konsumowany nie tylko na wykonywanie badań na danych, ale także na implementację silnika gry, a także strategii rozgrywki. **Każde zaprezentowane w pracy badanie zostało wykonane samodzielnie, zaś wyniki są rezultatem badań własnych na symulacjach przeprowadzonych na autorsko zaimplementowanym symulatorze rozgrywki.** Dodatkowo wyjątkowo czasochłonne

okazały się symulacje rozgrywki, które były wykonywane na urządzeniu przez kilkadziesiąt godzin w ramach badań.

Innym ograniczeniem był dostęp do literatury. Obecnie Scrabble często są tematem prac magisterskich, jednakże najbardziej wartościowe artykuły naukowe rzucające światło na podstawowe aspekty nie są dostępne w zasobach internetowych. Przykładem jest praca Ballarda [15], która została ujęta w mojej pracy jedynie dlatego, ponieważ została zacytowana i opisana w pracy Gordona [14], która jest dostępna w sieci. Jednakże, o czym warto wspomnieć, w pracy Ballarda została opisana strategia wymiany płytek, do której nie miałem dostępu, a chętnie zostałaby zaimplementowana i zbadana. Dodatkowym podcięciem skrzydeł był brak dostępu do implementacji symulatora MAVEN Sheppard [2], która była przez jakiś czas dostępna w Internecie, ale w momencie podejmowania się pisania pracy już nie była.

Ostatnim ograniczeniem była ambicja autora, aby nie prezentować w pracy implementacji, analiz gry oraz strategii graczy gorszych niż możliwe do znalezienia w zasobach internetowych. Każda ze strategii została zaimplementowana w symulatorze oraz zbadano jej wyniki, stąd badanie nie ograniczało się do przepisania wyniku z pracy innego badacza, lecz przeprowadzenia badania rzetelnie i samodzielnie. Z tego względu objętość pracy mogła okazać się mniejsza, ponieważ nie można było przeprowadzić takiej samej liczby badań jak w sytuacji, w której jedynie powoływałoby się na wyniki z pracy innego badacza.

6.2. Wnioski

W inżynieryjnej części pracy zauważono nieścisłości w zaproponowanym przez Gordona grafie GADDAG służącym do wyszukiwania wszystkich możliwych ruchów do wykonania w danej turze przez gracza. Zastosowane autorskie modyfikacje algorytmu oraz opis działania z pewnością rzucają sporo światła na ten obszar, jednakże możliwe, że silnik do gry mógłby opierać się na czymś lepszym, a algorytm jest nieco przestarzały. Możliwe, że w przyszłości zostanie wymyślony lepszy algorytm służący do generacji ruchu.

W ramach analiz rozgrywek wykonanych na symulatorze warto zwrócić uwagę na powiązanie między strategiami i statystyką. Każda z heurystycznych strategii dla etapu niedeterministycznego miała swoje fundamenty w tych analizach, tj. strategia opisana w podrozdziale 4.2 opierała się na analizach przeprowadzonych w podrozdziale 3.4. Można stąd wywnioskować, że w celu poszukiwania optymalnej strategii ewaluacji stojaka należałoby odwoływać się do statystyki.

Kolejnym wnioskiem jest wpływ elementu losowości w grze Scrabble. Nawet jeśli gracz wykorzystywałby optymalny algorytm, ale rozkład losowy płytek w stojaku gracza, przeciwnika oraz w worku byłby bardzo korzystny dla przeciwnika, gracz wciąż miałby niewielkie szanse na zwycięstwo. Warto zauważyć, że najlepszy uzyskany w ramach badań algorytm był w stanie wygrać 63% rozgrywek przeciwko strategii zachłannej, co nie jest

wysokim wynikiem biorąc pod uwagę to, że w grach takich jak szachy, arcymistrzowie są w stanie wygrywać 99% rozgrywek.

Innym wnioskiem jest brak zastosowania algorytmów tj. Minimax dla gry Scrabble, przynajmniej w obecnym czasie. Złożoność obliczeniowa tych algorytmów jest zbyt duża, zaś operacje wykonywane w programie takie jak kopiowanie stanu rozgrywki, przeszukiwanie dziesiątek tysięcy plansz do gry w poszukiwaniu listy wszystkich możliwych ruchów do wykonania są zbyt kosztowne czasowo, skutkując brakiem użyteczności w rzeczywistości gry Scrabble, czyli wariancie turniejowym dla dwóch graczy.

6.3. Możliwości w przyszłości

Uważa się, że komputer kwantowy jest około 100 razy szybszy niż tradycyjny komputer [20]. Zastosowanie algorytmów do etapu końcowego, takich jak Minimax, miałoby zastosowanie na takim urządzeniu. W przypadku strategii opisanej w podrozdziale 5.4, która na moim urządzeniu nie mogła zostać zasymulowana w czasie krótszym niż 30 sekund, można szacować, że na komputerze kwantowym zostałaby zrealizowana w ciągu 2 sekund. Możliwe, że w przyszłości, gdy każdy będzie posiadał swój osobisty komputer kwantowy, wydajne wykonywanie symulacji z użyciem takich algorytmów będzie możliwe.

6.4. Dalsze badania

Możliwe jest dopracowanie heurystyk literowych Ballarda i Gordona ze względu na turę lub etap gry. Autorzy zaproponowali wartości liter dla całej gry, jednakże w wyniku analizy rozgrywek można zauważyć, że wartości powinny być różne w zależności od tury, w której gracz wykonuje ruch. Dzięki wygenerowaniu tabel dla każdej tury rozgrywki można uzyskać strategię o lepszych rezultatach.

Warto także skupić się na poszukiwaniu optymalnej strategii dla końca rozgrywki w Scrabble, która mogłaby być stosowana w wersji turniejowej gry. Typowe algorytmy takie jak Minimax nie mogą być w tym celu wykorzystane, lecz być może pewien wariant takiego drzewa mógłby mieścić się w ramach czasowych.

Problem z brakiem zastosowania algorytmu Minimax mógłby zostać rozwiązany także przez znalezienie lepszego, optymalnego algorytmu do wyszukiwania wszystkich poprawnych ruchów. Już kilkukrotne przyspieszenie działania takiego algorytmu mogłoby skutkować mieszczeniem działania algorytmów drzewowych w wyznaczonym czasie na turę.

Mogłoby być ciekawe podjęcie próby wygenerowania wszystkich poprawnych plansz do gry Scrabble jako węzłów grafu wraz z listą krawędzi, po których można przejść posiadając zadany stan stojaka do gry. Spowodowałoby to wyeliminowanie problemu wyszukiwania wszystkich poprawnych ruchów, ponieważ już nie trzeba byłoby szukać - wszystkie znajdowałyby się na liście. Symulacje z takim zestawem danych stałyby się

wyjątkowo wydajne i nawet przy obecnej technologii można byłoby wykonywać nawet złożone algorytmy takie jak Minimax.

Bibliografia

[1] P. A. Piccione, "GAMING WITH THE GODS: The Game of Senet and Ancient Egyptian Religious Beliefs", 1990. adr.: https://piccionep.people.charleston.edu/senet_web.html.

[2] B. Sheppard, "Towards perfect play of Scrabble", 2002. adr.: https://api.semanticscholar.org/CorpusID:152336915.

[3] P. J. Turcan, "A competitive Scrabble program", *SIGART Newsletter 80*, 1982.

[4] S. C. Shapiro, "Scrabble crossword game-playing programs", *SIGART Newsletter 80*, 1982.

[5] I. Nation i A. Coxhead, *Measuring Native-Speaker Vocabulary Size*. John Benjamins, 2021. adr.: https://www.jbe-platform.com/content/books/9789027260291.

[6] C. Carrol i N. Ballard, "Effective Study Habits", *Medleys*, 1991.

[7] N. Kobzeva, "Scrabble as a Tool for Engineering Students' Critical Thinking Skills Development", *Procedia - Social and Behavioral Sciences*, t. 182, s. 369–374, 2015, 4th WORLD CONFERENCE on EDUCATIONAL TECHNOLOGY RESEARCHES (WCETR-2014), ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2015.04.791. adr.: https://www.sciencedirect.com/science/article/pii/S1877042815030669.

[8] M. Klijn, "A perfect information Scrabble game", 2017. adr.: https://theses.liacs.nl/pdf/MichelKlijn.pdf.

[9] F. D. Maria i A. Strade, "An Artificial Intelligence that plays for Competitive Scrabble", w *PAI*, 2012. adr.: https://api.semanticscholar.org/CorpusID:2032395.

[10] J. Abraham, C. Pollett, P. Heller, R. Chun, C. Pollett i G. Rajeev, "A Scrabble Artificial Intelligence Game A Project Presented to The Faculty of the Department of Computer Science San Jose State University In Partial Fulfillment of the Requirements for the Degree Master of Science by Priyatha", 2017. adr.: https://api.semanticscholar.org/CorpusID:9169236.

[11] F. Connolly i D. Gren, "Smart Scrabble playing - strategies and their impact", 2012. adr.: https://api.semanticscholar.org/CorpusID:110728193.

[12] A. W. Appel i G. J. Jacobson, "The world's fastest Scrabble program", *Commun. ACM*, t. 31, nr. 5, s. 572–578, maj 1988, ISSN: 0001-0782. DOI: 10.1145/42411.42420. adr.: https://doi.org/10.1145/42411.42420.

[13] S. A. Gordon, "A faster Scrabble move generation algorithm", *Softw. Pract. Exper.*, t. 24, nr. 2, s. 219–232, lut. 1994, ISSN: 0038-0644. DOI: 10.1002/spe.4380240205. adr.: https://doi.org/10.1002/spe.4380240205.

[14] S. Gordon, "A COMPARISON BETWEEN PROBABILISTIC SEARCH AND WEIGHTED HEURISTICS IN A GAME WITH INCOMPLETE INFORMATION", *AAAI*, 1993.

[15] N. Ballard, "Renaissance of Scrabble theory 2", *Games Medleys*, 1992.

[16] J. Nguyen, *Scrabble*, https://github.com/joineynguyen/Scrabble, 2020.

[17] C. Browne, *scrabble-trie*, https://github.com/calbro7/scrabble-trie, 2021.

[18] Keegan, *Scrabble*, https://github.com/kmp3325/Scrabble, 2014.

[19] HASBRO, “Game Rules”, 2003. adr.: https://www.hasbro.com/common/instruct/Scrabble_(2003).pdf.

[20] R. M. Abdullah, R. Basher, A. A. Alwan i A. Z. Abualkishik, “Quantum Computers for Optimization the Performance”, *Procedia Computer Science*, t. 160, s. 54–60, 2019, The 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2019) / The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019) / Affiliated Workshops, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.09.443. adr.: https://www.sciencedirect.com/science/article/pii/S1877050919316588.

Spis rysunków

2.1 Drzewo Trie dla leksykonu: ions, jins, cans, can. 18
2.2 DAWG dla leksykonu: ions, jins, cans, can. 19
2.3 Niezminimalizowany GADDAG dla leksykonu: ions, jins, cans, can. 20
2.4 Częściowo zminimalizowany GADDAG dla leksykonu: ions, jins, cans, can. . . 22
2.5 Przykładowa wygenerowana plansza Scrabble za pomocą gotowego symulatora. Pozycje kandydatów wg. autorów artykułu o grafie DAWG. 23
2.6 Przykładowa wygenerowana plansza Scrabble za pomocą gotowego symulatora. Pozycje kandydatów dla GADDAG, które są rezultatem badań. . . 24
2.7 Przykładowa kolumna z wygenerowanej plansza Scrabble za pomocą gotowego symulatora. Ilustracja przedstawiająca uproszczony graf GADDAG jedynie z tymi węzłami i krawędziami, które zostały wykorzystane do obliczenia zestawów skrośnych. 26
2.8 Przykładowy wiersz z wygenerowanej plansza Scrabble za pomocą gotowego symulatora. Ilustracja przedstawiająca generowanie ruchu "AIRCAPES"ze stojaka "RSEACRA", od pola kandydata "S", z zestawem skrośnym "R"dla pola bezpośrednio po lewej, wraz z podaną ścieżką w grafie, którą generator przeszedł. 27
2.9 Przykładowa generacja ruchu z duplikatem. Działanie algorytmu jest zgodne z treścią artykułu. W momencie przejścia automatu przez A oraz G, następuje iteracja po pierwszym oraz drugim Z w stojaku zgodnie z linią 5 pseudokodu. W rezultacie algorytm znajduje dwa takie same słowa do ułożenia. 28
2.10 Przykładowy ruch do obliczenia wartości oraz stojak z literami. Niebieskie pole jest 3-krotną premią literową. Znakiem specjalnym oznaczono mydełko. Wartość ruchu jest zależna od miejsca, w którym mydełko zostałoby ułożone. 29
3.1 Częstotliwość użycia pozycji na planszy w rozgrywkach. 32
3.2 Wartość ruchu w zależności od pozycji na planszy, na którym jest układane. . 32
5.1 Graf reprezentujący drzewo Minimax przy założeniu, że gracz ma do wyboru 100 ruchów w każdej turze. 49
5.2 Graf reprezentujący drzewo Minimax przy założeniu, że gracz ma do wyboru 100 ruchów w każdej turze oraz przeciwnik gra zachłannie. 50

Spis tabel

3.1 Wynik jako funkcja tury gry. 33
3.2 Wynik ruchu w zależności od zawartości stojaka oraz ruchu. 34

3.3 Średnia wartość ruchu od liczby płytek, które zostały w stojaku po poprzednim ruchu w całej grze. 35
3.4 Średnia wartość ruchu od liczby płytek, które zostały w stojaku po poprzednim ruchu w turze 10. gry. 35
3.5 Średnia wartość zdobyczy punktowej od liczby spółgłosek i samogłosek pozostałych po poprzednim ruchu w stojaku. 36
3.6 Wynik w zależności od liczby użytych płytek. 36
3.7 Wynik w zależności od długości słowa. 37
3.8 Dziesięć najczęściej występujących słów "JQXZ". 37
3.9 Wynik w zależności od częstotliwości słów. 38
3.10 Użycie mydełka w zależności od tury. 39
3.11 Litery znajdujące się w stojakach graczy po ukończonej rozgrywce. 40
3.12 Litery znajdujące się w stojaku gracza w przypadku braku możliwości wykonaniu ruchu. 41
3.13 Wpływ braku możliwości ruchu na zwycięstwo w rozgrywce. 42
3.14 Procent wygranych gier w zależności od otrzymanej litery. 42
4.1 Rezultat rozgrywek graczy korzystających ze strategii zachłannej 43
4.2 Tabela wartości liter pozostających w stojaku z artykułu Ballarda 44
4.3 Wyniki własnej symulacji dla strategii literowej Ballarda oraz zachłannej 45
4.4 Wyniki własnej symulacji dla strategii literowej Gordona oraz zachłannej . . . 45
4.5 Tabela wartości liter pozostających w stojaku z artykułu Gordona 46
4.6 Tabela wartości balansu spółgłosek i samogłosek z artykułu Gordona 47
4.7 Wyniki własnej symulacji dla połączonej strategii literowej oraz balansu spółgłosek i samogłosek Gordona oraz zachłannej. 47
5.1 Wyniki własnej symulacji dla połączonej strategii literowej oraz balansu spółgłosek i samogłosek Gordona z dodatkiem szybkiego zakończenia oraz zachłannej. 48