Politechnika Warszawska

WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

Praca dyplomowa licencjacka

na kierunku Matematyka

NAJPIERW WYGENERUJ STRONĘ TYTUŁOWĄ

KTÓRA ZNAJDUJE SIĘ

Numer albumu 000000

promotor

W KATALOGU title_page

konsultacje

I OTRZYMANEGO PDF-A NAZWIJ titlepage.pdf I WSTAW DO KATALOGU Z SZABLONEM!

WARSZAWA 2018

....

podpis promotora podpis autora

Streszczenie

Wizualizacja drzewa stanów algorytmu UCT

Streszczam.

Lorem ipsum dolor sit amet, consetetur sadipscing elit, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Słowa kluczowe: slowo1, slowo2, ...

Abstract

Visualization of UCT trees

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Keywords: keyword1, keyword2, ...

Warszawa, dnia .....

Oświadczenie

Oświadczam, że pracę inżynierską pod tytułem „Wizualizacja drzewa stanów algorytmu UCT", której promotorem jest mgr inż. Jan Karwowski, wykonałam/wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Spis treści

1. Wykaz najważniejszych oznaczeń i skrótów

- **MCTS** (Monte Carlo Tree Search) - heurystyka podejmowania decyzji w pewnych zadaniach sztucznej inteligencji, np. ruchów w grach. Najczęściej MCTS opiera się na wariancie metody UCT.
- **UCT** (Upper Confidence Bound Applied to Trees) - algorytm przeszukujący drzewo stanów rozgrywki w poszukiwaniu najbardziej opłacalnych ruchów. Algorytm stara się zachować równowagę między eksploatacją ruchów po ruchach o wysokiej średniej wygranej a eksploracją tych mało sprawdzonych.
- **Podstawowa wizualizacja** - możliwość wyświetlenia całego drzewa stanów. W podstawowej wizualizacji wygląd wierzchołków i krawędzi jest nieistotny.
- **Zaawansowana wizualizacja** - podstawowa wizualizacja wzbogacona o możliwość analizowania statystyk poszczególnych wierzchołków. Kolor wierzchołków będzie reprezentował aktualnego gracza, a kolor krawędzi częstość odwiedzin danego węzła. Możliwe powinno być też przewijanie, przybliżanie oraz oddalanie podglądu drzewa.
- **CSV** - plik w formacie .csv (ang. *comma-separated values*) służący do przechowywania danych w plikach tekstowych, gdzie separatorem jest przecinek.

2. Wstęp i cel pracy

2.1. Cel biznesowy

Algorytm UCT, będący usprawnieniem MCTS, jest powszechnie stosowanym algorytmem w sztucznej inteligencji. Jest metodą analizującą obiecujące ruchy na podstawie generowanego drzewa, która równoważy eksploatację najbardziej korzystnych z eksploracją mniej korzystnych decyzji. Każdemu wierzchołkowi drzewa odpowiada pewien stan rozgrywki, z którego algorytm rozgrywa losowe symulacje, rozszerzając potem drzewo o kolejne możliwe stany. Sposób, w jaki rozrasta się opisywane drzewo, jest kluczowy dla podejmowania przez algorytm jak najlepszych decyzji.

Celem projektu jest stworzenie aplikacji pozwalającej na wizualizację drzew algorytmu UCT. Aplikacja będzie pozwalała na wizualizowanie drzew generowanych podczas rozgrywania dwóch przykładowych gier (pozwalając przetestować rozwiązanie). Aplikacja powinna pozwalać na wizualizację drzew, ich sekwencji i róznic między kolejnymi drzewami w sekwencji. Powinna być możliwość płynnego przybliżania/oddalania i przewijania wizualizacji oraz zapisu aktualnego stanu do pliku graficznego - wszystko, aby klient mógł wygodnie korzystać z naszego programu. Taki produkt pozwoliłby zrozumieć klientowi ideę i sposób działania algorytmu UCT.

2.2. Założenia projektowe

2.2.1. Założenia funkcjonalne

Użytkownik korzystający z naszej aplikacji będzie mógł wybrać jedną z dwóch gier deterministycznych, a do wyboru będzie miał trzy tryby rozgrywki:

1. Gracz vs PC – użytkownik będzie decydował o swoich posunięciach i zmierzy się on z zaimplementowanym algorytmem,
2. PC vs PC – użytkownik będzie świadkiem symulacji algorytmu, który rozgrywa partię z

samym sobą,

3. Gracz vs Gracz – rozgrywka dwóch graczy, bez wizualizacji.

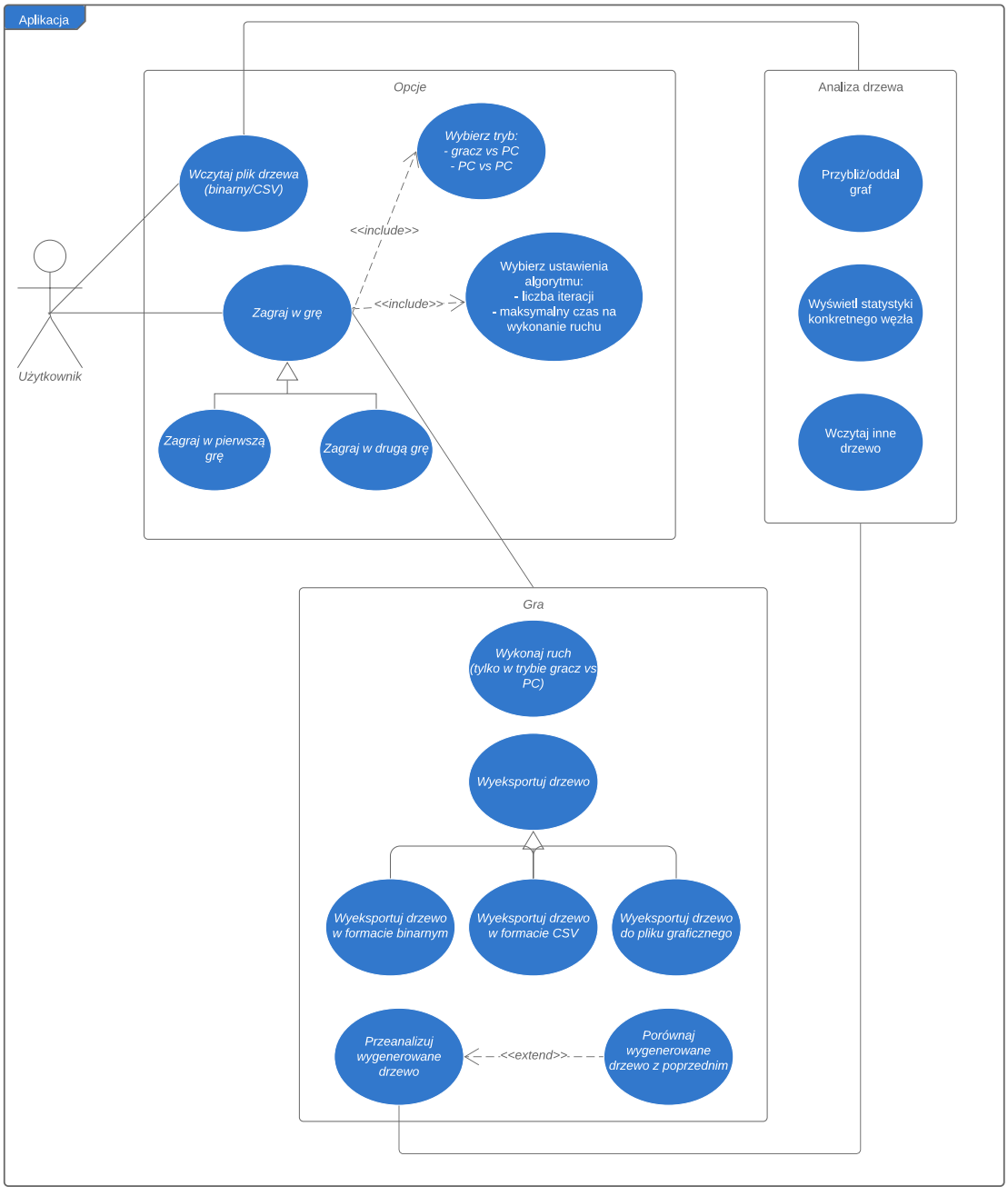
Zanim jednak przejdzie do rozgrywki, będzie on miał możliwość ustawienia parametrów algorytmu, tj. liczbę iteracji podczas tworzenia drzewa, czy też maksymalny czas na ruch przeciwnika. Druga opcja, którą będzie dysponować użytkownik, to możliwość wczytania drzewa w formacie zarówno binarnym jak i CSV, a następnie możliwość jego dogłębnej analizy. Będzie on mógł wyświetlać informacje na temat wybranego węzła, a także przybliżać i oddalać całą wygenerowaną strukturę. Podczas samej rozgrywki, po wykonanym ruchu przeciwnika gracz będzie mógł analizować drzewo w sposób opisany powyżej, a także wyeksportować je. Będzie też miał możliwość zapisania go w wyżej wymienionych formatach, a także w formacie rastrowym. Użytkownik będzie mógł oglądać animację rozrostu drzewa. Powyższe rzeczy dotyczą trybów gry z udziałem algorytmu i każdej z gier.

Aplikacja będzie potrzebować procesora graficznego. Dostęp do sieci nie jest potrzebny dla poprawnego działania aplikacji.

Diagram przypadków użycia, który ilustruje przedstawione możliwości, znajduje się na rysunku 2.1.

2.2.2. Założenia niefunkcjonalne

Wymagania niefunkcjonalne opisane są w tabeli 2.1 przy użyciu metody FURPS. Zawiera ona między innymi opis wymagań sprzętowych aplikacji oraz jej ograniczenia wydajnościowe.



Rysunek 2.1: Diagram przypadków użycia

Tablica 2.1: Analiza FURPS

Obszar	**Opis**
Używalność	Aplikacja działa w jednym oknie wyposażonym w przejrzysty interfejs dla użytkownika. Ponadto, dostarczona jest instrukcja instalacji i obsługi programu.
Niezawodność	Aplikacja działa na komputerze lokalnym i po zainstalowaniu jest dostępna cały czas. Potencjalne błędy nie powinny zamykać aplikacji, a jedynie wyświetlić komunikat dla użytkownika.
Wydajność	Aplikacja koryszta głównie z pamięci RAM, procesora i procesora graficznego. Dla drzew do 100 000 wierzchołków wizualizacja nie powinna zajmować więcej niż 3s, a dla 250 000 — 5s. Wszystkie obliczenia i wizualizacje są przeprowadzane wewnątrz jednej maszyny.
Wsparcie	Aplikacja jest przeznaczona dla komputerów z systemami operacyjnymi Windows oraz Linux.

3. Teoria

3.1. Algorytmy MCTS

MCTS (Monte Carlo Tree Search) - heurystyka podejmowania decyzji w pewnych zadaniach sztucznej inteligencji, np. ruchów w grach. Najczęściej MCTS opiera się na jakimś wariancie metody UCT.

3.1.1. Opis grupy algorytmów

Algorytm jest opisany w formie pseudokodu w listingu 3.1. Nasza implementacja będzie oparta o [1].

Algorytm Monte Carlo Tree Search opiera się na rozbudowywaniu drzewa ze stanami gry, poprzez iteracyjne wykonywanie czterech faz, opisanych poniżej.

1. **Faza wyboru** (linia 6 w listingu) - wybranie najbardziej obiecującego wierzchołka do rozrostu drzewa. Istotny w tej fazie jest balans pomiędzy eksploracją ruchów przeanalizowanych najdokładniej oraz eksploatacją jeszcze niezbadanych.

2. **Faza rozrostu** (linia 7 w listingu) - utworzenie wierzchołków potomnych dla najbardziej obiecującego wierzchołka. Tworzone wierzchołki odpowiadają stanom możliwym do uzyskania poprzez wykonanie jednego ruchu ze stanu wierzchołka obiecującego.

3. **Faza symulacji** (linia 8 w listingu) - rozegranie losowej rozgrywki ze stanu jednego z utworzonych wierzchołków utworzonych w poprzedniej fazie.

4. **Faza propagacji wstecznej** (linia 9 w listingu) - aktualizacja informacji wierzchołków na ścieżce od wierzchołka, z którego rozpoczęto symulację, do korzenia drzewa.

```
1 def find_next_move(curr_state):
2 iterations_counter = 0
3 tree = initialize_tree(curr_state)
4
```

```
while iterations_counter < max_iterations_counter:
# selection(tree.root)
curr_node = select promising node based on UCT formula

# expansion(node)
create child nodes from node

# simulation(node)
playout_result = simulate random playout from curr_node

# backpropagation(node, playout_result)
update tree according to playout_result

iterations_counter++

best_state = select best child(tree.root)
return best_state
```

Listing 3.1: Pseudokod algorytmu Monte Carlo Tree Search

3.2. Algorytm UCT

UCT (Upper Confidence Bound Applied to Trees) - algorytm przeszukujący drzewo stanów rozgrywki w poszukiwaniu najbardziej opłacalnych ruchów. Algorytm stara się zachować równowagę między eksploatacją ruchów po ruchach o wysokiej średniej wygranej a eksploracją tych mało sprawdzonych.

3.2.1. Opis algorytmu

Moduł **Algorytm** jest implementacją algorytmu Monte Carlo Tree Search, korzystającą z wariantu UCT. Odpowiedzialnością tego modułu jest wyznaczanie kolejnego ruchu na podstawie dostarczonego stanu gry. Opisywany moduł będzie odpowiadał za iteracyjne tworzenie drzewa stanów i przeszukiwanie go w celu wyznaczenia najbardziej korzystnego ruchu. Użytkownik będzie miał możliwość zmiany liczby iteracji algorytmu albo ograniczenie czasowe jego działania.

W listingu 3.1 operujemy na trzech istotnych zmiennych - **tree**, **curr_node** i **curr_state**. Odpowiedzialnością struktury opisującej **tree** jest przechowywanie korzenia drzewa oraz stanu wyjściowego rozgrywki, który jest tej samej struktury co zmienna **curr_state**. Struktura opi-

sująca **curr_node** przechowuje wszystkie informacje na temat wierzchołka drzewa, wraz z referencjami do wierzchołków potomnych i rodzica.

3.2.2. Dodatkowe założenia

Aby gra była poprawnie obsłużona przez utworzony system, musi spełniać następujące założenia:

1. Rozgrywka jest prowadzona naprzemiennie przez dwóch graczy.
2. Każdy ruch ma jednoznaczny wpływ na dalszą rozgrywkę (rozgrywka jest deterministyczna).
3. Każdy z graczy ma dostęp do pełnej informacji o aktualnym stanie gry.

3.3. Algorytm wizualizacji drzewa

3.3.1. Określenie problematyki

3.3.2. Założenia

Aby wizualizacja była czytelna, poczyniliśmy następujące założenia:

1. Krawędzie drzewa nie mogą się przecinać.
2. Wierzchołki będą ustawione od góry w rzędach, a przynależność do rzędów będzie zależała od odległości wierzchołków od korzenia.
3. Wierzchołki mają być narysowane możliwie najwęziej.

3.3.3. Usprawniony algorytm Walkera

Aby wyznaczyć układ wierzchołków na płaszczyźnie, spełniając powyższe 3 założenia, skorzystamy z usprawnionego algorytmu Walkera, który działa w czasie liniowym względem liczby wierzchołków. Algorytm, który zaimplementujemy, został opisany w [2].

4. Implementacja

4.1. Wykorzystane technologie

W naszym projekcie zdecydowaliśmy się skorzystać z:

1. Języka *Python* w wersji 3.7.2.

2. Biblioteki *VisPy* w wersji 0.6.3, która udostępnia komponenty związane z wizualizacją graficzną. Wykorzystujemy tę bibliotekę w połączeniu z *OpenGL* w wersji 2.1. Biblioteka *VisPy* jest stworzona w oparciu o licencję *BSD*, co w kontekście projektu na pracę inżynierską pozwala na modyfikowanie i wykorzystywanie jej.

3. Nakładki na bibliotekę Qt - *PyQt5* w wersji 5.9.2. *PyQt5* umożliwia tworzenie interfejsu graficznego. Dla projektów takich jak praca inżynierska, *PyQt* dystrybuowana jest na zasadach *GNU General Public License*.

4. Biblioteki *fman build system (fbs)* w wersji 0.8.4, ułatwiającej pakowanie aplikacji w języku *Python* korzystających z biblioteki *PyQt*. To oprogramowanie dystrybuowane jest na zasadach *GNU General Public License*.

5. Narzędzia *pdoc* w wersji 0.3.2, służącego do automatycznego generowania dokumentacji aplikacji.

4.2. Architektura i działanie systemu

Aplikacja będzie podzielona na pięć oddzielnych modułów: *Algorytm*, *Serializacja*, *Wizualizacja*, *Gry*, które będą funkcjonować w obrębie nadrzędnego modułu - *Aplikacji głównej*. Cele każdego z modułów i zadania powierzone im są przedstawione poniżej.

4.2.1. Algorytm

Moduł *Algorytm* jest implementacją algorytmu Monte Carlo Tree Search, korzystającą z wariantu UCT. Odpowiedzialnością tego modułu jest wyznaczanie kolejnego ruchu na podstawie dostarczonego stanu gry. Opisywany moduł będzie odpowiadał za iteracyjne tworzenie drzewa stanów i przeszukiwanie go w celu wyznaczenia najbardziej korzystnego ruchu. Użytkownik będzie miał możliwość zmiany liczby iteracji algorytmu albo ograniczenie czasowe jego działania.

4.2.2. Serializacja

Serializacja jest modułem odpowiadającym za zapisywanie drzew do plików formacie binarnym lub csv. Oba schematy są rekurencyjne, bo taka jest również struktura generowanych przez aplikację drzew. To oznacza, że w celu zapisania całego drzewa, wystarczy zserializować jego korzeń.

Serializacja binarna

W serializacji binarnej przyjmujemy opisany niżej schemat.

- **liczba całkowita** - wartość liczby zakodowanej w U2 na 4 bajtach. Bajty liczby w kolejności little endian.
- **napis**:
  - liczba bajtów w napisie *(liczba całkowita)*,
  - zawartość napisu kodowana w UTF8.
- **liczba zmiennoprzecinkowa** - wartość liczby zakodowanej w IEEE754 na 64 bitach w kolejności little endian.
- **wierzchołek:**
  - nazwa stanu *(napis)*,
  - m - liczba węzłów potomnych *(liczba całkowita)*,
  - m powtórzeń następującego bytu:
    - nazwa ruchu *(napis)*,
    - licznik odwiedzin *(liczba całkowita)*,
    - dodatkowy licznik odwiedzin *(liczba całkowita)*,
    - średnia wypłata *(liczba zmiennoprzecinkowa)*,

* węzeł potomny *(wierzchołek)*.

Serializacja do plików csv

W serializacji do plików csv przyjmujemy, że każdy kolejny wiersz odpowiada kolejnemu wierzchołkowi drzewa, a kolejne wartości opisujące wierzchołek oddzielamy przecinkami. Ostatnią wartością jest liczba wierzchołków potomnych. Każdy wierzchołek serializujemy do wiersza postaci:

R, O, O2, W, S, D

Oznaczenia:

- R - nazwa ruchu,
- O - licznik odwiedzin,
- O2 - dodatkowy licznik odwiedzin,
- W - średnia wypłata algorytmu za ruch,
- S - nawa stanu,
- D - liczba wierzchołków potomnych.

Kolejność wierszy opisujących wierzchołki jest analogiczna do odwiedzania wierzchołków przez algorytm przeszukiwania drzewa wgłąb, począwszy od korzenia.

- Jeśli wierzchołek v ma jednego potomka v_1, to wiersz opisujący v_1 znajduje się pod wierszem opisującym v.
- Jeśli wierzchołek v ma n potomków $v_1, v_2, ..., v_n$ i żaden z potomków nie ma swoich potomków, to pod wierszem opisującym v kolejne n wierszy opisuje wierzchołki $v_1, v_2, ..., v_n$.

4.2.3. Wizualizacja

Moduł *Wizualizacja* udostępnia funkcjonalność wizualizacji dostarczonych drzew. Użytkownik będzie miał również możliwość przybliżania, oddalania oraz poruszania się po wizualizacji. Opisana interaktywność ma na celu umożliwić dokładne zbadanie struktury drzewa oraz poszczególnych wartości w interesujących go wierzchołkach.

4.2.4. Gry

Aplikacja będzie udostępniała 2 gry planszowe umożliwiające zademonstrowanie efektywności wizualizacji oraz algorytmu.

4.2.5. Aplikacja główna

Aplikacja główna jest modułem łączącym wszystkie pozostałe. Ten moduł skupia się na zaprezentowaniu funkcjonalności wszystkich modułów w formie aplikacji okienkowej. Obszerny opis projektu aplikacji okienkowej znajduje się w rozdziale ???.

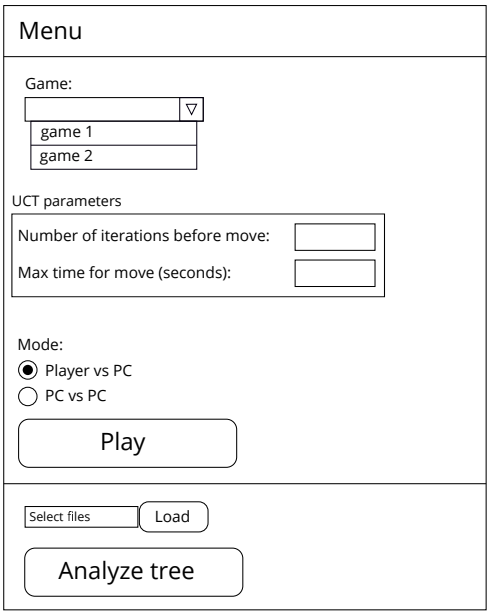
4.3. Główne komponenty aplikacji

4.4. Interfejs użytkownika

Graficzny interfejs użytkownika składać się będzie z trzech głównych okien, a logika jego działania będzie w całości zawarta w module *Aplikacja główna*. Zadaniem graficznego interfejsu jest umożliwienie uruchomienia poszczególnych modułów użytkownikom końcowym.

4.4.1. Menu główne

Ukazane na rysunku 4.1 menu główne będzie głównym oknem aplikacji i będzie to pierwsza rzecz, którą zobaczy użytkownik po uruchomieniu programu.



Rysunek 4.1: Okno menu głównego

Dwa moduły, do których można przejść z tego okna, to rozgrywka i analiza drzewa. Żeby rozegrać grę, należy nacisnąć na przycisk *Play*. Powyżej tego przycisku znajdować się będzie szereg opcji, który pozwoli użytkownikowi ustawić parametry gry dostosowane do jego preferencji, w tym między innymi:

- wybór gry - rozwijana lista, w której znajdować się będą zaimplementowane gry (nasz projekt przewiduje dwa tytuły),
- liczba iteracji (rozgrywek), jaką komputer będzie wykonywał przed wykonaniem ruchu,
- maksymalny czas na wykonanie ruchu - czas, po którym komputer będzie przerywał obliczenia i wykona ruch,
- tryb rozgrywki:
  - człowiek kontra człowiek,
  - człowiek kontra maszyna,
  - maszyna kontra maszyna.

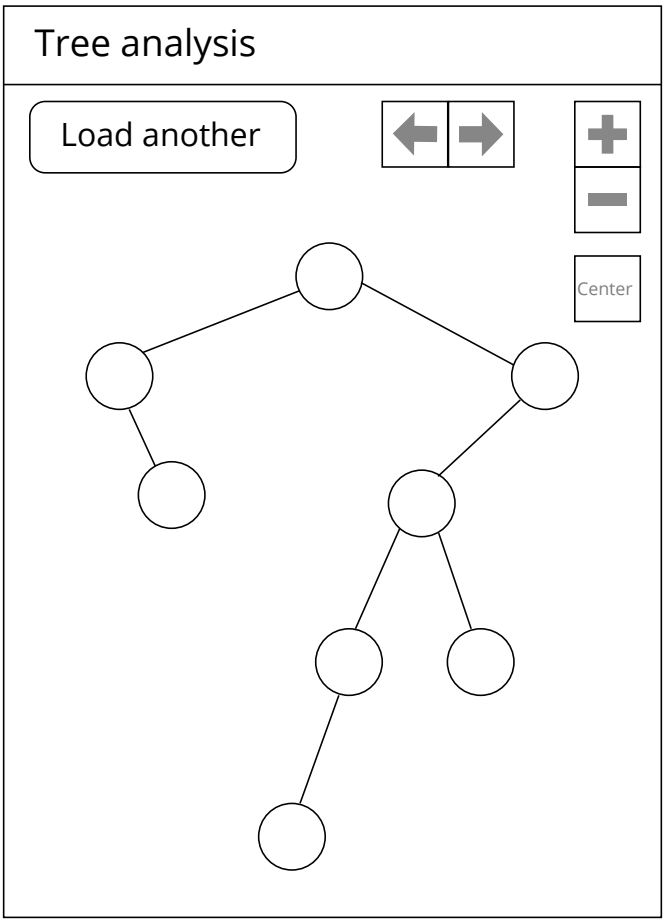
Analiza drzewa (lub drzew) będzie dostępna po naciśnięciu przycisku *Analyze tree* i uprzedniego wczytania wybranych plików (.tree, .csv).

4.4.2. Analiza drzewa

W oknie ukazanym na rysunku 4.2 będziemy mogli oglądać wczytane lub wygenerowane drzewa.

Kluczową funkcją będzie tutaj możliwość dynamicznego przybliżania i oddalania go (za pomocą przycisków plusa i minusa na ekranie lub scrolla) wraz z możliwością klikania poszczególnych węzłów w celu pozyskania stanu rozgrywki w danym momencie. Widoczna będzie także informacja o tym, ile razy algorytm odwiedził dany węzeł, ile razy doprowadził on do wygranej oraz średnią nagrodę za ruch w danym węźle. Możliwe będzie też wycentrowanie oglądanego drzewa.

W przypadku wczytania większej ilości drzew będzie możliwość przełączania ich za pomocą przycisków ze strzałkami w lewo i w prawo. Zaznaczane wtedy będą różnice w węzłach i krawędziach względem poprzedniego drzewa.



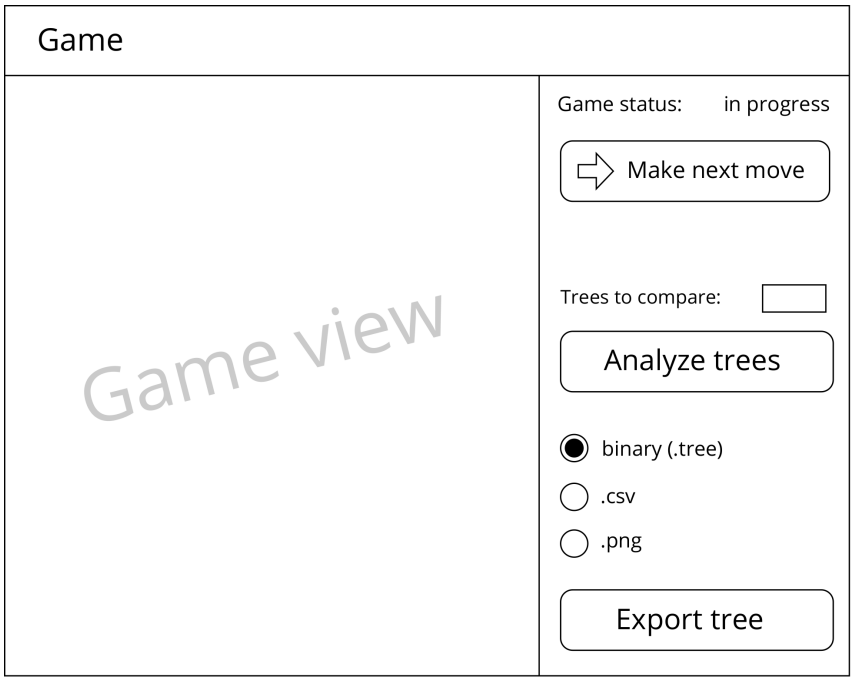
Rysunek 4.2: Okno analizy drzewa

4.4.3. Rozgrywka

Poniżej przedstawiony jest przykładowy interfejs graficzny, do którego użytkownik będzie miał dostęp podczas rozgrywki.

Zgodnie z projektem okna przedstawionym na rysunku 4.3, widok rozgrywki będzie podzielony na dwie części. Gra zawierać się będzie w wyżej pokazanym oknie po lewej stronie. To tutaj użytkownik za pomocą przygotowanego do gier GUI będzie mógł wykonać ruch. W prawej części okna znajdować się będą opcje związane z aktualnym stanem rozgrywki, między innymi:

- informacja o aktualnym stanie gry.
- wykonaj kolejny ruch - wyłącznie w trybie rozgrywki maszyna kontra maszyna. Użytkownik będzie miał możliwość kontrolowania wykonywanych przez komputer ruchów, aby samemu móc powodować postęp w rozgrywce.
- przeanalizuj wygenerowane drzewa - będzie to przycisk otwierający drugie okno z opisaną wcześniej analizą drzewa. Nad przyciskiem znajduje się parametr mówiący ile ruchów



Rysunek 4.3: Okno rozgrywki

wstecz użytkownik ma zamiar analizować. Jedno drzewo będzie odpowiadać jednemu ruchowi komputera, a jako pierwsze wyświetli się drzewo przedstawiające stan z ostatniego ruchu i reszta będzie odpowiednio w kolejności chronologicznej (od końca).

- wyeksportuj drzewo do pliku (csv, png lub binarnego).

5. Instrukcje

5.1. Instrukcja instalacji

5.2. Instrukcja użytkownika

6. Podsumowanie i ocena

6.1. Uzyskane efekty

6.2. Kontynuacja pracy

6.3. Wydajność

6.4. Testy akceptacyjne

Testy akceptacyjne zostały przeprowadzone w celu sprawdzenia, czy aplikacja spełnia założenia opisane w dokumentacji wymagań projektu. Test 6.1 konfrontuje założenia modułu *Gry*, test 6.2 – modułu *Serializacja*, a pozostałe testy weryfikują założenia modułu *Wizualizacja*.

Testy akceptacyjne zostały wykonane na komputerze:

- z zainstalowanym systemem operacyjnym *Windows 10 Education N*,
- z zainstalowanym interpreterem języka *Python 3.7.2* i biblioteką *PyQt5*,
- wyposażonym w procesor *Intel Core i7-8700k @3.70 GHz*,
- wyposażonym w kartę graficzną *NVIDIA GeForce GTX 1060 6GB*,
- wyposażonym w 32GB pamięci RAM.

Pierwszy z przeprowadzonych testów dotyczy funkcjonalności modułu *Gry*, a wynik opisany został w tabeli 6.1.

Tablica 6.1: Raport z pierwszego testu

Testowane wymaganie	*Użytkownik będzie mógł wybrać jedną z dwóch przykładowych gier, a do wyboru będzie miał trzy tryby rozgrywki.*
Kroki testowe	1. Z menu głównego aplikacji wybierz opcję *Chess*. 2. Z menu głównego aplikacji wybierz opcję *Player vs player* i sprawdź tryb rozgrywki dla dwóch graczy. 3. Z menu głównego aplikacji wybierz opcję *Player vs PC* dla różnych ustawień algorytmu UCT. 4. Z menu głównego aplikacji wybierz opcję *PC vs PC* dla różnych ustawień algorytmu UCT. 5. Z menu głównego aplikacji wybierz *Mancala* i powtórz kroki 2–5.
Wynik	Pozytywny.

Drugi z przeprowadzonych testów dotyczy funkcjonalności modułu *Serializacja*, a wynik opisany został w tabeli 6.2.

Tablica 6.2: Raport z drugiego testu

Testowane wymaganie	*Użytkownik będzie mógł zapisać analizowane drzewa do pliku csv, do pliku binarnego oraz do bitmapy.*
Kroki testowe	1. Z menu głównego aplikacji wybierz ścieżkę do dowolnego pliku z zserializowanym drzewem. 2. Naciśnij przycisk *Inspect tree*. 3. Naciśnij przycisk *Save to csv file*. 4. Naciśnij przycisk *Save to binary file*. 5. Naciśnij przycisk *Save to bitmap file*. 6. Sprawdź, czy bitmapa wygenerowana w kroku 5 odpowiada drzewu z pliku początkowego. 7. Z menu głównego aplikacji wybierz ścieżkę plików wygenerowanych w kroku 3 i 4, żeby sprawdzić, czy zapisane drzewa wizualizowane są tak samo jak w początkowym pliku.
Wynik	Pozytywny.

Trzeci z przeprowadzonych testów dotyczy funkcjonalności modułu *Wizualizacja*, a wynik opisany został w tabeli 6.3.

Tablica 6.3: Raport z trzeciego testu

Testowane wymaganie	*Użytkownik będzie mógł wyświetlić informacje związane z wybranym węzłem drzewa, a także przybliżać i oddalać cały graf.*
Kroki testowe	1. Z menu głównego aplikacji wybierz ścieżkę do dowolnego pliku z drzewem. 2. Naciśnij przycisk *Inspect tree*. 3. Przy użyciu prawego przycisku myszki chwyć za obszar rysowania i poruszaj się po wizualizacji. 4. Używając kółka myszki, przybliż i oddal wizualizowane drzewo. 5. Kliknij dowolny wierzchołek drzewa lewym przyciskiem myszki i sprawdź, czy panel z prawej strony wyświetla informacje związane z wybranym wierzchołkiem.
Wynik	Pozytywny.

Czwarty z przeprowadzonych testów dotyczy wydajności modułu *Wizualizacja*, a wynik opisany został w tabeli 6.4.

Tablica 6.4: Raport z czwartego testu

Testowane wymaganie	*Dla drzew do 100 000 wierzchołków wizualizacja nie powinna zajmować więcej niż 3s.*
Kroki testowe	1. Z menu głównego aplikacji wybierz ścieżkę do pliku *tree_100k.csv*. 2. Naciśnij przycisk *Inspect tree*.
Wynik	Pozytywny – deserializacja, ulepszony algorytm Walkera i wyświetlenie drzewa z pliku zajęło 2.802s.

Piąty z przeprowadzonych testów dotyczy wydajności modułu *Wizualizacja*, a wynik opisany został w tabeli 6.5.

Tablica 6.5: Raport z piątego testu

Testowane wymaganie	*Dla drzew do 250 000 wierzchołków wizualizacja nie powinna zajmować więcej niż 5s.*
Kroki testowe	1. Z menu głównego aplikacji wybierz ścieżkę do pliku *tree_250k.csv*. 2. Naciśnij przycisk *Inspect tree*.
Wynik	Pozytywny – deserializacja, ulepszony algorytm Walkera i wyświetlenie drzewa z pliku zajęło 4.626s.

Wszystkie testy akceptacyjne zakończyły się pozytywnie, a więc wymagania zostały spełnione.

7. Wnioski

asd

Bibliografia

[1] Levente Kocsis, Csaba Szepesvári, *Bandit based Monte-Carlo Planning*, Berlin, Germany, September 18–22, 2006.

[2] Christop Buchheim, Michael Jünger, Sebastian Leipert *Improving Walker's Algorithm to Run in Linear Time*, Universität zu Köln, Institut für Informatik, 2002.

Wykaz symboli i skrótów

nzw.	nadzwyczajny
*	operator gwiazdka
~	tylda

Jak nie występują, usunąć.

Spis rysunków

Spis tabel

Spis załączników

1. Załącznik 1
2. Załącznik 2
3. Jak nie występują, usunąć rozdział.