Politechnika Warszawska

WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

Praca dyplomowa inżynierska

na kierunku Informatyka

Wizualizacja drzewa stanów algorytmu UCT

Patryk Fijałkowski

Numer albumu 286350

Grzegorz Kacprowicz

Numer albumu 286358

promotor

mgr inż. Jan Karwowski

opiekun naukowy

prof. dr hab. inż. Jacek Mańdziuk

WARSZAWA 2020

....

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
- **UCT** - Upper Confidence Bound Applied to Trees
- **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.

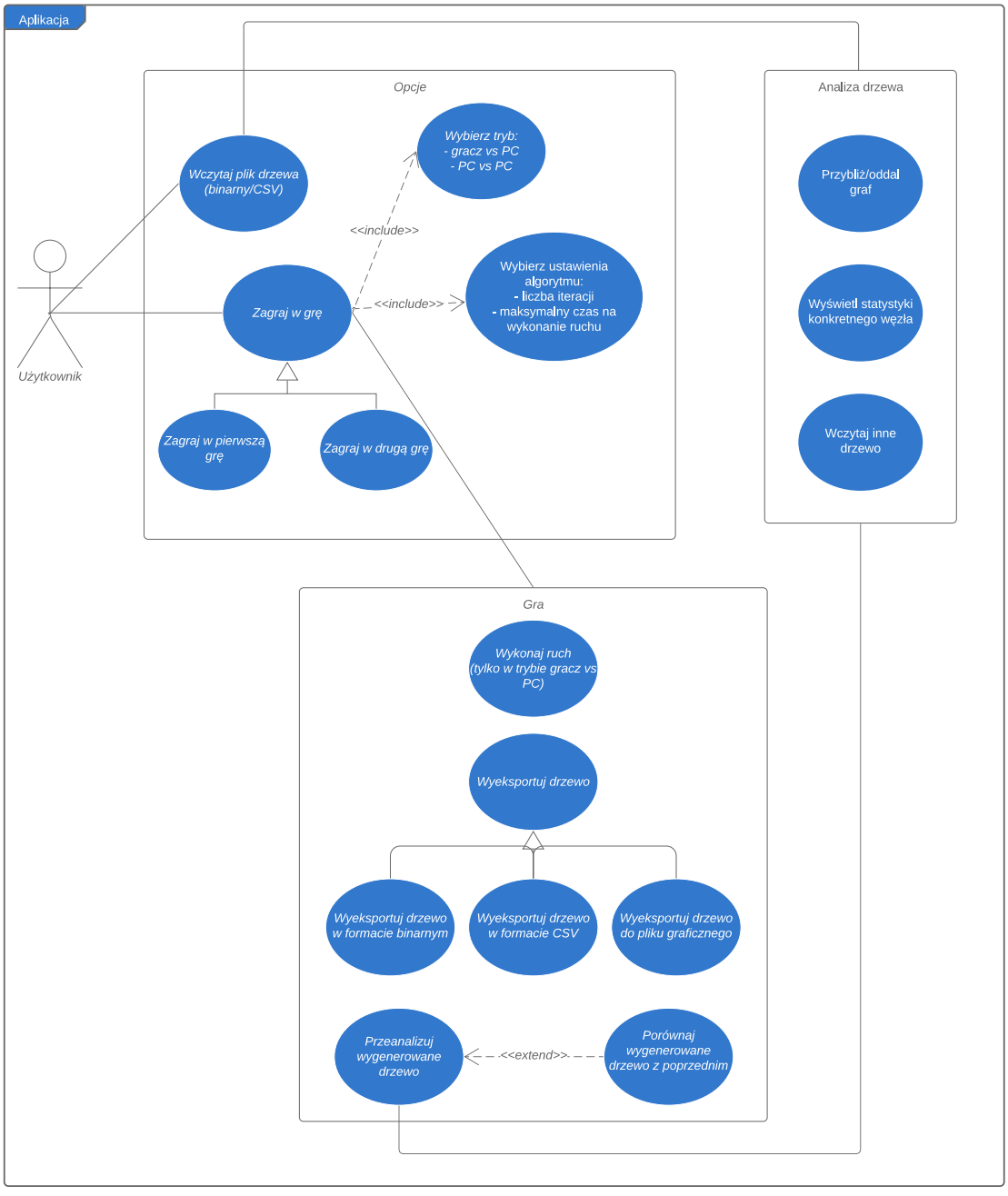
Zanim jednak przejdzie do rozgrywki, będzie on miał możliwość ustawienia parametrów algorytmu, tj. liczbę iteracji podczas tworzenia drzewa, czy też maksymalny czas na ruch przeciwnika. Druga opcja, którą będzie dysponować użytkownik, to możliwość wczytania drzewa w formacie zarówno binarnym jak i CSV, 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

3.1.1. Opis grupy algorytmów

MCTS jest heurystyką, której celem jest podejmowanie decyzji w pewnych zadaniach sztucznej inteligencji, na przykład ruchów w grach. Metoda jest oparta na przeszukiwaniu możliwych stanów zapisanych w wierzchołkach drzewa i losowym symulowaniu rozgrywek. Najpowszechniejszym wariantem MCTS jest algorytm UCT. MCTS opiera się na rozbudowywaniu drzewa ze stanami gry poprzez iteracyjne wykonywanie czterech faz. Pseudokod opisany w listingu 3.1 oraz implementacja MCTS w prezentowanym systemie bazują na [1].

1. **Faza wyboru** (linia 7 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 9 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 11 w listingu) - rozegranie losowej rozgrywki ze stanu jednego z utworzonych wierzchołków utworzonych w poprzedniej fazie.

4. **Faza propagacji wstecznej** (linia 13 w listingu) - aktualizacja informacji wierzchołków na ścieżce od wierzchołka, z którego rozpoczęto symulację, do korzenia drzewa.

```
def find_next_move(curr_state):
    iterations_counter = 0
    tree = initialize_tree(curr_state)

    while iterations_counter < max_iterations_counter:
        # selection(tree.root)
        curr_node = select promising node
        # 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

3.2.1. Opis algorytmu

UCT jest wariantem metody MCTS, który stara się zachować równowagę między eksploatacją ruchów o wysokiej średniej wygranej a eksploracją tych mało zbadanych. Formuła, która odpowiada za wyznaczenie najbardziej obiecującego wierzchołka w fazie wyboru MCTS jest przedstawiona w wyrażeniu 3.1.

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}} \tag{3.1}$$

W wyrażeniu 3.1, indeks i odnosi się do liczby wykonanych przez algorytm iteracji. W pierwszym składniku sumy wyrażenia 3.1, licznik w_i oznacza liczbę wygranych w danym węźle, a mianownik n_i oznacza liczbę rozegranych symulacji. Zatem ułamek ten przyjmuje wartości większe dla ruchów o większej średniej wygranej, co odpowiada ze eksploatację drzewa. Drugi składnik sumy wyrażenia 3.1 przyjmuje wartości większe dla wierzchołków, dla których wykonano mniej symulacji; ponieważ $N_i = \sum_i n_i$, a c jest parametrem eksploracji, który może być dostosowany do badanego problemu (najczęściej przyjmuje się $c = \sqrt{2}$).

3.2.2. Dodatkowe założenia

Stworzone przez nas rozwiązanie dodaje pewne założenia do algorytmu UCT. Dodanie założeń pozwoliło nam na stworzenie rozwiązania bardziej uniwersalnego i niezależnego od zasad i logiki badanych gier.

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

Celem prezentowanego rozwiązania jest wizualizacja drzewa w sposób najbardziej przejrzysty dla użytkownika. Ma to ułatwić użytkownikowi poznanie struktury drzewa i zależności między wierzchołkami. Jako że zaprojektowanie układu wierzchołków przy pewnych założeniach jest problemem NP-zupełnym nawet dla drzew binarnych, co zostało opisane w [3], w prezentowanym rozwiązaniu posłużymy się heurystyką.

3.3.2. Założenia

W celu uczynienia wizualizacji jak najbardziej czytelną, poczyniliśmy pewne założenia w kontekście układu wierzchołków i krawędzi wizualizowanych drzew. Zostały one wymienione poniżej.

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

W celu wyznaczenia układ wierzchołków drzewa na płaszczyźnie, spełniając powyższe 3 założenia, skorzystamy z usprawnionego algorytmu Walkera, który działa w czasie liniowym względem liczby wierzchołków. Algorytm, który zaimplementowaliśmy, został opisany w [2].

4. Implementacja

4.1. Architektura i działanie systemu

4.1.1. Wykorzystane technologie

W naszym projekcie zdecydowaliśmy się skorzystać z technologii wymienionych poniżej.

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. Biblioteki *NumPy* w wersji 1.18.1, która odpowiada za wydajne operacje na macierzach. Zgodnie z umową licencyjną opisaną przez autorów *NumPy*, można wykorzystywać ich narzędzie w zakresie pracy naukowej.

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

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

6. Narzędzia *pdoc* w wersji 0.3.2, służącego do automatycznego generowania dokumentacji aplikacji. Zgodnie z umową licencyjną opisaną przez autorów *pdoc*, można wykorzystywać ich narzędzie bez ograniczeń.

4.1.2. Wstęp

Aplikacja jest 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 w rozdziałach 4.1.3 - 4.1.7.

4.1.3. Algorytm

Moduł *Algorytm* jest implementacją metody MCTS, 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, opisującym zaimplementowany algorytm, operujemy na trzech istotnych zmiennych - *tree*, *curr_node* i *curr_state*. Odpowiedzialnością struktury opisującej drzewo - *tree* - jest przechowywanie korzenia oraz stanu wyjściowego rozgrywki, który jest tej samej struktury co zmienna *curr_state*. Struktura opisująca wierzchołek - *curr_node* - przechowuje wszystkie informacje na temat wierzchołka drzewa, wraz z referencjami do wierzchołków potomnych i rodzica. Dokładne relacje między komponentami zostały opisane na diagramie 4.1.

4.1.4. 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 przekazać odpowiednim komponentom 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 $v_1, v_2, ..., v_n$.

4.1.5. Wizualizacja

Moduł *Wizualizacja* udostępnia funkcjonalność wizualizacji dostarczonych drzew. *Wizualizacja* jest jedynym modułem, który korzysta z technologii *VisPy* oraz *NumPy*. Wykorzystanie tych technologii ma na celu odpowiednio wydajne wyświetlenie wizualizacji oraz przechowywanie wektorów z danymi, które będzie można przekazać karcie graficznej. Zadania *Wizualizacji* są wymienione poniżej.

- Przypisanie wierzchołkom drzewa miejsce na płaszczyźnie przy użyciu usprawnionego algorytmu Walkera i przetransformowanie wyznaczonych koordynatów do układu współrzędnych OpenGL.
- Przypisanie krawędziom koloru w zależności od liczby odwiedzin wierzchołka.
- Przypisanie wierzchołkom koloru w zależności od gracza reprezentowanego stanu gry.
- Detekcja kliknięcia wierzchołka przez użytkownika.
- Wyświetlenie drzewa.
- Przybliżanie, oddalanie i poruszanie się po wizualizacji.

4.1.6. Gry

Prezentowane rozwiązanie udostępnia dwie gry planszowe w ramach modułu *Gry*. System został przygotowany z myślą o rozszerzaniu o kolejne gry. Wymagania, które należy spełnić, dodając kolejną grę, są opisane w rozdziale 4.2.1.

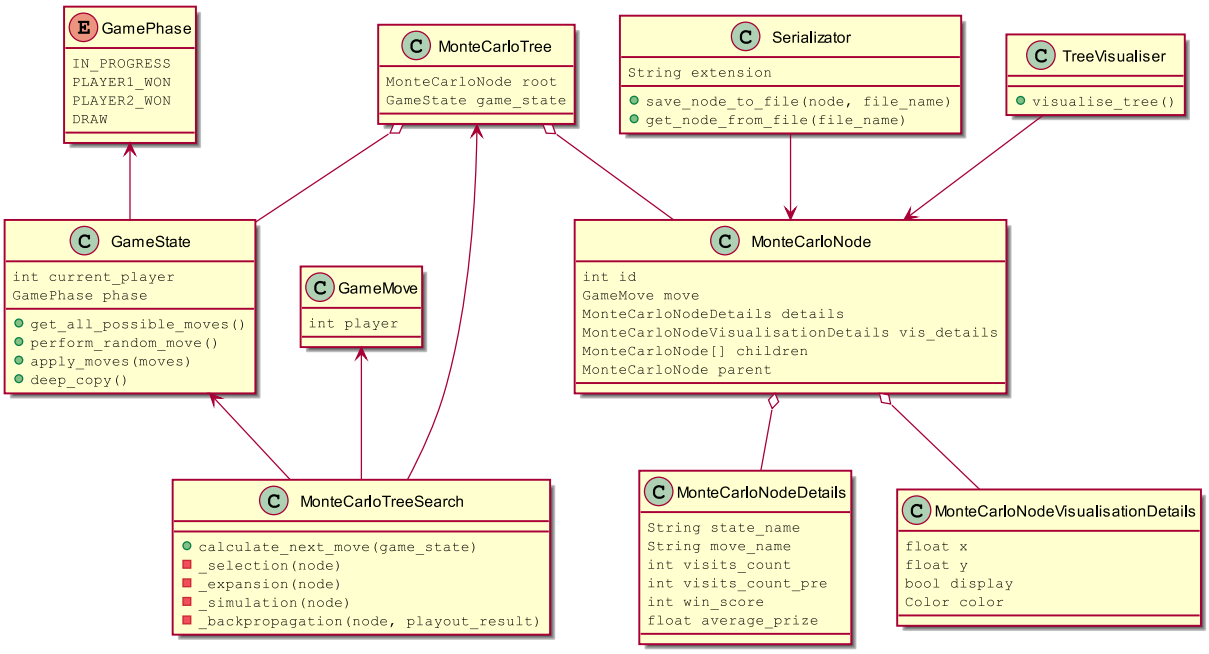
4.1.7. Aplikacja główna

Aplikacja główna jest modułem łączącym wszystkie pozostałe. Ten moduł skupia się na zaprezentowaniu funkcjonalności wszystkich modułów w formie aplikacji okienkowej. Interfejsy, które udostępnia *Aplikacja główna*, są opisane w rozdziale 4.3.

4.2. Główne komponenty aplikacji

4.2.1. Diagram klas

Zgodnie z diagramem 4.1, klasy *MonteCarloTreeSearch*, *TreeVisualiser* oraz *Serializator* są pośrednio lub bezpośrednie zależne od klasy *MonteCarloNode*, opisującej wierzchołek w drzewie.



Rysunek 4.1: Diagram UML głównych komponentów aplikacji

Jest to część wspólna modułów *Algorytm*, *Wizualizacja* i *Serializacja*. Klasa *MonteCarloNode* przechowuje referencję do swojego rodzica oraz wierzchołków potomnych, aby zachować rekurencyjną strukturę drzewa.

Metoda *calculate_next_move* klasy *MonteCarloTreeSearch* odpowiada za wykonanie kolejnych iteracji algorytmu. Algorytm zapisuje informacje o rozgrywanych playoutach w polach klasy *MonteCarloNodeDetails* analizowanych wierzchołków. Ruch oraz stan analizowanej gry są opisane odpowiednio przez klasy *GameMove* i *GameState*. Implementacja metod tych klas daje możliwość łatwego rozszerzenia aplikacji o inne gry. Istotny z punktu widzenia konstrukcji drzewa jest stan rozgrywki, który opisują pola typu wyliczeniowego *GamePhase*.

TreeVisualiser jest głównym komponentem modułu *Wizualizacja*. Jego odpowiedzialnością jest wyznaczenie układu wierzchołków drzewa na płaszczyźnie oraz wyświetlenie wygenerowanej wizualizacji. Szczegóły związane z rysowaniem każdego wierzchołka, takie jak jego współrzędne czy kolor, zawarte są w polach klasy *MonteCarloVisualisationDetails*.

Serializator jest klasą opisującą funkcjonalności, które mają udostępnić właściwe implementacje serializatorów, czyli serializowanie drzew do plików oraz deserializację z plików.

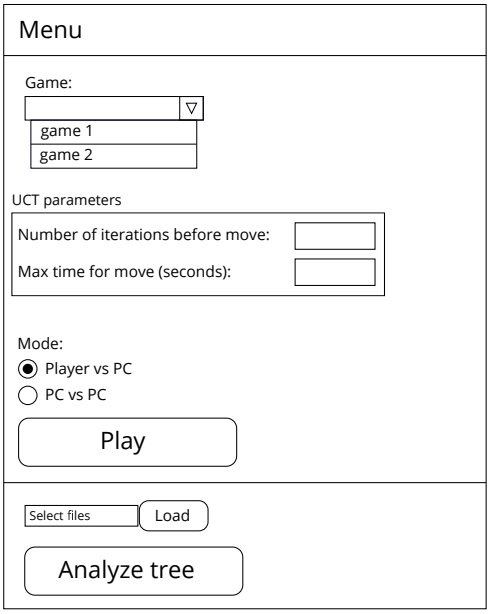
4.3. Interfejs użytkownika

4.3.1. Wstęp

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.

4.3.2. Menu główne

Ukazane na rysunku 4.2 menu główne będzie głównym oknem aplikacji i będzie to pierwsza rzecz, którą zobaczy użytkownik po uruchomieniu programu.



Rysunek 4.2: 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:

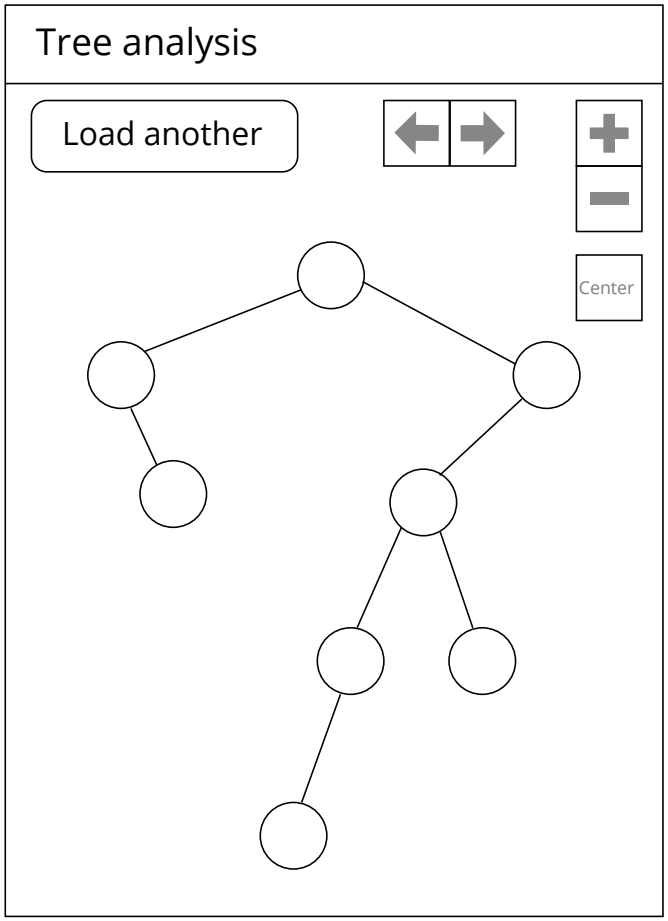
- wybór gry - rozwijana lista, w której znajdować się będą zaimplementowane gry (nasz projekt przewiduje dwa tytuły),
- liczba iteracji (rozgrywek), jaką komputer będzie wykonywał przed wykonaniem ruchu,

- 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).

4.3.3. Analiza drzewa

W oknie ukazanym na rysunku 4.3 będziemy mogli oglądać wczytane lub wygenerowane drzewa.



Rysunek 4.3: Okno analizy drzewa

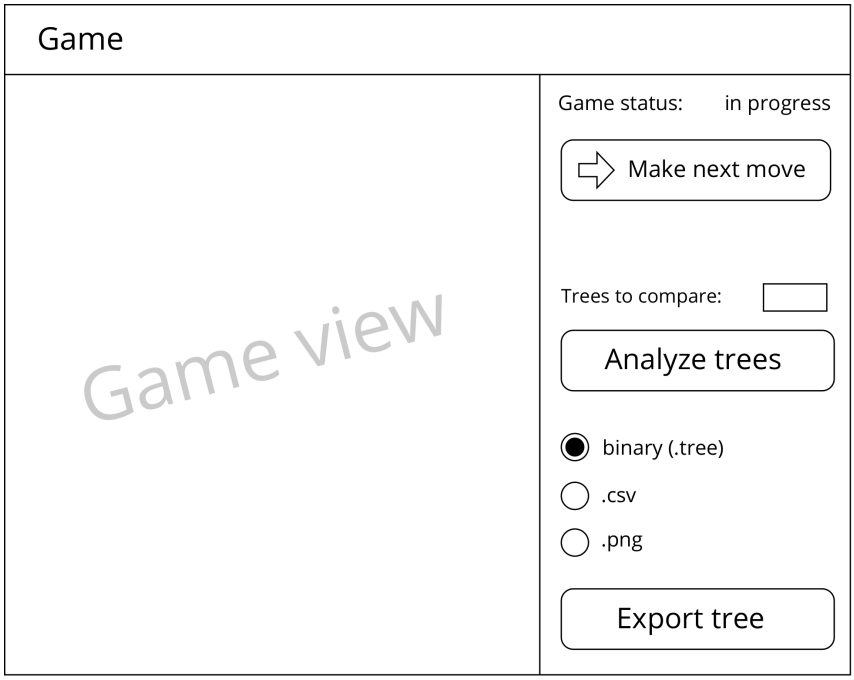
Kluczową funkcją będzie tutaj możliwość dynamicznego przybliżania i oddalania go (za pomocą przycisków plusa i minusa na ekranie lub scrolla) wraz z możliwością klikania poszczegól-

nych 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.

4.3.4. Rozgrywka

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



Rysunek 4.4: Okno rozgrywki

Zgodnie z projektem okna przedstawionym na rysunku 4.4, 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 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

Aplikacja spełnia wszystkie wymagania określone w opisie tematu pracy, jednak w przyszłości mogą zostać do niej dodane usprawnienia. W celu dalszej poprawy jakości i zwiększenia liczby możliwych opcji, obecny produkt można rozszerzyć o następujące funkcjonalności:

1. **Dodanie kolejnych gier** - obecna architektura projektu w dużym stopniu umożliwia rozszerzenie aplikacji o kolejne gry logiczne, które spełniają założenia algorytmu UCT.

2. **Usprawnienie i zrównoleglenie symulacji szachów** - symulacja ruchu szachowego przez komputer zajmuje dużo więcej czasu w przypadku szachów niż mankali. W takim wypadku można pokusić się o zrównoleglenie pewnych czynności szachowych, takie jak znajdowanie wszystkich możliwych dozwolonych ruchów, które zajmują najwięcej czasu. Przy przeprowadzonych próbach okazało się, że znajdowanie , jednakże można by poszukać sposobu implementacji wielowątkowości, który sprawiałby, że symulacja ruchu szachowego przez komputer działała szybciej i wydajniej, jednocześnie nie powodując przy tym dodatkowych błędów.

3. **Dodanie księgi otwarć dla szachów** - algorytm wyznaczający ruch komputera w rozgrywce szachowej

4. **Lepsza ewaluacja wartości figur w szachach** - sposób wartościowania figur jest miejscem, w którym można znacznie rozwinąć potencjał decyzji podejmowanych przez algorytm UCT. Na ten moment każda figura ma arbitralnie ustaloną wartość, bez względu na swoje położenie na szachownicy. Tymczasem, wartość danej figury może zależeć od wielu czynników, między innymi od pozycji, w której się znajduje. Dla przykładu, koń w centrum planszy będzie miał większy potenciał od takiego, który umieszczony jest w rogu. Tak samo pion, który jest bliżej ostatniego rzędu pól, jest bardziej wartościowy od piona

na swojej początkowej pozycji, a pion zdublowany prawdopodobnie jeszcze mniej. Liczba konfiguracji i czynników wpływających na potencjał figury w danym miejscu na szachownicy wykracza poza temat tej pracy, jednak potencjalne rozwinięcie lepiej przemyślanej ewaluacji figur wpłynęło by korzystnie na zdolność algorytmu do podejmowania bardziej obiecujących ruchów.

5. **Inteligentne przydzielanie pamięci dla drzew w sekwencji** - podczas przełączania się pomiędzy kolejnymi drzewami w sekwencji, za każdym razem następuje wczytanie do pamięci wartości i pozycji wszystkich węzłów danego drzewa. Ocznacza to, że jeśli użytkownik analizuje dwa kolejne drzewa i przełącza między nimi na zmianę, za każdym razem te same drzewa są wczytywane do pamięci na nowo. Usprawnieniem tego procesu byłoby przechowywanie wcześniej już wczytanych drzew w pamięci podręcznej, jednocześnie automatycznie kontrolując ilość wykorzystanych zasobów. Co więcej, program mógłby wczytywać od razu pewną liczbę kolejnych drzew z wyprzedzeniem. Takie operacje zaoszczędziłyby czas użytkownika, zwłaszcza podczas wczytywania drzew z dużą ilością węzłów.

Co więcej, w celu usprawnienia modułów, Przepisanie modułów odpowiedzialnych za obliczenia do C/C++

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.

[3] K. Marriott, *NP-Completeness of Minimal Width Unordered Tree Layout*, Journal of Graph Algorithms and Applications, vol. 8, no. 3, pp. 295-312 (2004).

Spis rysunków

Spis tabel

Spis załączników

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