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

Tematem pracy inżynierskiej jest implementacja projektu wizualizacji drzew stanów algorytmu z dziedziny sztucznej inteligencji – Upper Confidence Bound Applied to Trees. Prezentowane rozwiązanie wykorzystuje wspomniany algorytm przy podejmowaniu decyzji podczas rozgrywki w dwie przykładowe gry planszowe - szachy i mankalę.

Kluczową funkcjonalnością prezentowanego systemu jest przejrzysta wizualizacja ukazująca kolejne etapy rozrastania się drzewa stanów. Aplikacja jest wygodnym narzędziem do analizy działania algorytmu w czasie rzeczywistym. Moduł odpowiedzialny za wizualizację, będący najistotniejszym, zawiera implementację usprawnionej wersji algorytmu Walkera. Opisane są również moduły odpowiedzialne za logikę zaimplementowanych gier i sposób ich ewaluacji przez algorytm, implementację algorytmu UCT, a także schematy serializacji generowanych drzew do plików binarnych i CSV. Opis powyższych komponentów i funkcjonalności systemu wzbogacony jest o liczne diagramy UML ilustrujące dane zagadnienie. Przedstawiony jest również obszerny opis interfejsu użytkownika ukazujący najistotniejsze okna aplikacji i przeprowadzający czytelnika przez funkcjonalności udostępniane przez system i sposób ich użycia. Ukazane w dokumencie instrukcje instalacji prowadzą użytkownika przez proces instalacji programu na systemach operacyjnych Windows oraz Ubuntu. Finalnie, opisane zostały wnioski autorów powstałe podczas pracy nad projektem dotyczące wykorzystanych technologii, w tym między innymi przemyślenia o tym, czy język Python sprawdził się pod względem efektywności jako język programowania użyty w aplikacji.

Słowa kluczowe: UCT, MCTS, wizualizacja, sztuczna inteligencja, gry logiczne

Abstract

Visualization of UCT trees

The topic of this engineering diploma project is to create a system that can visualise state trees of the artificial intelligence algorithm – Upper Confidence Bound Applied to Trees. The presented solution uses the UCT algorithm to make decisions while playing two sample board games – chess and mancala.

The essential functionality of the presented system is a precise visualisation showing the subsequent stages of tree growth. The application is a convenient tool for analyzing the algorithm's operations in real-time. The module responsible for visualization, which is the most important, contains an implementation of the Improved Walker's Algorithm. The work also contains descriptions of modules responsible for games' logic and their evaluation methods, implementation of the UCT algorithm, and the serialization schemes of generated trees to binary and CSV files. The components mentioned above and system functionalities descriptions are enriched with numerous UML diagrams illustrating a specific issue. A comprehensive description of the graphical user interface is presented as well, showing the most significant application windows and guiding the reader through the functionalities provided by the system and their methods of usage. The system installation manuals lead the user through the installation process on Windows and Ubuntu operating systems. Finally, there are listed authors' conclusions drawn from the work on the project relating to the used technologies, including thoughts about the effectiveness of Python as a primary application's programming language.

Keywords: UCT, MCTS, visualisation, artificial intelligence, logic games

Warszawa, dnia .....

Oświadczenie

Oświadczam, że moją część pracy inżynierskiej (zgodnie z podziałem zadań opisanym w pkt. 1.3) pod tytułem „Wizualizacja drzewa stanów algorytmu UCT", której promotorem jest mgr inż. Jan Karwowski wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Spis treści

Wykaz najważniejszych oznaczeń i skrótów

- **MCTS** – Monte Carlo Tree Search
- **UCT** – Upper Confidence Bound Applied to Trees
- **CSV** – Comma Separated Values
- **PC** – Personal Computer

1. Założenia projektowe

Algorytm UCT, będący usprawnieniem metody MCTS, jest powszechnie stosowanym algorytmem w sztucznej inteligencji. Analizuje on obiecujące ruchy na podstawie generowanego drzewa, równoważąc eksploatację najbardziej zbadanych z eksploracją mniej zbadanych 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 obiecujących decyzji.

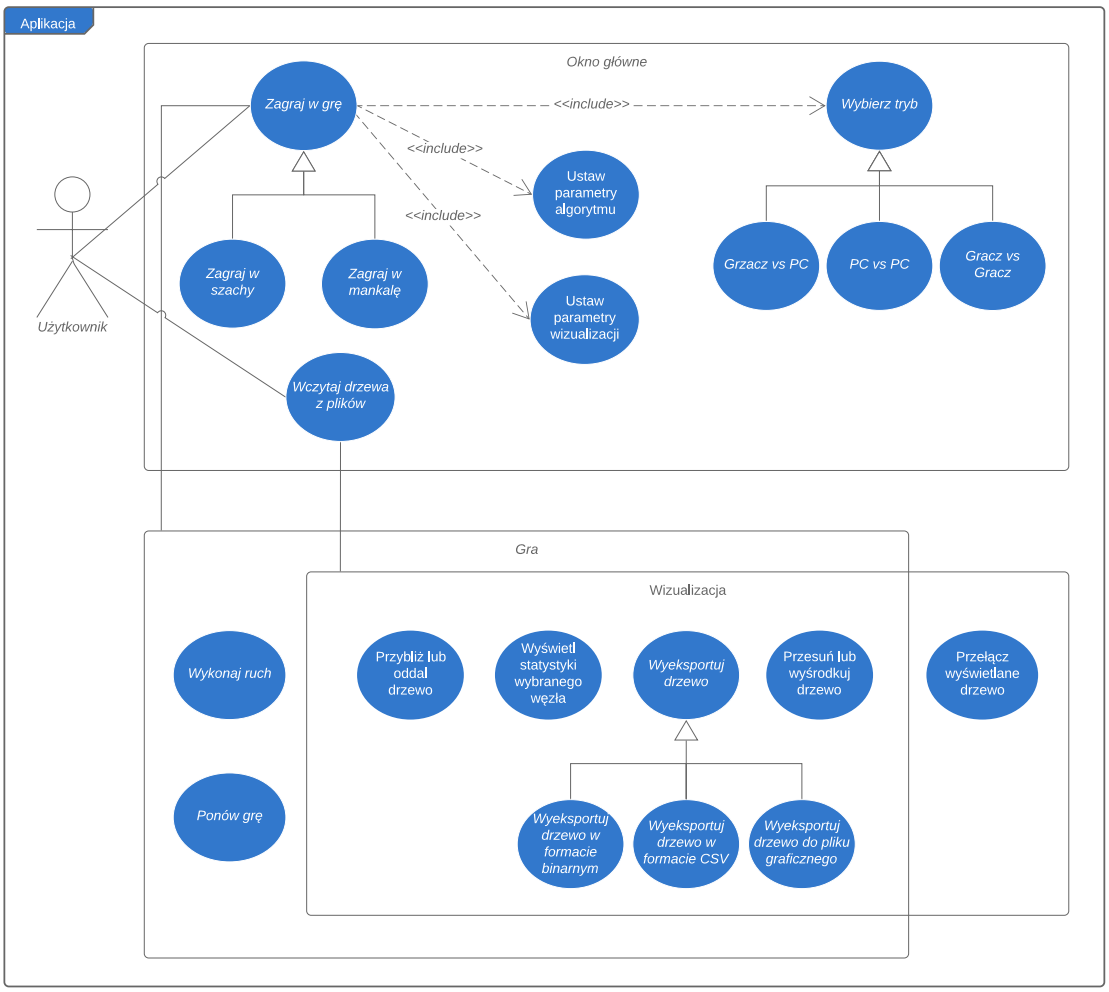
W celu umożliwienia głębszego zrozumienia idei opisywanego zagadnienia postanowiono stworzyć aplikację, która stanowiłaby wygodne narzędzie dla użytkownika zainteresowanego tą tematyką. Pozwalałaby ona na wizualizację drzew stanów algorytmu UCT generowanych podczas rozgrywki w dwie przykładowe gry, co stanowi cel biznesowy projektu.

1.1. Założenia funkcjonalne

Użytkownik korzystający z naszej aplikacji ma do wyboru jedną z dwóch gier i trzy tryby rozgrywki, wymienione poniżej.

1. Gracz versus PC – użytkownik decyduje o swoich posunięciach i mierzy się on z zaimplementowanym algorytmem.
2. PC versus PC – użytkownik jest świadkiem symulacji algorytmu, który rozgrywa partię z samym sobą.
3. Gracz versus Gracz – rozgrywka dwóch graczy, bez wizualizacji.

Co więcej, ma on możliwość ustawienia parametrów algorytmu, takich jak liczbę iteracji podczas tworzenia drzewa, czy też maksymalny czas na ruch przeciwnika. Druga opcja, którą dysponuje użytkownik, to możliwość wczytania plików reprezentujących drzewa w formacie zarówno binarnym jak i CSV, a następnie możliwość jego interaktywnej analizy. Może on wyświetlać informacje na temat wybranego węzła, a także przybliżać i oddalać całą wygenerowaną



Rysunek 1.1: Diagram przypadków użycia

strukturę. Podczas samej rozgrywki, po wykonanym ruchu przeciwnika, gracz może analizować drzewo w sposób opisany powyżej, a także wyeksportować je. Dostępna jest możliwość zapisania go w formatach wymienionych powyżej, a także w formacie rastrowym. Użytkownik może oglądać animację rozrostu drzewa. Powyższe rzeczy dotyczą obu gier w trybach gry z udziałem algorytmu.

Diagram przypadków użycia[1], który ilustruje przedstawione możliwości, znajduje się na Rysunku 1.1.

[1]Grafika stworzona przy użyciu narzędzia na stronie: `https://www.lucidchart.com/` – na zasadzie *free use*.

1.2. Założenia niefunkcjonalne

Wymagania niefunkcjonalne opisane są w Tabeli 1.1. Zawiera ona opis wymagań sprzętowych aplikacji oraz jej założenia wydajnościowe.

Tabela 1.1: Analiza założeń niefunkcjonalnych

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 korzysta głównie z pamięci RAM, procesora i procesora graficznego. Dla drzew do 100 000 wierzchołków wizualizacja nie powinna zajmować więcej niż 3 sekundy, a dla 250 000 — 5 sekund. Wszystkie obliczenia i wizualizacje są przeprowadzane wewnątrz jednej maszyny.
Wsparcie	Aplikacja jest przeznaczona dla komputerów z systemami operacyjnymi Windows oraz Linux opartymi na dystrybucji Debian, na przykład Ubuntu.

1.3. Harmonogram i podział pracy

Podział zaimplementowanych komponentów między autorów systemu opisuje harmonogram z tabeli 1.2, a czasowy podział pracy – harmonogram z tabeli 1.3. Tabela 1.3 nie opisuje zależności między kolejnymi fazami rozwoju projektu, wyznacza jedynie planowe terminy ich zakończenia. Tworząc czasowy podział pracy, autorzy pracy rozłożyli pracę regularnie na cały zaplanowany czas tworzenia projektu. Ponadto, ze względu na skomplikowanie modułu odpowiedzialnego za wizualizację, wydzielono w nim trzy odrębne fazy rozwoju, wymienione poniżej.

- Podstawowa wizualizacja – możliwość wyświetlenia wszystkich wierzchołków drzewa. W podstawowej wizualizacji wygląd wierzchołków jest nieistotny.

Tabela 1.2: Podział pracy

Osoba odpowiedzialna	**Zaimplementowany komponent**
Patryk Fijałkowski	Algorytm UCT Mankala Serializacja Wyświetlanie drzew
Grzegorz Kacprowicz	Usprawniony algorytm Walkera Szachy Graficzny interfejs użytkownika Połączenie algorytmu UCT z grami

Tabela 1.3: Harmonogram pracy

Deadline	**Przygotowane zadania**
24.10.2019	Serializacja Algorytm Pierwsza gra
7.11.2019	Podstawowa wizualizacja Połączenie algorytmu i gry
21.11.2019	Zaawansowana wizualizacja Aplikacja okienkowa Zapis drzew do pliku graficznego
5.12.2019	Pełna wizualizacja Druga gra
19.12.2019	Usprawnienia, poprawki

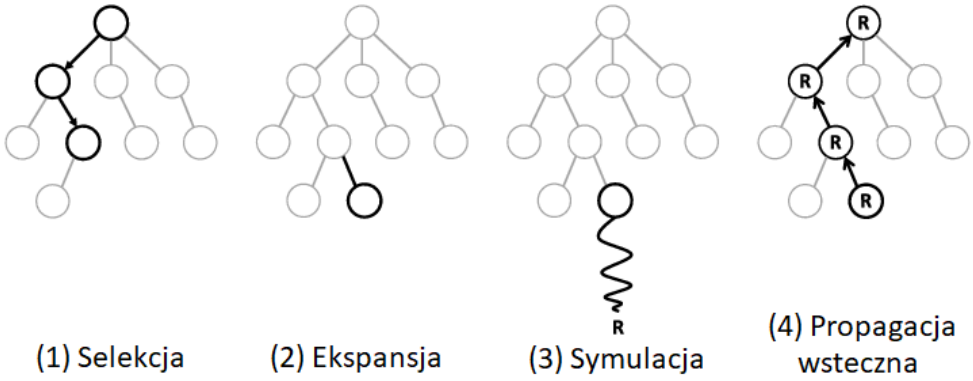
- Zaawansowana wizualizacja – podstawowa wizualizacja wzbogacona o możliwość analizowania statystyk poszczególnych wierzchołków. Kolor wierzchołków będzie reprezentował aktualnego gracza.
- Pełna wizualizacja – zaawansowana wizualizacja wzbogacona o możliwość przewijania, przybliżania oraz oddalania podglądu drzewa.

2. Teoria

Prezentowany system, poza tytułowym algorytmem UCT, wykorzystuje również usprawniony algorytm Walkera do wizualizacji drzew. W rozdziałach 2.1.1 i 2.2 zostały przedstawione zagadnienia teoretyczne z zakresu obu algorytmów.

2.1. Algorytmy MCTS

Monte-Carlo Tree Search to heurystyka, której celem jest podejmowanie decyzji w pewnych zadaniach sztucznej inteligencji, na przykład wybieranie ruchów w grach. Metoda jest oparta na przeszukiwaniu możliwych stanów gry zapisanych w wierzchołkach drzewa i losowym symulowaniu rozgrywek. Algorytmy MCTS opierają się na rozbudowywaniu drzewa ze stanami gry przez iteracyjne wykonywanie czterech faz. Jednym z najpowszechniejszych wariantów MCTS jest algorytm UCT. Pseudokod opisany w Listingu 2.1 oraz implementacja MCTS w projekcie bazują na [1]. Przykład działania algorytmu ze szczególnym uwzględnieniem kolejnych faz znajduje się na Rysunku 2.1, pochodzącym z [2].



Rysunek 2.1: Fazy MCTS

1. **Faza selekcji** (wiersz 7 w listingu) – wybór najbardziej obiecującego wierzchołka do rozrostu drzewa. Istotny w tej fazie jest balans między eksploatacją ruchów przeanalizowanych najdokładniej oraz eksploracją tych jeszcze niezbadanych. Przez eksploatację powinno ro-

zumieć się analizowanie bardziej obiecujących rejonów drzewa, z kolei przez eksplorację — tych rzadko odwiedzanych.

2. **Faza ekspansji** (wiersz 9 w listingu) – utworzenie wierzchołków potomnych dla najbardziej obiecującego wierzchołka drzewa. Tworzone wierzchołki odpowiadają stanom możliwym do uzyskania przez wykonanie jednego ruchu ze stanu wierzchołka obiecującego.

3. **Faza symulacji** (wiersz 11 w listingu) – rozegranie partii składającej się z losowych ruchów ze stanu jednego z wierzchołków utworzonych w poprzedniej fazie. Rozgrywana jest ona do końca, czyli do wyłonienia zwycięzcy lub spowodowania remisu, lub jest ucinana po pewnej liczbie ustalonych ruchów i wynik gry jest ewaluowany przez pewną funkcję.

4. **Faza propagacji wstecznej** (wiersz 13 w listingu) – aktualizacja informacji na temat wierzchołków na ścieżce od wierzchołka-liścia, z którego rozpoczęto symulację, do korzenia drzewa. Główną przekazywaną wartością jest wynik symulacji.

```
1  def find_next_move(curr_state):
2      iterations_counter = 0
3      tree = initialize_tree(curr_state)
4
5      while iterations_counter < max_iterations_counter:
6          # selection(tree.root)
7          curr_node = select promising node
8          # expansion(node)
9          create child nodes from node
10         # simulation(node)
11         playout_result = simulate random playout from curr_node
12         # backpropagation(node, playout_result)
13         update tree according to playout_result
14
15         iterations_counter++
16
17     best_state = select best child(tree.root)
18     return best_state
```

Listing 2.1: Pseudokod algorytmu Monte Carlo Tree Search

2.1.1. Algorytm UCT

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 jako wyrażenie (2.1).

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}} \tag{2.1}$$

W wyrażeniu (2.1), indeks i odnosi się do liczby wykonanych przez algorytm iteracji, czyli czterech faz MCTS. W pierwszym składniku sumy wyrażenia (2.1), 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 (2.1) przyjmuje wartości większe dla wierzchołków, dla których wykonano mniej symulacji i odpowiada eksploracji drzewa. $N_i = \sum_i n_i$, a c jest parametrem eksploracji, który może być dostosowany do badanego problemu.

Stworzone przez nas rozwiązanie dodaje pewne założenia do algorytmu UCT. Dodanie założeń pozwoliło nam na stworzenie rozwiązania bardziej wyspecjalizowanego – takiego, które będzie niezależne 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.

2.2. Algorytm wizualizacji drzew

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

W celu sprawienia, aby wizualizacja była jak najbardziej czytelna, poczyniono 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.

W celu wyznaczenia układu wierzchołków drzewa na płaszczyźnie, który spełnia powyższe 3 założenia, skorzystano z usprawnionego algorytmu Walkera, którego złożoność czasowa jest liniowa względem liczby wierzchołków. Algorytm ten został zaimplementowany i opisany w [4].

3. Implementacja

Aplikacja jest podzielona na pięć oddzielnych modułów: *Algorytm*, *Serializacja*, *Wizualizacja* i *Gry*, które będą funkcjonować w obrębie nadrzędnego modułu – *Aplikacji głównej*. Cele każdego z modułów i zadania powierzone im są przedstawione w rozdziałach 3.2 - 3.9.

3.1. Wykorzystane technologie

W naszym projekcie zdecydowano się skorzystać z technologii wymienionych poniżej.

1. Języka *Python* w wersji 3.7.2, który jest udostępniany na licencji *GNU General Public License*.

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 na podstawie licencji *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* w celu stworzenia pliku instalacyjnego. 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.

3.2. Moduł – 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 2.1 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 Rysunku 4.2.

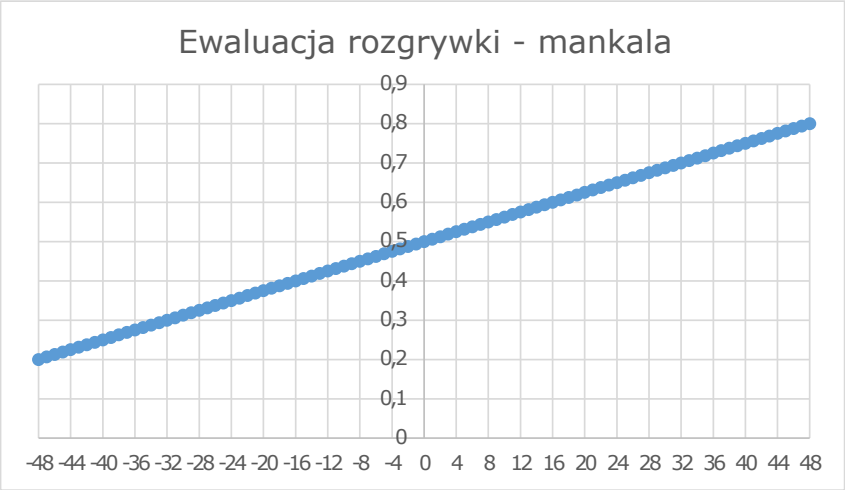
3.3. Ewaluacja rozgrywek

Algorytm dla wygranych gier zwraca wartość 1, dla remisów 0,5, a dla przegranych 0. W celu usprawnienia działania algorytmu, do fazy symulacji została dodana funkcjonalność przerywania rozgrywanych gier po zadanej liczbie ruchów. Po przerwaniu symulacji w fazie propagacji wstecznej przekazana zostaje wartość nagrody wyznaczona przez pewną funkcję, która przyjmuje wartości z przedziału $[0{,}2; 0{,}8]$. Funkcja może przyjmować różne wartości, a jej najważniejszą cechą jest, że dla sytuacji neutralnej zwraca wartość 0,5, tak jak w przypadku remisu.

W przypadku mankali za główny czynnik pozwalający określić wypłatę dla gracza przyjęta została różnica między punktami tego gracza, a punktami jego przeciwnika. Maksymalną liczbą punktów do uzyskania jest 48, a minimalną 0, stąd zakres wypłat wynosi $[-48, 48]$. Zastosowano funkcję liniową, która dla różnicy punktów wynoszącej 0 przyjmuje wartość 0,5. Wzór funkcji został przedstawiony we wzorze (3.1), a jej reprezentację graficzną można ujrzeć na Rysunku 3.1.

$$f(x) = \frac{1}{160}x + \frac{1}{2} \tag{3.1}$$

W przypadku szachów, podobnie jak w mankali, wartość nagrody jest zależna od różnicy materiału graczy. Różnicą względem rozwiązania ewaluacji w mankali jest rozróżnienie wartości



Rysunek 3.1: Wykres funkcji nagrody dla mankali

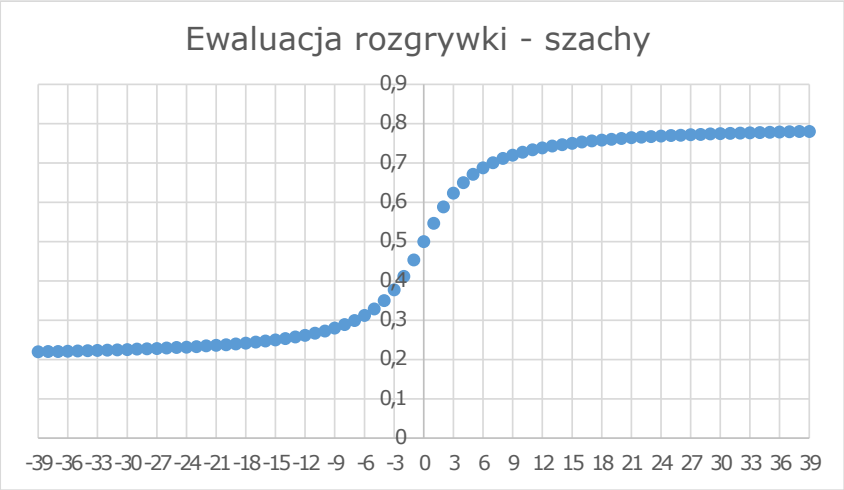
figur. Przyjęte wartości zostały wymienione poniżej.

- Pion – 1 punkt,
- Skoczek – 3 punkty,
- Goniec – 3 punkty,
- Wieża – 5 punków,
- Hetman – 9 punktów.

Oznacza to, że wartość początkowej kolekcji figur na szachownicy dla jednego gracza wynosi 39 punktów. Maksymalna różnica punktów między graczami wynosić może zatem 39 (lub więcej), zatem przyjęty przedział funkcji wyznaczającej wartość wypłaty dla gracza wynosi $[-39, 39]$. Dla różnicy wartości figur wynoszącej 0 również przyjęto, że jest to wartość remisowa 0,5. Zamiast funkcji liniowej zdecydowano się na użycie funkcji $\arctan x$, której wartości w otoczeniu punktu przegięcia $x = 0$ wraz ze wzrostem argumentów znacząco rosną, a wraz ze spadkiem – znacząco maleją. Celem takiego zabiegu jest wypłacanie względnie wysokiej nagrody już dla małej różnicy w materiale (punktach) graczy. Dokładny wzór funkcji przekształcający obraz funkcji $\arctan x$ na $[0{,}2; 0{,}8]$ prezentuje się następująco:

$$f(x) = 0{,}3 \cdot \left[\arctan\left(\frac{x}{4}\right) \cdot \frac{2}{\pi} + 1 \right] + 0{,}2$$

a jej wykres znajduje się na Rysunku 3.2.



Rysunek 3.2: Wykres funkcji nagrody dla szachów

3.4. Moduł – Serializacja

Serializacja jest modułem odpowiadającym za zapisywanie drzew do plików formacie binarnym lub CSV. Pliki z drzewami w formacie binarnym mają umowne rozszerzenie *.tree*. Oba schematy są rekurencyjne, ponieważ taka jest również struktura generowanych przez aplikację drzew. To oznacza, że w celu zapisania całego drzewa, wystarczy przekazać odpowiednim komponentom jego korzeń.

Prezentowany system ma wizualizować również drzewa z licznikami długoterminowymi i krótkoterminowymi generowanymi przez wariant metody MCTS przedstawiony w [5]. Licznik krótkoterminowy będzie w rozdziałach 3.5 i 3.6 nazywany „dodatkowym licznikiem odwiedzin”.

3.5. Serializacja binarna

W serializacji binarnej przyjmujemy opisany niżej schemat.

- **Liczba całkowita** – wartość liczby zakodowanej w U2 przy użyciu 4 bajtów. Bajty liczby w kolejności little endian.
- **Napis** – liczba bajtów w napisie *(liczba całkowita)* i następnie zawartość napisu kodowana w UTF-8.
- **Liczba zmiennoprzecinkowa** – wartość liczby zakodowanej w IEEE754 przy użyciu 64 bitów w kolejności little endian.

Schemat serializowania wierzchołka wykorzystuje wymienione powyżej zasady i ma następującą strukturę:

- 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)*.

3.6. 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 $v_1, v_2, ..., v_n$.

3.7. Moduł – Wizualizacja

Moduł *Wizualizacja* udostępnia funkcjonalność wyświetlania 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 procesorowi graficznemu. Zadania *Wizualizacji* są wymienione poniżej.

- Przypisanie wierzchołkom drzewa miejsca na płaszczyźnie przy użyciu usprawnionego algorytmu Walkera i przetransformowanie wyznaczonych współrzędnych do układu współrzędnych OpenGL.

- Przypisanie krawędziom koloru w zależności od liczby odwiedzin wierzchołka.

- Przypisanie wierzchołkom odpowiedniego koloru. W pracy przyjęto, że jeśli w stanie gry reprezentowanym przez wierzchołek podejmuje decyzję pierwszy gracz, to wierzchołkowi zostanie przypisany kolor biały. Natomiast jeśli wierzchołek będzie odpowiadał stanowi, w którym następuje ruch gracza drugiego, wierzchołkowi zostanie przypisany kolor czarny.

- Detekcja kliknięcia wierzchołka przez użytkownika.

- Wyświetlenie drzewa.

- Przybliżanie, oddalanie i poruszanie się po wizualizacji.

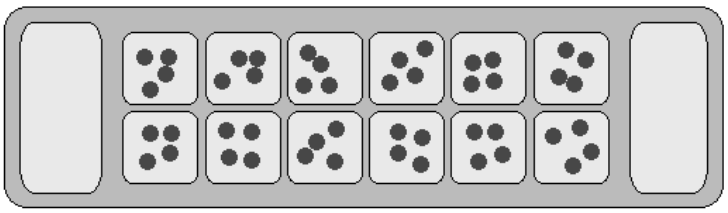
3.8. Moduł – Gry

Prezentowane rozwiązanie udostępnia dwie gry planszowe w ramach modułu *Gry* – szachy oraz mankalę. System został przygotowany z myślą o rozszerzaniu o kolejne gry. Wymagania, które należy spełnić, dodając kolejną grę, są opisane w rozdziale 4.1.

3.8.1. Zasady gry w mankalę

Mankala, jako jedna z przykładowych gier modułu *Gry*, jest mniej popularną grą logiczną od szachów, więc postanowiono przedstawić instrukcję rozgrywki.

Plansza mankali, ukazana na schemacie na Rysunku 3.3, zawiera dwanaście mniejszych pól i dwa większe, nazywane domami lub bazami. Każdemu z graczy przypisane jest sześć mniejszych pól leżących przed nim i dom po jego prawej stronie. Na początku gry w każdym z pól znajdują się cztery kamienie. Celem obu graczy jest zebranie jak największej liczby kamieni w ich domach. Gra kończy się, gdy wszystkie pola jednego z graczy są puste – wtedy pozostałe kamienie przydzielane są drugiemu graczowi.



Rysunek 3.3: Plansza mankali

Ruch gracza polega na wyjęciu wszystkich kamieni z wybranego własnego pola i rozdysponowaniu po jednym do kolejnych pól, omijając dom przeciwnika. Rozdysponowanie jest wykonywane w kierunku odwrotnym do ruchu wskazówek zegara. Ponadto, jeśli ostatni kamień wyląduje we własnym domu – gracz musi wykonać kolejny ruch. W przeciwnym przypadku – następuje ruch przeciwnika.

Ostatnią zasadą mankali jest bicie. Bicie następuje, jeśli po wykonaniu ruchu, ostatni kamień wyląduje na pustym polu gracza. W takiej sytuacji gracz zabiera wszystkie kamienie z przeciwległego pola planszy. Jeżeli w sąsiednim polu nie ma kamieni, nie dochodzi do ich przejęcia. Tak samo, ruch nie kończy się biciem, jeżeli ostatni kamień wyląduje na pustym polu przeciwnika.

3.9. Moduł – Aplikacja główna

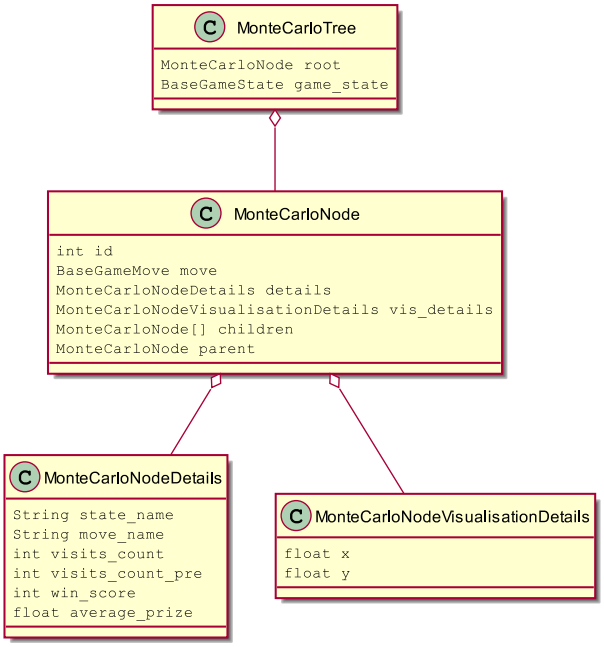
Aplikacja główna jest modułem łączącym wszystkie pozostałe. Ten moduł skupia się na zaprezentowaniu funkcjonalności wszystkich modułów w formie aplikacji okienkowej. Interfejsy graficzne, które udostępnia *Aplikacja główna*, są opisane w rozdziale 5.

4. Główne komponenty aplikacji

W celu poprawnego działania aplikacji niezbędna jest efektywna komunikacja między jej komponentami. Niniejszy rozdział poświęcony jest opisowi procesu komunikacji komponentów, który jest odzwierciedleniem implementacji architektury programu.

4.1. Struktura klas

W celu poprawnego działania aplikacji niezbędna jest efektywna komunikacja między jej komponentami. Niniejszy rozdział poświęcony jest opisowi procesu komunikacji komponentów, który jest odzwierciedleniem implementacji architektury programu.

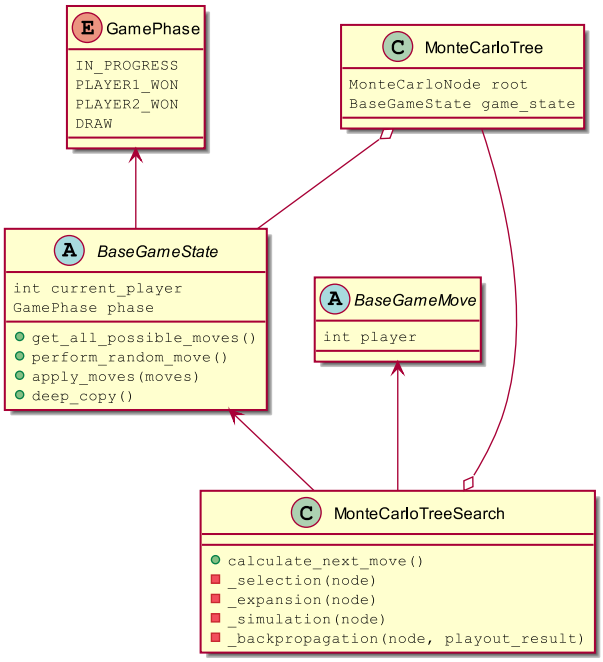


Rysunek 4.1: Diagram UML klasy wierzchołka

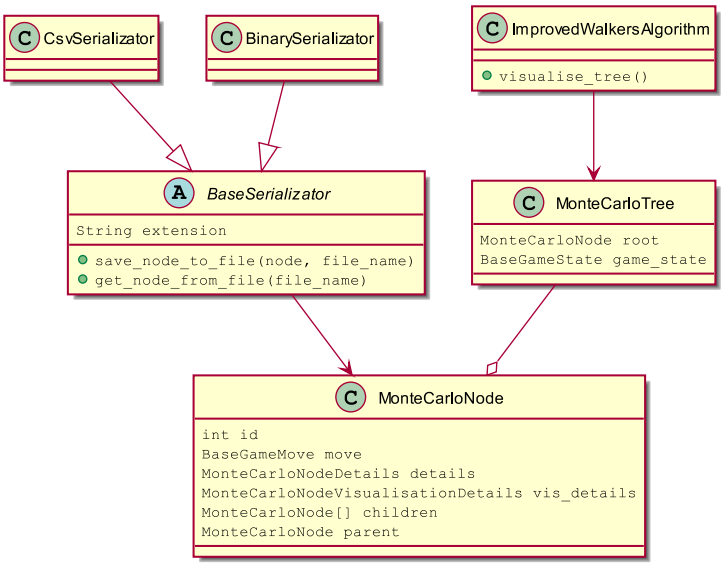
Klasa *MonteCarloNode*, zgodnie z Rysunkiem[1] 4.1, reprezentuje wierzchołek w drzewie, więc przechowuje referencje do swojego rodzica i wierzchołków potomnych, aby zachować

[1]Diagramy na Rysunkach 4.1-4.7 zostały wygenerowane przy użyciu oprogramowania *PlantUML* na licencji *GNU General Public License.*

rekurencyjną strukturę drzewa. Ponadto zawiera wszystkie informacje związane z przebiegiem algorytmu UCT w polu *details* oraz algorytmu Walkera w polu *vis_details*.



Rysunek 4.2: Diagram UML dla modułu *Algorytm*



Rysunek 4.3: Diagram UML dla modułów *Serializacja* i *Wizualizacja*

Diagram ukazany na Rysunku 4.2 ukazuje najistotniejsze klasy modułu *Algorytm*. 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*.

Zgodnie z Rysunkami 4.2 i 4.3, klasy *MonteCarloTreeSearch*, *ImprovedWalkersAlgorithm* oraz *Serializator* są pośrednio lub bezpośrednie zależne od klasy *MonteCarloNode*, opisującej wierzchołek w drzewie. Jest to część wspólna modułów *Algorytm*, *Wizualizacja* i *Serializacja*.

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

4.2. Struktura stanów rozgrywki

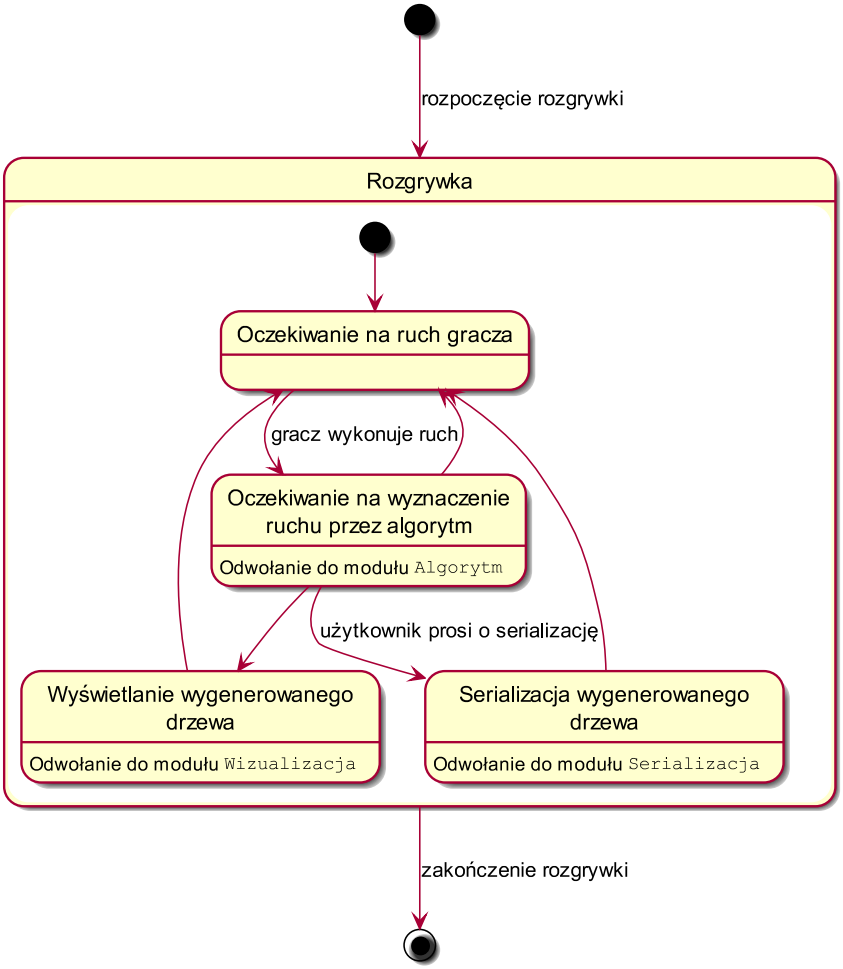
Rysunek 4.4 ukazuje diagram stanów aplikacji w przypadku rozgrywki w trybie *Gracz versus PC*. Zgodnie z diagramem, aplikacja po zainicjowaniu rozgrywki przechodzi do stanu *Rozgrywka* zawierającego cztery stany wewnętrzne. Będąc w stanie *Rozgrywka*, aplikacja może korzysta z modułów *Algorytm* i *Wizualizacja*, a także opcjonalnie z modułu *Serializacja*.

Istotna z punktu widzenia użytkownika jest wizualizacja drzewa stanów bezpośrednio po ruchu wyznaczonym przez algorytm, co powoduje przejście aplikacji odpowiednio w stany *Serializacja wygenerowanego drzewa* oraz *Wyświetlanie wygenerowanego drzewa*, a także możliwość serializacji i zapisania powstałego drzewa – stan *Serializacja wygenerowanego drzewa*.

4.3. Struktura sekwencji rozgrywki

Rysunek 4.5 ukazuje diagram sekwencji rozgrywki w trybie *Gracz versus PC*. Na diagramie przedstawiona jest istota komunikacji głównych modułów podczas rozgrywania partii w jedną z dwóch gier. *Aplikacja główna*, *Gra* i *Algorytm*.

Użytkownik końcowy przy użyciu menu aplikacji głównej może ustawić parametry gry i uruchomić ją. Inicjalizowana jest wówczas rozgrywka w komponencie *Gra*. Następnie, dopóki gra trwa i możliwe jest wykonanie ruchu, wykonywane są na zmianę ruchy gracza i PC – wymaga to komunikacji odpowiednio użytkownika z aplikacją główną, aplikacji głównej z grą



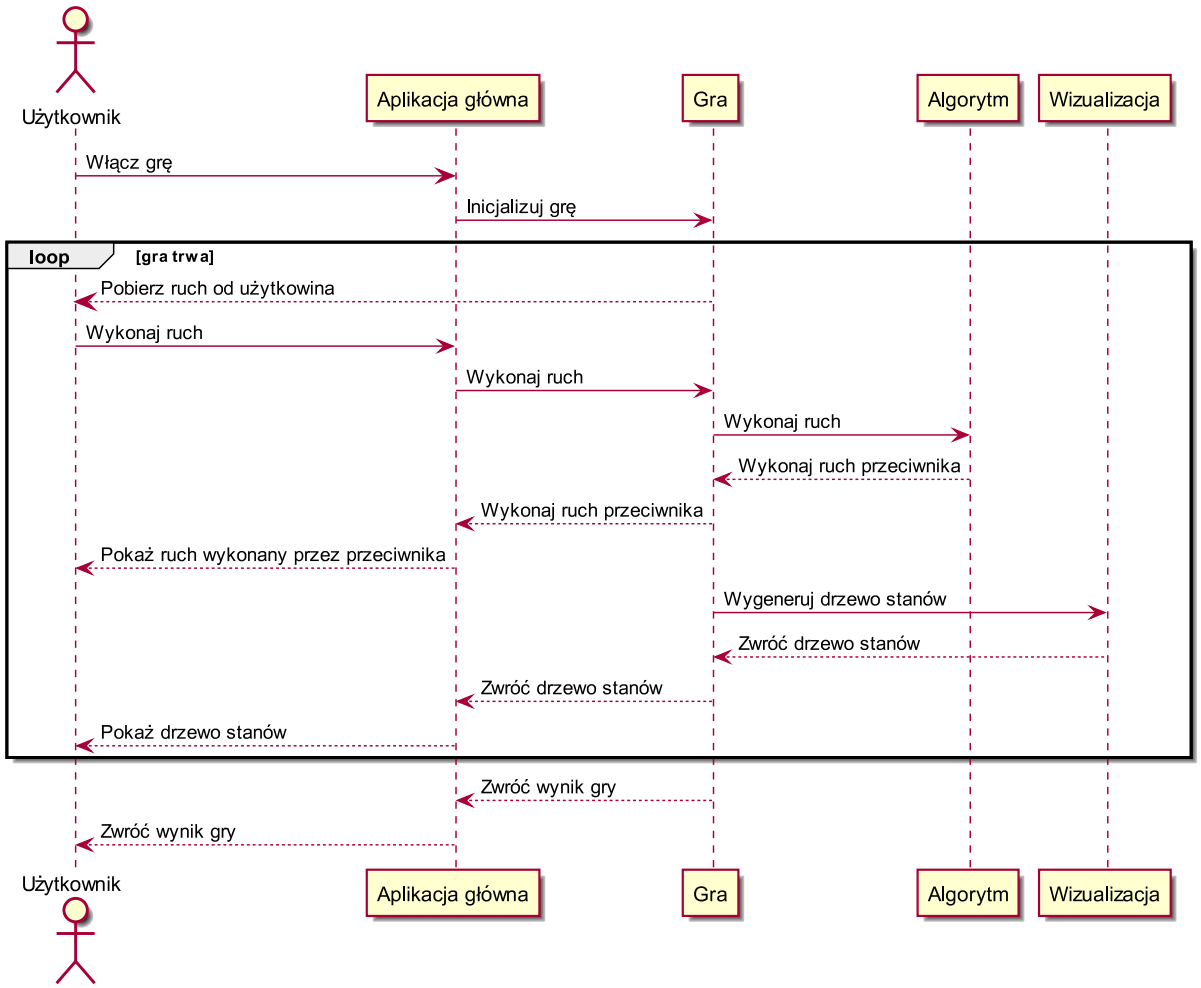
Rysunek 4.4: Diagram stanów aplikacji

i gry z modułem *Algorytm* (i na odwrót). Po zakończeniu rozgrywki gra zwraca swój stan, który jest możliwy do zobaczenia przez użytkownika w oknie aplikacji głównej. Dodatkowo, po każdym wykonanym ruchu, przy użyciu modułu *Wizualizacja* generowane jest drzewo stanów algorytmu UCT, a następnie pokazywane jest ono użytkownikowi.

Diagram ukazuje, że w tym trybie każdy ruch gracza jest ściśle związany z odpowiedzią od modułu *Algorytm*, który pobiera stan rozgrywki z modułu *Gra*.

4.4. Struktura sekwencji eksportu drzewa

Rysunek 4.6 przedstawia proces współpracy różnych komponentów aplikacji w celu wyeksportowania wygenerowanego przez algorytm drzewa podczas rozgrywki. Proces uruchamiania gry i wykonywania ruchów jest analogiczny do tego na Rysunku 4.5. Jest to diagram dla trybu



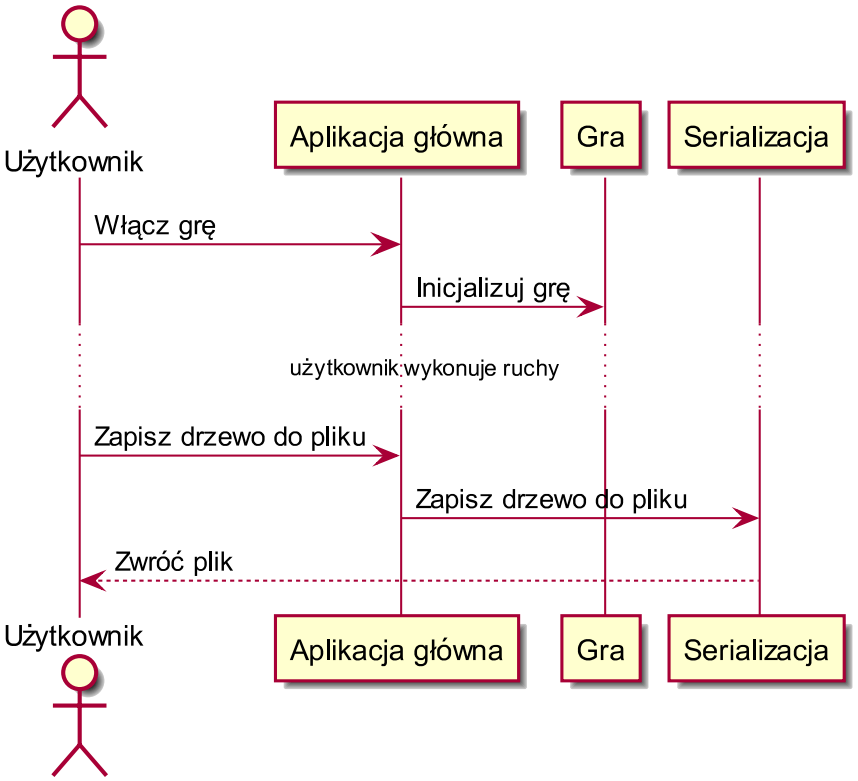
Rysunek 4.5: Diagram sekwencji rozgrywki

rozgrywki *Gracz versus PC*, jednak w przypadku ustawienia *PC versus PC* sposób serializacji się nie zmienia i diagram wygląda identycznie.

Istotną cechą zaprojektowanego rozwiązania jest to, że gracz może wyeksportować drzewo w dowolnym momencie rozgrywki (po każdym ruchu przeciwnika). Żądanie takiej operacji przez użytkownika przesyłane jest do aplikacji głównej, która następnie komunikuje się z modułem odpowiedzialnym za serializację, który zapisuje drzewo do pliku. Plik drzewa zapisywany jest do wybranego przez użytkownika folderu.

4.5. Struktura sekwencji wizualizacji drzewa

Rysunek 4.7 przedstawia proces wyświetlania wizualizacji drzewa przez użytkownika jako współpracę poszczególnych komponentów aplikacji. Ponownie, proces uruchamiania gry i

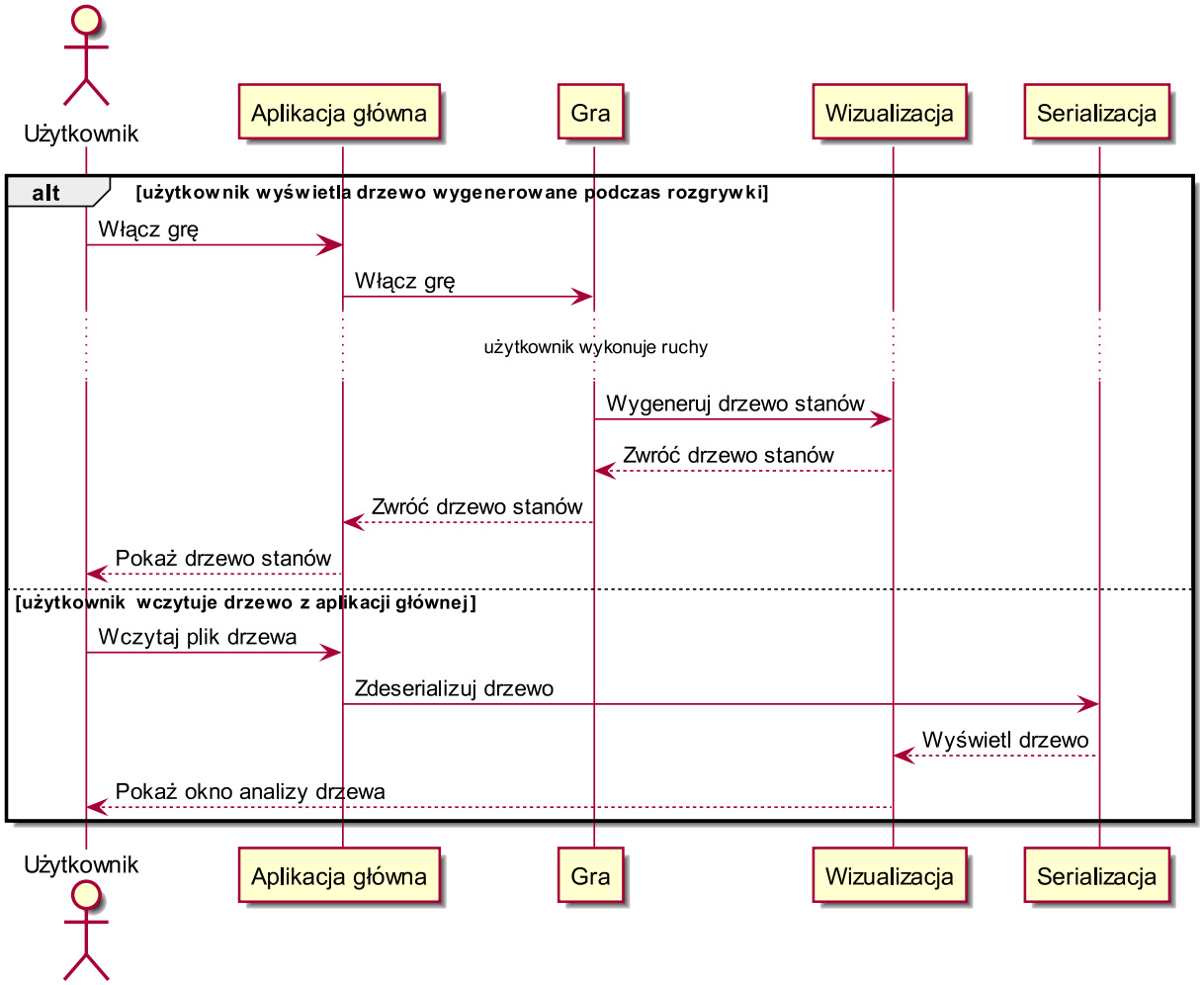


Rysunek 4.6: Diagram sekwencji eksportu drzewa

wykonywania ruchów wygląda tak jak na Rysunku 4.5.

Istotny jest fakt, że użytkownik może oglądać drzewa stanów wybierając pliki drzew z dysku w menu głównym aplikacji, ale też oglądać je bezpośrednio podczas rozgrywki po wykonanym ruchu algorytmu. Gdy żądanie jest z poziomu rozgrywki, komponent *Aplikacja główna* komunikuje się z komponentem *Wizualizacja*, który generuje aktualne drzewo i pokazuje je użytkownikowi na ekranie.

Drugi sposób, czyli wczytanie plików drzew z menu głównego, wymaga wcześniejszego zdeserializowania ich w celu wyświetlenia – wymaga to komunikacji modułu *Wizualizacja* i *Serializacja*, gdzie ten drugi będzie zwracał wynik deserializacji temu pierwszemu. Następnie, analogicznie, użytkownik będzie mógł zobaczyć okno z wygenerowanymi drzewami.



Rysunek 4.7: Diagram sekwencji wizualizacji drzewa

5. Interfejs użytkownika

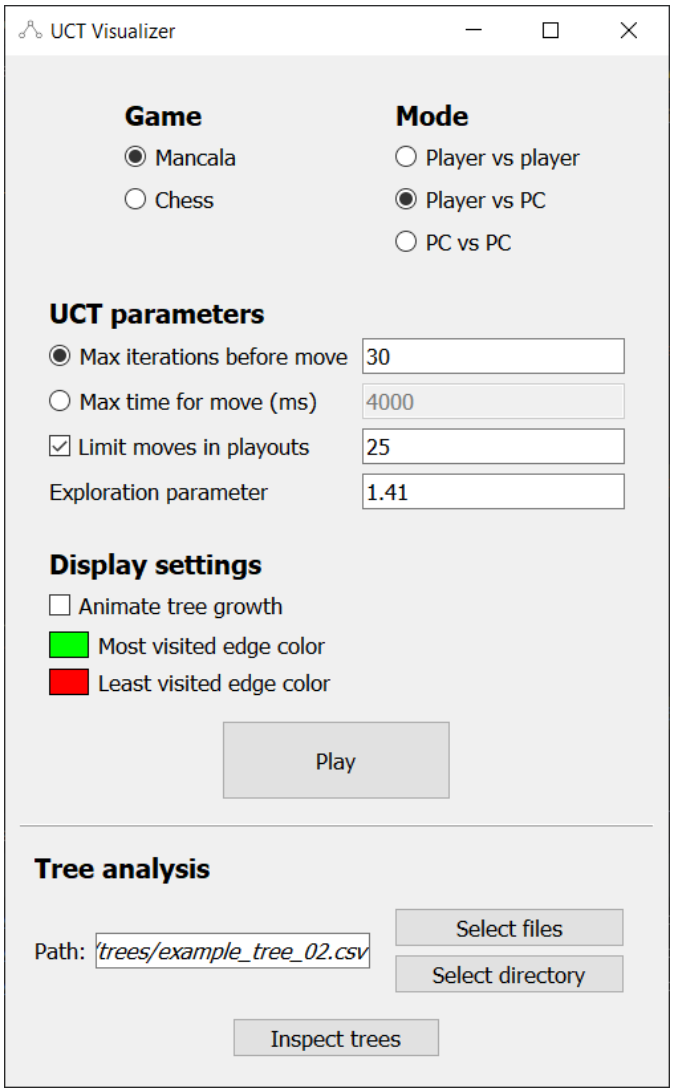
Graficzny interfejs użytkownika składa się z trzech głównych okien, a logika jego działania jest w całości zawarta w module *Aplikacja główna*. Zadaniem graficznego interfejsu jest umożliwienie uruchomienia poszczególnych modułów użytkownikom końcowym.

5.1. Menu główne

Ukazane na Rysunku 5.1 menu główne jest głównym oknem aplikacji i jest to pierwsze okno ukazywane użytkownikowi po uruchomieniu programu.

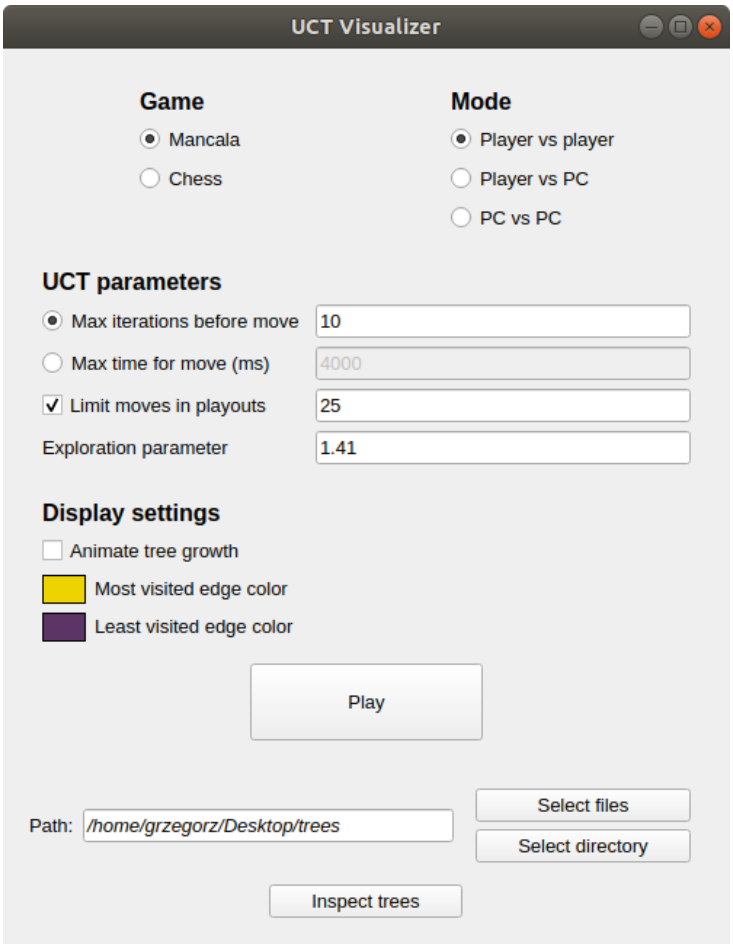
Z poziomu *Menu głównego* użytkownik może przejść do okien *Rozgrywka* i *Analiza drzewa*. W celu rozegrania gry, należy nacisnąć przycisk *Play*. Powyżej przycisku *Play* znajdują się opcje, które pozwolą użytkownikowi ustawić parametry gry dostosowane do jego preferencji.

- **Wybór gry** – do dyspozycji gracza oddane są dwie gry – szachy i mankala.
- **Wybór trybu rozgrywki** – użytkownik może wybrać jeden z trzech trybów gry, opisanych poniżej.
  - Gracz versus PC – w tym trybie gracz ma okazję zmierzyć się z komputerem, oglądając przy tym generowane drzewa stanów algorytmu UCT.
  - PC versus PC – rozgrywka komputera z samym sobą, z możliwością analizowania wynikowych drzew stanów algorytmu. Wybór tego trybu oznacza brak możliwości wykonywania samodzielnych ruchów przez użytkownika, jednakże to od niego będzie zależało, kiedy algorytm wykona kolejny ruch – będzie miał do dyspozycji przycisk, przy użyciu którego będzie mógł powodować postęp w rozgrywce.
  - Gracz versus Gracz – tryb bez udziału algorytmu UCT, a co za tym idzie, również bez wizualizacji.
- **Wybór liczby iteracji algorytmu** – liczba powtórzeń wykonania przez algorytm czterofazowej iteracji metody MCTS opisanej w rozdziale 2.1.



Rysunek 5.1: Okno menu głównego – Windows

- **Wybór maksymalnego czasu** – podany w milisekundach; czas, po którym komputer będzie przerywał obliczenia i wykona ruch.
- **Ustawienie limitu ruchów** – opcjonalne pole, w którym użytkownik może ustalić maksymalną liczbą ruchów w fazie symulacji algorytmu UCT, w celu wcześniejszego kończenia niepożądanie długich rozgrywek.
- **Ustawienie wartości parametru eksploracji** – pole, w którym użytkownik może ustalić wartość parametru eksploracji algorytmu UCT. Wpływ parametru na działanie algorytmu został opisany w rozdziale 2.1.1.
- **Animowanie rozrostu drzewa** – wybór tej opcji sprawia, że drzewo stanów algorytmu zostaje wyświetlone nie tylko po każdym wykonanym ruchu przez komputer, ale też po każdej iteracji. Co więcej, w panelu po prawej stronie dynamicznie aktualizować się też



Rysunek 5.2: Okno menu głównego – Linux

będzie wartość liczby wszystkich wierzchołków drzewa.

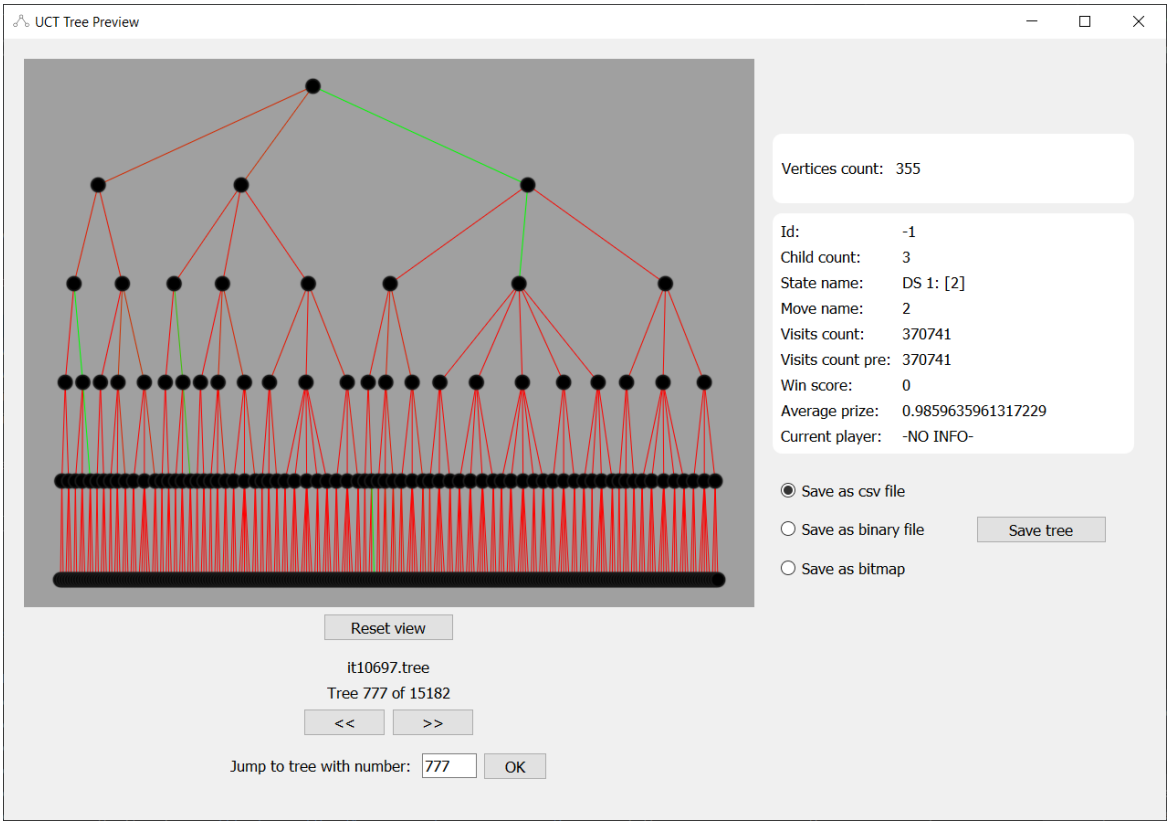
- **Wybór kolorów krawędzi drzewa** – po kliknięciu na któryś z widocznych kolorów ukaże się paleta systemowa, z której można wybrać inne kolory krawędzi. Najczęściej i najrzadziej odwiedzane węzły drzewa przyjmą wskazane przez użytkownika kolory, inne zaś — kolory z ich gradientu — proporcjonalnie do wartości skrajnych licznika odwiedzin.

Drugą kluczową opcją dostępną z menu głównego aplikacji jest możliwość przejścia do okna analizy drzewa lub sekwencji drzew. Jest ona dostępna po naciśnięciu przycisku *Inspect trees*. Użytkownik może wczytać pliki z dysku na dwa sposoby: wybierając pliki z folderu bezpośrednio lub wybierając cały katalog, w którym się one znajdują. Do tego celu służą osobno dwa przyciski: *Select files* i *Select directory*, które znajdują się na prawo od pola tekstowego, w którym wyświetlana jest pełna ścieżka do aktualnie wybranego pliku lub katalogu. W przypadku wybrania większej liczby plików, pokazywana jest ich liczność. Można jednocześnie wybierać pliki w formacie CSV i binarnym (.tree), których schemat serializacji został opisany w rozdziale 3.4. W przypadku wybrania nieprawidłowego katalogu lub niepoprawnych plików, na ekranie

użytkownika pojawi się błąd informujący o problemie, który nastąpił.

5.2. Analiza drzewa

W oknie ukazanym na Rysunku 5.3 będziemy mogli oglądać i analizować drzewa wczytane z plików. Ekran podglądu drzewa pozwala w wygodny sposób analizować i porównywać wczytane drzewa oraz udostępnia opcje wymienione poniżej.



Rysunek 5.3: Okno analizy drzewa

1. **Wyświetlenie statystyk węzła** – po naciśnięciu na węzeł lewym przyciskiem myszy, panel z prawej strony zostaje uzupełniony informacjami, które przechowuje węzeł. Dokładność wykrywania wybranego węzła nie jest zaburzona podczas operowania na przybliżonym lub oddalonym drzewie.

2. **Przybliżanie i oddalanie widoku** – należy naprowadzić kursor na szary obszar drzewa, a następnie przewijać środkowym przyciskiem myszy w górę lub w dół.

3. **Przesuwanie drzewa** – gdy kursor znajduje się w obszarze drzewa, należy nacisnąć prawy przycisk myszy, przytrzymać i poruszać nim na boki w celu przesunięcia widoku całego

drzewa.

4. **Reset pozycji drzewa** – pierwotną pozycję i rozmiar drzewa można przywrócić za pomocą przycisku *Reset view* znajdującego się pod widokiem drzewa.

5. **Zmiana drzewa w sekwencji** – jeżeli wczytane zostało więcej niż jedno drzewo, kluczową funkcją jest opcja wygodnego przełączania się między nimi. Można to zrobić na trzy sposoby: używając przycisków „«" i „»", które znajdują się w dolnej części okna; klikając lewy i prawy przycisk myszy na klawiaturze (czasami trzeba ustawić uprzednio uaktywnić obszar drzewa, klikając na niego) oraz użycie pola tekstowego, pozwalającego na zmianę aktualnie wyświetlanego drzewa i kliknięcie przycisku *OK*. W przypadku wyświetlania pierwszego lub ostatniego drzewa w sekwencji odpowiednie przyciski strzałek stają się nieaktywne, a w przypadku pola tekstowego ze skokiem następuje walidacja, czy podany numer drzewa mieści się w zakresie.

6. **Zapis drzewa** – możliwość zapisu wyświetlanego drzewa w jednym z trzech dostępnych formatów: CSV, binarnym lub rastrowym (.png). Opcja ta znajduje się z prawej strony okna, pod panelem zawierającym wartości przechowywane przez węzeł. Podczas eksportu do grafiki rastrowej zadbano o to, aby nie występowały artefakty związane z dużym zagęszczeniem krawędzi. Taka grafika zachowuje też aktualne powiększenie i przesunięcie drzewa – działa na zasadzie zrzutu ekranu.

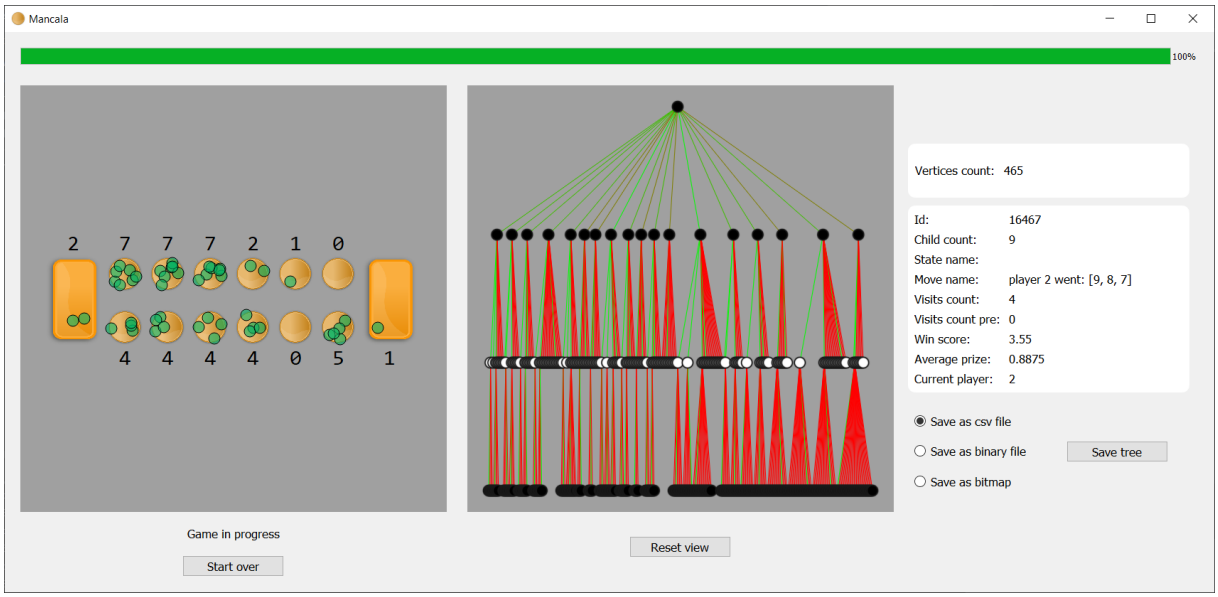
Co więcej, dla każdego drzewa w sekwencji w panelu dolnym wyświetla się informacja o tym, którym z kolei na liście jest oglądane drzewo oraz jaka jest nazwa pliku, z którego zostało ono wczytane. Dodatkowo, wyświetlana jest liczba wszystkich wczytanych drzew.

5.3. Rozgrywka

Na Rysunkach[1] 5.4 i 5.5 przedstawiony jest interfejs graficzny podczas rozgrywki. Środkową i prawą część okna zajmuje komponent dotyczący analizy drzewa, który został opisany w rozdziale 5.2. W związku z tym, wszystkie wyżej opisane funkcjonalności mają zastosowanie również w oknie rozgrywki, poza aspektem dotyczącym sekwencji drzew. Z lewej strony znajduje się panel z wybraną wcześniej grą. Funkcjonalności okna *Rozgrywka* zostały opisane poniżej.

1. **Wykonywanie ruchów** przez gracza, kiedy gra przeciwko PC:

[1] Źródło grafik wykorzystanych w mankali: `https://www.wpclipart.com/` i w szachach: `https://icons8.com/`, na zasadzie *free use*.



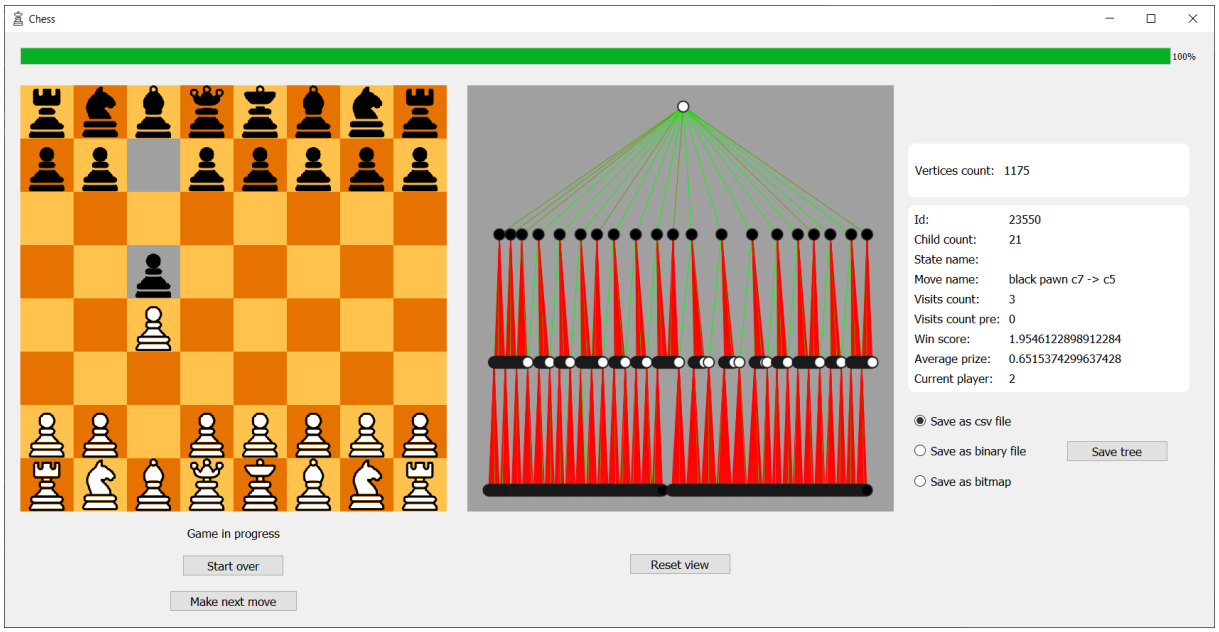
Rysunek 5.4: Okno rozgrywki – mankala, gracz przeciwko PC

- w przypadku szachów – wybierając odpowiednią figurę na szachownicy, a następnie jeden z wyświetlonych dostępnych ruchów. Gracz w tym trybie zawsze gra białymi,
- w przypadku mankali – wybierając pewien niepusty dołek z dolnego rzędu planszy.

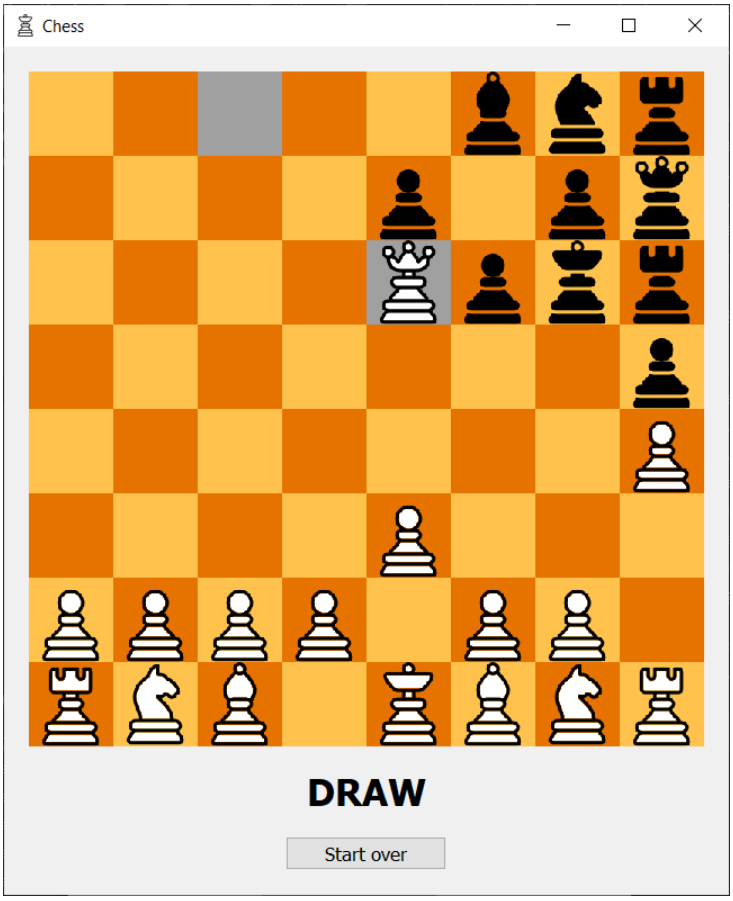
2. **Zrestartowanie gry** przy użyciu przycisku *Start over* w lewej dolnej części okna,

3. **Zapisanie drzewa do pliku** przy użyciu przycisku *Save tree*. Format zapisywanego pliku wybierany jest przez zaznaczenie odpowiedniej opcji w liście znajdującej się na lewo od przycisku *Save tree*.

Bezpośrednio pod obszarem gry znajduje się etykieta z napisem dotyczącym obecnego stanu rozgrywki. *Game in progress* oznacza, że gra się toczy i wciąż można wykonywać ruchy. W przypadku zakończenia rozgrywki, obszar gry lub przycisk *Make next move* zostaje dezaktywowany i nie można wykonywać kolejnych ruchów aż do ponowienia gry, a etykieta będzie informować o tym, który gracz zwyciężył, wyświetlając napis *Player 1 wins*, *Player 2 wins* lub *Draw*. Przykładowy stan zakończonej gry pokazany jest na Rysunku 5.6.



Rysunek 5.5: Okno rozgrywki – szachy, PC przeciwko PC



Rysunek 5.6: Okno rozgrywki – szachy, dwóch graczy

6. Instrukcja instalacji

W celu wykorzystania możliwości, które daje prezentowany system, należy uruchomić go na komputerze, który spełnia wymienione poniżej wymagania sprzętowe.

1. Procesor: Intel Core i5-3470 3.2 GHz / AMD FX-8350 4 GHz.
2. Pamięć RAM: 8 GB.
3. Karta graficzna: Nvidia GTX 660 2GB / AMD HD 7870 2 GB.
4. Miejsce na dysku twardym: 150 MB.
5. System operacyjny: Windows 10 / Ubuntu 16.04.

W celu zainstalowania aplikacji na dysku lokalnym należy rozpakować jedno z dostarczonych archiwum. Prezentowany system został zbudowany w formie aplikacji przenośnej, więc wszystkie komponenty potrzebne do uruchomienia aplikacji są dostarczone razem z nią. W zależności od systemu operacyjnego urządzenia, należy wybrać odpowiednio:

1. dla urządzeń z systemem Windows - *UCTVisualisation-portable.zip*,
2. dla urządzeń z systemem Linux - *UCTVisualisation-portable.tar*.

Aplikacja znajduje się w rozpakowanym katalogu o nazwie *UCTVisualisation* i aby ją włączyć, należy uruchomić odpowiedni plik w rozpakowanym katalogu. W zależności od systemu operacyjnego urządzenia, należy wybrać odpowiednio:

1. dla urządzeń z systemem Windows - *UCT Visualisation.exe*,
2. dla urządzeń z systemem Linux - *UCT Visualisation*.

7. Podsumowanie i ocena

Dostarczany produkt jest sprawnie działającym i wygodnym narzędziem do wizualizowania i analizowania drzew stanów algorytmu UCT. Pozwala on również na ich generowanie podczas rozgrywki przeciwko komputerowi w jedną z dwóch zaimplementowanych gier logicznych. Dodatkowo, wszystkie opisywanie wcześniej funkcjonalności spaja intuicyjny i uporządkowany interfejs graficzny.

7.1. Testy akceptacyjne

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

Testy akceptacyjne zostały wykonane na komputerze:

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

Poniżej przedstawiony jest opis pięciu testów wraz z uzyskanymi wynikami.

1. - **Testowany moduł**: *Gra*
   - **Testowane wymaganie**: Użytkownik będzie mógł wybrać jedną z dwóch przykładowych gier, a do wyboru będzie miał trzy tryby rozgrywki.
   - **Kroki testowe**:
     – Z menu głównego aplikacji wybierz opcję *Chess*.

  - Z menu głównego aplikacji wybierz opcję *Player vs Player* i sprawdź tryb rozgrywki dla dwóch graczy.
  - Z menu głównego aplikacji wybierz opcję *Player vs PC* dla różnych ustawień algorytmu UCT.
  - Z menu głównego aplikacji wybierz opcję *PC vs PC* dla różnych ustawień algorytmu UCT.
  - Z menu głównego aplikacji wybierz *Mancala* i powtórz kroki 2–5.
- **Wynik**: Pozytywny.

2. - **Testowany moduł**: *Serializacja*
   - **Testowane wymaganie**: Użytkownik będzie mógł zapisać analizowane drzewa do pliku csv, do pliku binarnego oraz do bitmapy.
   - **Kroki testowe**:
     - Z menu głównego aplikacji wybierz ścieżkę do dowolnego pliku z zserializowanym drzewem.
     - Naciśnij przycisk *Inspect tree*.
     - Naciśnij przycisk *Save to csv file*.
     - Naciśnij przycisk *Save to binary file*.
     - Naciśnij przycisk *Save to bitmap file*.
     - Sprawdź, czy bitmapa wygenerowana w kroku 5 odpowiada drzewu z pliku początkowego.
     - 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.

3. - **Testowany moduł**: *Wizualizacja*
   - **Testowane wymaganie**: Użytkownik będzie mógł wyświetlić informacje związane z wybranym węzłem drzewa, a także przybliżać i oddalać cały graf.
   - **Kroki testowe**:
     - Z menu głównego aplikacji wybierz ścieżkę do dowolnego pliku z drzewem.
     - Naciśnij przycisk *Inspect tree*.

  - Przy użyciu prawego przycisku myszy chwyć za obszar rysowania i poruszaj się po wizualizacji.
  - Używając środkowego przycisku myszy, przybliż i oddal wizualizowane drzewo.
  - Kliknij dowolny wierzchołek drzewa lewym przyciskiem myszy i sprawdź, czy panel z prawej strony wyświetla informacje związane z wybranym wierzchołkiem.
- **Wynik**: Pozytywny.

4.
- **Testowany moduł**: *Wizualizacja*
- **Testowane wymaganie**: Dla drzew do 100000 wierzchołków wizualizacja nie powinna zajmować więcej niż 3s.
- **Kroki testowe**:
  - Z menu głównego aplikacji wybierz ścieżkę do pliku *tree_100k.csv*.
  - Naciśnij przycisk *Inspect trees*.
- **Wynik**: Pozytywny – deserializacja, ulepszony algorytm Walkera i wyświetlenie drzewa z pliku zajęło 2.802s.

5.
- **Testowany moduł**: *Wizualizacja*
- **Testowane wymaganie**: Dla drzew do 250000 wierzchołków wizualizacja nie powinna zajmować więcej niż 5s.
- **Kroki testowe**:
  - Z menu głównego aplikacji wybierz ścieżkę do pliku *tree_250k.csv*.
  - 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.

7.2. Kontynuacja pracy

Aplikacja spełnia wszystkie początkowe wymagania określone w rozdziale **??**, 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 umożliwia rozszerzenie aplikacji o kolejne gry logiczne, które spełniają założenia algorytmu UCT. Zwiększenie liczby dostępnych gier poszerzyłoby możliwości dokładniejszego badania algorytmu UCT.

2. **Usprawnienie i zrównoleglenie symulacji szachów** – symulacja ruchu szachowego przez komputer zajmuje dużo więcej czasu w przypadku szachów niż mankali. Zrównoleglenie pewnych obliczeń szachowych, które zajmują najwięcej czasu, poprawiłoby jakość decyzji podejmowanych przez algorytm w jednostce czasu.

3. **Dodanie otwarć szachowych** – algorytm UCT dla niewielkiej liczby iteracji ma tendencję do wybierania ruchów dających natychmiastowe wynagrodzenie, tym samym ignorując ruchy, które niebezpośrednio prowadzą do lepszych, złożonych zagrań. To oznacza, że algorytm podejmuje lepsze decyzje w końcowym stadium rozgrywki, niż w początkowym — szczególnie dobrze widać to w przypadku rozgrywki szachowej. Możliwe byłoby zatem wprowadzenie usprawnienia w postaci zbioru sprawdzonych szachowych rozwiązań dotyczących rozpoczęcia rozgrywki, z których korzystałby komputer przy wyznaczaniu początkowych ruchów.

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. Na przykład, skoczek w centrum planszy będzie miał większy potencjał od takiego, który umieszczony jest w rogu. Tak samo pion, który jest bliżej ostatniego rzędu pól, ma większą wartość 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 między kolejnymi drzewami w sekwencji, za każdym razem następuje wczytanie do pamięci podręcznej wartości i pozycji wszystkich węzłów danego drzewa. Oznacza to, że jeśli użytkownik analizuje dwa kolejne drzewa i przełącza między nimi na zmianę, za każdym razem te same drzewa są wczytywane do pamięci podręcznej na nowo. Usprawnieniem tego

procesu byłoby przechowywanie wcześniej już wczytanych drzew, 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.

Jest kilka czynników wpływających negatywnie na szybkość wykonywania obliczeń w języku *Python*. Przykładowymi czynnikami są fakt, że jest to język dynamicznie typowany, oraz to, że udostępnia relatywnie mało możliwości w zakresie zarządzania pamięcią. W celu przyspieszenia modułu odpowiedzialnego za grę w szachy, można by przepisać go w całości z języka *Python* do języka takiego jak *C++*.

8. Wnioski

Praca nad projektem powiązanym z tematyką sztucznej inteligencji pozwoliła autorom zrozumieć dokładną zasadę działania algorytmu UCT i sprawiła, że zgłębili oni wiedzę na temat jednego z popularniejszych algorytmów dotyczących symulacji obiecujących ruchów komputera w grach logicznych.

Proces implementacji wizualizacji drzewa stanów uświadomił autorom, jak bardzo wydajne może być zastosowanie karty graficznej, w celu wykonywania sprawnych obliczeń związanych z grafiką. Utwierdził ich też w przekonaniu, jak dużo możliwości daje korzystanie z biblioteki *OpenGL*.

Dodatkowo podczas pracy autorzy nauczyli się efektywnego łączenia komponentów z różnych bibliotek w celu stworzenia jednej spójnej aplikacji, a także sprawnego korzystania z zewnętrznych dokumentacji. Zaobserwowano również, że biblioteki przyciągające programistów swoją łatwością w obsłudze nie zawsze są najlepszym wyborem. Za przykład może posłużyć używana biblioteka *PyGame* do graficznej reprezentacji gier. Ostatecznie zrezygnowano z niej, ponieważ nie była ona kompatybilna z innymi komponentami, między innymi z modułem *PyQt*, odpowiedzialnym za tworzenie interfejsu graficznego.

Zmierzono się również z problemem dotyczącym rysowania drzew nieprzecinających się krawędziami, w sposób możliwie zwięzły. Okazało się, że znaleziony w opracowaniach naukowych ulepszony algorytm Walkera nie wydaje się bardzo popularny, mimo swojego czasu obliczeń rzędu $\Theta(n)$. Pozwoliło to uświadomić autorom, jak wiele wydajnych, a jednocześnie mało znanych rozwiązań można znaleźć we wszelkiego rodzaju źródłach naukowych.

Podczas pracy zrozumiano, że trendy panujące na rynku informatycznym dotyczące języków programowania nie zawsze współgrają z ich wydajnością. Przykładowo, *Python* ze względu na swoje dynamiczne typowanie nie jest najlepszym wyborem do programów wymagających wykonywania bardzo dużej ilości obliczeń i ustępuje szybkością starszym językom programowania,

takim jak na przykład *C*. Ze względu na swoją nieskomplikowaną składnię jest on natomiast dobrym wyborem do zadań, które wymagają napisania kodu w szybki sposób, lecz niekoniecznie optymalnie działającego.

Bibliografia

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

[2] Steven James, George Konidaris, Benjamin Rosman, *An Analysis of Monte Carlo Tree Search*, University of the Witwatersrand, Johannesburg, South Africa.

[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).

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

[5] Jan Karwowski, Jacek Mańdziuk, *A Monte Carlo Tree Search approach to finding efficient patrolling schemes on graphs*, European Journal of Operational Research, vol. 277, no. 1, pp. 255-268 (2019).

Spis rysunków

Spis tabel

Spis załączników

1. Płyta CD zawierająca:
    - treść całej pracy dyplomowej,
    - stronę tytułową pracy,
    - streszczenie w języku polskim,
    - streszczenie w języku angielskim,
    - kod programu,
    - dokumentację wygenerowaną z kodu.