

Wizualizacja drzewa stanów algorytmu UCT

Dokumentacja powykonawcza

Patryk Fijałkowski
Grzegorz Kacprowicz

18 grudnia 2019

Streszczenie

Poniższy dokument zawiera dokumentację powykonawczą projektu, którym było stworzenie aplikacji pozwalającej na wizualizację drzewa stanów algorytmu UCT. Ma ona w zamyśle pozwalać na oglądanie i dokładną analizę rozgrywki z komputerem podczas grania w jedną z dwóch gier planszowych. Dokument przeprowadza czytelnika przez instrukcję poprawnego uruchomienia programu oraz opis funkcjonalności połączony z instrukcją użytkowania. Będzie on zawierał również opis interfejsu użytkownika, dokładnie opisujący najistotniejsze okna aplikacji. Dokument pozwala także zaznajomić się z architekturą programu oraz opisem i schematami modułów aplikacji - zaczynając od tego odpowiedzialnego za wizualizację. Pierwszy moduł, będący najistotniejszym, będzie opierał się na usprawnionej wersji algorytmu Walkera. Opisane są również moduły odpowiedzialne za logikę zaimplementowanych gier, implementację algorytmu oraz serializowanie generowanych drzew wraz ze schematami serializacji. *Aplikacja główna*, czyli ostatni opisywany moduł, jest modułem służącym do prezentacji działania poprzednich modułów. Ostatni rozdział dokumentu opisuje i uzasadnia technologie wybrane do stworzenia aplikacji.

Historia zmian

Wersja	Data	Autor(zy)	Zmiany
1.0	14.12.2019	PF, GK	stworzenie szkicu dokumentu
1.1	17.12.2019	PF, GK	stworzenie pierwszej wersji dokumentu

Spis treści

1	Instrukcja uruchamiania	3
2	Poradnik użytkownika	4
2.1	Okno główne aplikacji	4
2.2	Okno podglądu drzewa	5
2.3	Okno gry i wizualizacji	6
3	Architektura systemu	7
3.1	Diagram klas głównych komponentów	7
3.2	Diagram stanów aplikacji	8
3.3	Diagram sekwencji rozgrywki	9
3.4	Diagram sekwencji eksportu drzewa	10
3.5	Diagram sekwencji wizualizacji	11
4	Raport z testów akceptacyjnych	12
5	Użyte technologie	14

1 Instrukcja uruchamiania

Aplikacja została napisana w języku *Python*, zatem klient powinien zainstalować jego interpreter, który może znaleźć na stronie: <https://www.python.org/downloads/>. Wymagana wersja to Python 3.7.2 lub nowsza.

Do tak ściągniętego interpretera załączony jest również *pip* - menedżer pakietów Python. Za jego pomocą należy pobrać niezbędny pakiet. Przed pobraniem pakietów zalecane jest uaktualnienie menedżera następującą komendą:

- na Windowsie:

```
1 python -m pip install -U pip
```

- na Linuxie:

```
1 pip install -U pip
```

Następnie, należy uruchomić następującą komendę w konsoli, aby pobrać pakiet PyQt5, o którym więcej będzie w ostatnim rozdziale dokumentu:

```
1 pip install PyQt5
```

Ze względu na fakt, iż Python jest językiem interpretowanym, program również należy uruchomić z poziomu konsoli. Należy zatem przejść do katalogu **/uct-visualization/src/main_application** i wpisać komendę:

```
1 python main_application_window.py
```

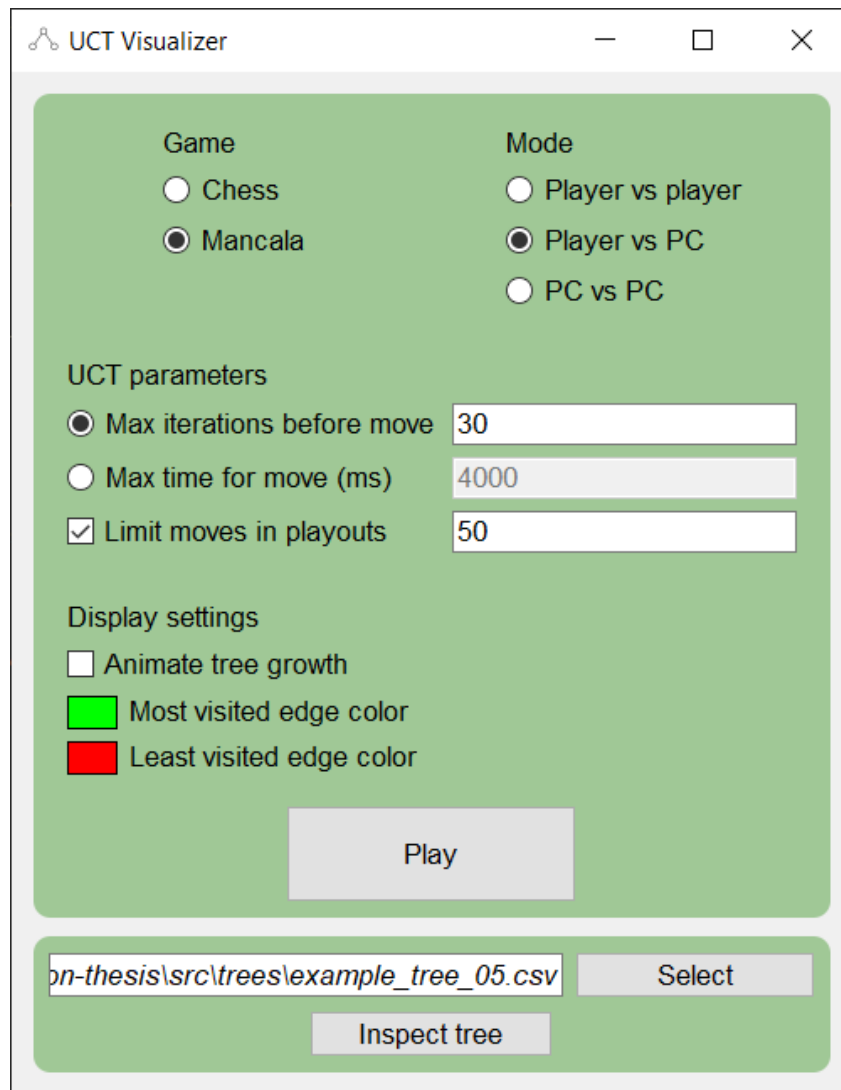
lub uruchomić program z głównego katalogu aplikacji **/uct-visualization**:

```
1 python /uct-visualization/src/main_application/main_application_window.py
```

Po uruchomieniu powinniśmy zobaczyć główne okno aplikacji o nazwie *UCT Visualizer*.

2 Poradnik użytkownika

2.1 Okno główne aplikacji

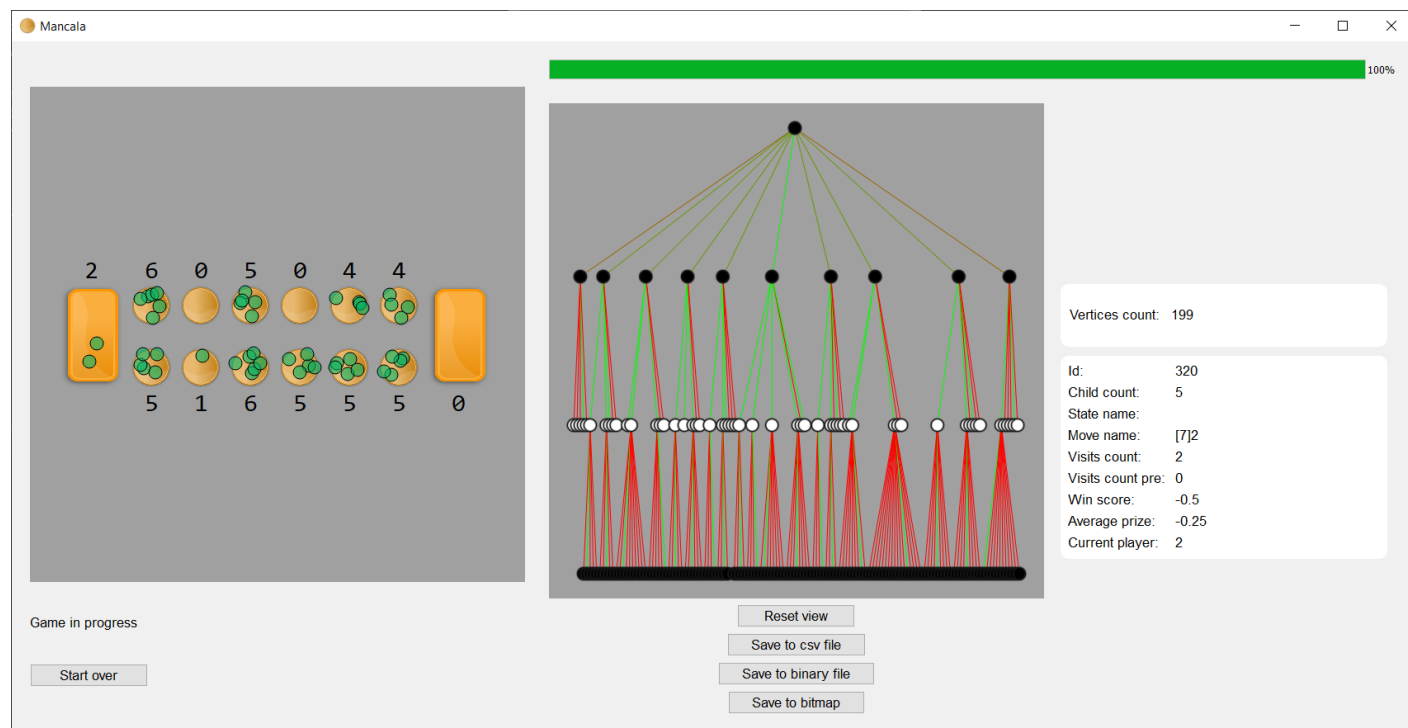


The screenshot shows the 'UCT Visualizer' application window. It has a title bar with a file icon, the text 'UCT Visualizer', and standard window controls (minimize, maximize, close). The main content area is green and contains several sections of controls:

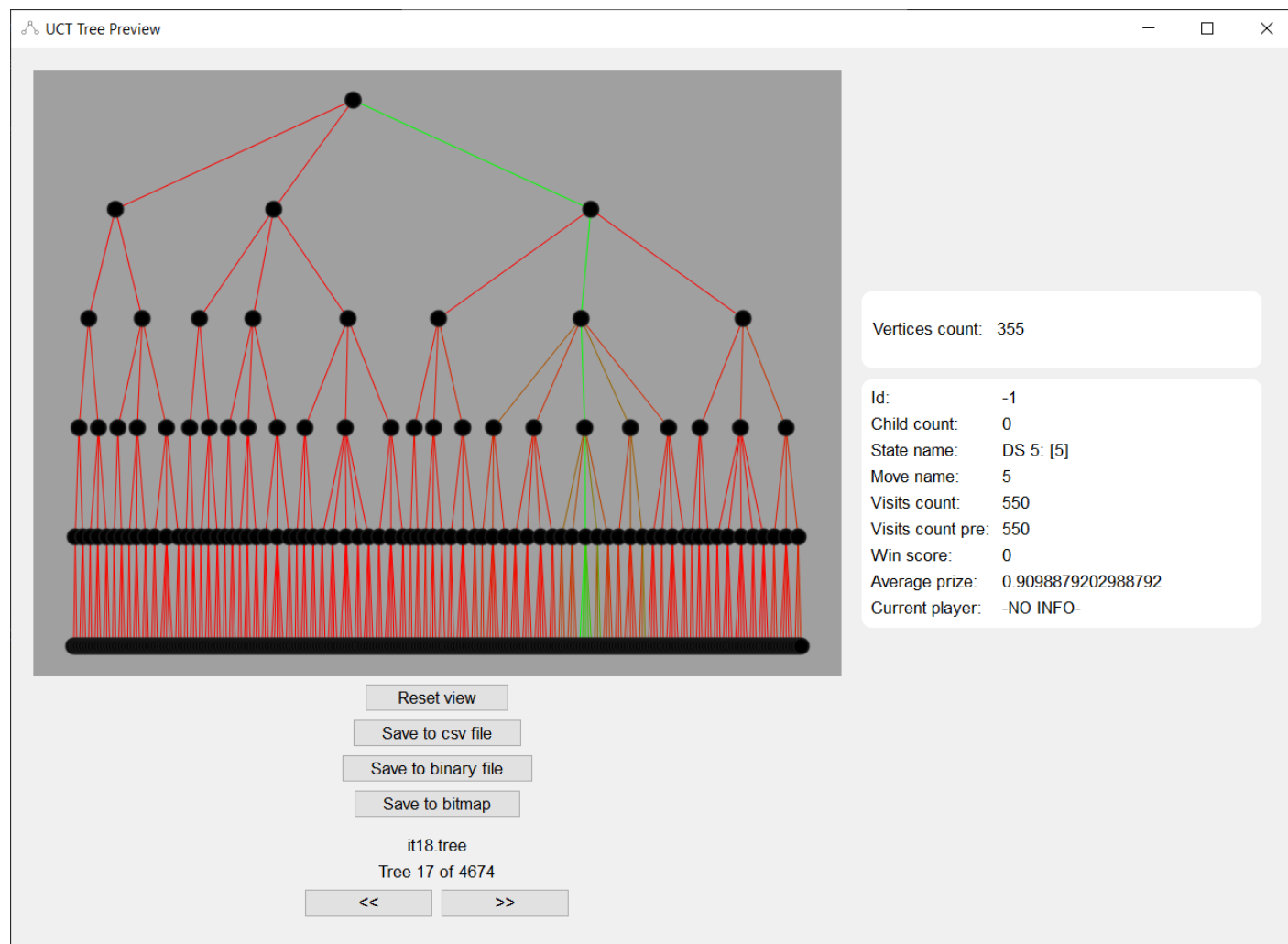
- Game**: Two radio buttons, 'Chess' and 'Mancala'. 'Mancala' is selected.
- Mode**: Three radio buttons, 'Player vs player', 'Player vs PC', and 'PC vs PC'. 'Player vs PC' is selected.
- UCT parameters**: Three rows of controls. The first row has a radio button for 'Max iterations before move' (selected) and a text input field with '30'. The second row has a radio button for 'Max time for move (ms)' and a text input field with '4000'. The third row has a checked checkbox for 'Limit moves in playouts' and a text input field with '50'.
- Display settings**: A checkbox for 'Animate tree growth' (unchecked). Below it are two color selection controls: a green square for 'Most visited edge color' and a red square for 'Least visited edge color'.

At the bottom of the green area is a large grey 'Play' button. Below the green area is a white section containing a text input field with the file path 'on-thesis\src\trees\example_tree_05.csv', a grey 'Select' button, and a grey 'Inspect tree' button.

2.2 Okno podglądu drzewa



2.3 Okno gry i wizualizacji

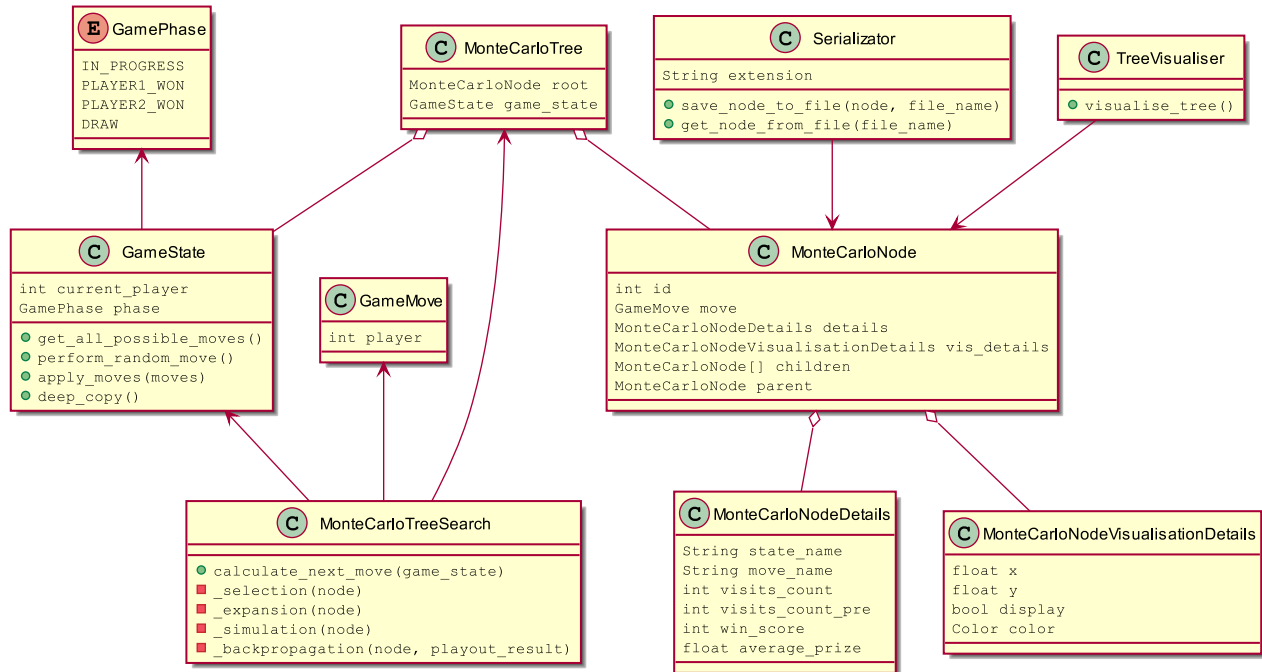


3 Architektura systemu

Aplikacja jest podzielona na pięć oddzielnych modułów: *algorytm*, *serializacja*, *wizualizacja*, *gry*, które będą funkcjonować w obrębie nadrzędnego modułu - *aplikacji głównej*.

3.1 Diagram klas głównych komponentów

Rysunek 1 ukazuje diagram klas najważniejszych komponentów związanych z modułami *Algorytm*, *Wizualizacja* i *Seria-
lizacja*.



Rys. 1: Diagram klas głównych komponentów

Zgodnie z diagramem, klasy `MonteCarloTreeSearch`, `TreeVisualiser` oraz `Serializer` są pośrednio lub bezpośrednio zależne od klasy `MonteCarloNode`, opisującej wierzchołek w drzewie. Jest to część wspólna modułów *Algorytm*, *Wizualizacja* i *Serializacja*. Klasa `MonteCarloNode` przechowuje referencję do swojego rodzica oraz wierzchołków potomnych, aby zachować rekurencyjną strukturę drzewa.

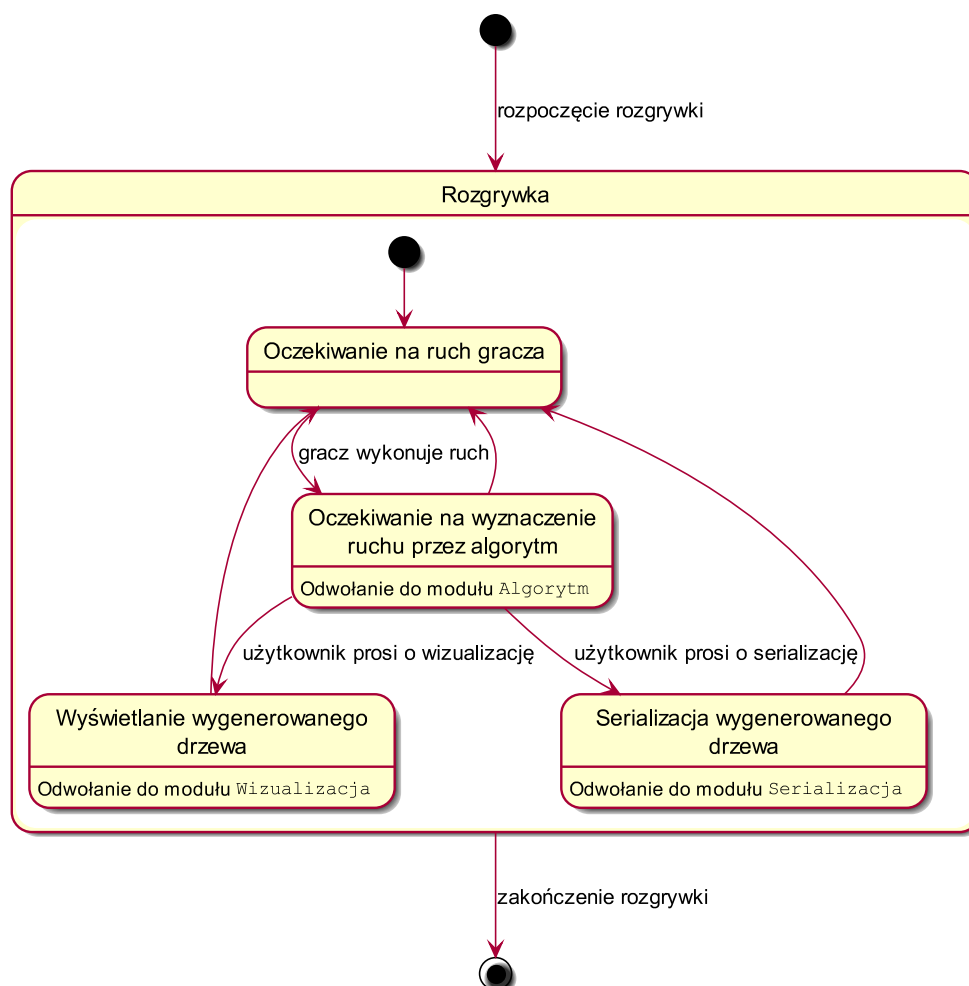
Metoda `calculate_next_move` klasy `MonteCarloTreeSearch` odpowiada za wykonanie kolejnych iteracji algorytmu. Algorytm zapisuje informacje o rozgrywanych playoutach w polach klasy `MonteCarloNodeDetails` analizowanych wierzchołków. Ruch oraz stan analizowanej gry są opisane odpowiednio przez klasy `GameMove` 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.

3.2 Diagram stanów aplikacji

Rysunek 2 ukazuje diagram stanów aplikacji w przypadku rozgrywki w trybie *człowiek kontra maszyna*.



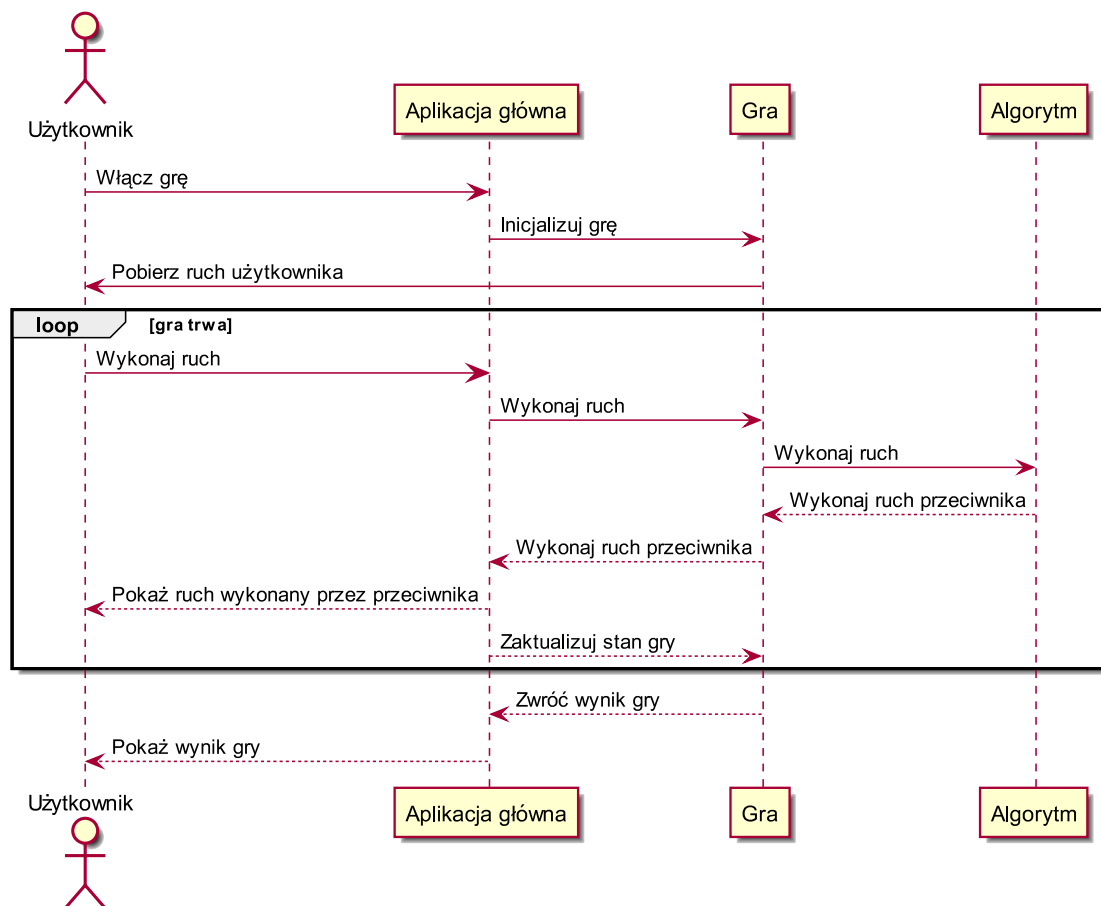
Rys. 2: Diagram stanów aplikacji

Zgodnie z diagramem, aplikacja po rozpoczęciu rozgrywki przechodzi do obszernego stanu *Rozgrywka*, zawierającego cztery wewnętrzne stany. Będąc w stanie *Rozgrywka*, aplikacja może potencjalnie korzystać z każdego modułu aplikacji.

Istotna z punktu widzenia użytkownika jest możliwość serializowania wygenerowanego drzewa lub jego wizualizacja zaraz po ruchu wyznaczonym przez algorytm, co powoduje przejście aplikacji odpowiednio w stany *Serializacja wygenerowanego drzewa* oraz *Wyświetlanie wygenerowanego drzewa*.

3.3 Diagram sekwencji rozgrywki

Rysunek 3 ukazuje diagram sekwencji rozgrywki w trybie *człowiek kontra maszyna*.



Rys. 3: Diagram sekwencji rozgrywki

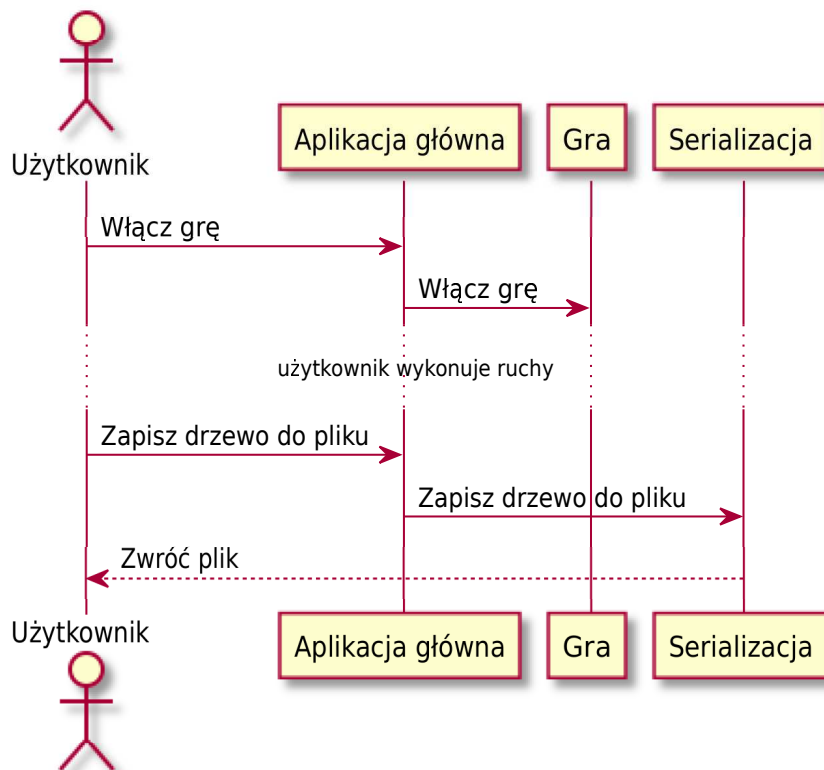
Istotne jest, jak w tej sytuacji komunikują się ze sobą moduły *Aplikacja główna*, *Gra* i *Algorytm*. Zgodnie z założeniami, *Aplikacja główna* jest interfejsem użytkownika do korzystania z pozostałych modułów.

Użytkownik końcowy za pomocą menu aplikacji głównej może ustawić parametry gry i następnie włączyć 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 i gry z modulem *Algorytm* (i vice versa). Po zakończeniu rozgrywki gra zwraca swój stan, który jest możliwy do zobaczenia przez użytkownika poprzez okno aplikacji głównej.

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

3.4 Diagram sekwencji eksportu drzewa

Rysunek 4 przedstawia proces współpracy różnych komponentów aplikacji w celu wyeksportowania wygenerowanego przez algorytm drzewa. Proces uruchamiania gry i wykonywania ruchów jest analogiczny do tego na rysunku 3.



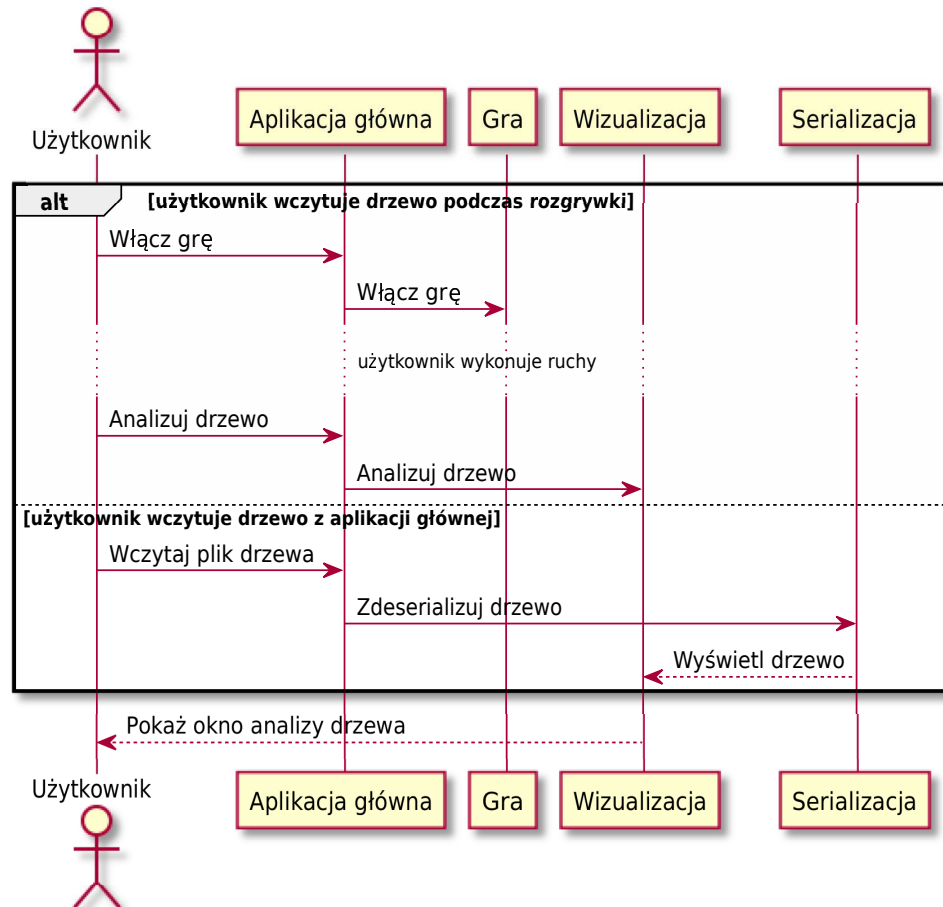
Rys. 4: Diagram sekwencji eksportu drzewa

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 modulem odpowiedzialnym za serializację, który zapisuje drzewo do pliku. Plik drzewa zapisywany jest do specjalnego folderu na tego typu pliku i posiada datę wygenerowania.

Jest to diagram dla ustawienia *człowiek kontra maszyna*, jednak w przypadku *maszyna kontra maszyna* istnieje taka sama funkcjonalność i diagram byłby analogiczny.

3.5 Diagram sekwencji wizualizacji

Rysunek 5 przedstawia proces uruchamiania wizualizacji drzewa przez użytkownika jako współpracę poszczególnych komponentów aplikacji. Ponownie, proces uruchamiania gry i wykonywania ruchów wygląda tak jak na rysunku 3.



Rys. 5: Diagram sekwencji wizualizacji

Ważne jest, że użytkownik może uruchomić wizualizację z poziomu rozgrywki tuż po wygenerowaniu nowego drzewa przez algorytm lub już na etapie menu głównego. Gdy żądanie jest z poziomu rozgrywki, komponent *Aplikacja główna* komunikuje się z komponentem *Wizualizacja*, który generuje aktualne drzewo i pokazuje je użytkownikowi w nowym oknie.

Drugi sposób (żądanie analizy drzewa z menu głównego) wymaga wcześniejszego wczytania drzewa z pliku i odpowiednio jego deserializację 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 wygenerowanym drzewem.

Interakcja wyżej wymienionych komponentów wygląda tak samo również w przypadku, gdy użytkownik poprosi o przeanalizowanie większej ilości drzew za jednym razem.

4 Raport z testów akceptacyjnych

Testy akceptacyjne zostały przeprowadzone w celu sprawdzenia, czy aplikacja spełnia założenia opisane w dokumentacji wymagań projektu. Test 1 konfrontuje założenia modułu *Gry*, test 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.

Testowane wymaganie	<i>Użytkownik będzie mógł wybrać jedną z dwóch przykładowych gier, a do wyboru będzie miał trzy tryby rozgrywki.</i>
Kroki testowe	<ol style="list-style-type: none">1. Z menu głównego aplikacji wybierz opcję <i>Chess</i>.2. Z menu głównego aplikacji wybierz opcję <i>Player vs player</i> i sprawdź tryb rozgrywki dla dwóch graczy.3. Z menu głównego aplikacji wybierz opcję <i>Player vs PC</i> dla różnych ustawień algorytmu UCT.4. Z menu głównego aplikacji wybierz opcję <i>PC vs PC</i> dla różnych ustawień algorytmu UCT.5. Z menu głównego aplikacji wybierz <i>Mancala</i> i powtórz kroki 2-5.
Wynik	Pozytywny.

Tab. 1: Raport z pierwszego testu

Testowane wymaganie	<i>Użytkownik będzie mógł zapisać analizowane drzewa do pliku csv, do pliku binarnego oraz do bitmapy.</i>
Kroki testowe	<ol style="list-style-type: none">1. Z menu głównego aplikacji wybierz ścieżkę do dowolnego pliku z zserializowanym drzewem.2. Naciśnij przycisk <i>Inspect tree</i>.3. Naciśnij przycisk <i>Save to csv file</i>.4. Naciśnij przycisk <i>Save to binary file</i>.5. Naciśnij przycisk <i>Save to bitmap file</i>.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.

Tab. 2: Raport z drugiego testu

Testowane wymaganie	<i>Użytkownik będzie mógł wyświetlić informacje związane z wybranym węzłem drzewa, a także przybliżać i oddalać cały graf.</i>
Kroki testowe	<ol style="list-style-type: none"> 1. Z menu głównego aplikacji wybierz ścieżkę do dowolnego pliku z drzewem. 2. Naciśnij przycisk <i>Inspect tree</i>. 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 przyskiem myszki i sprawdź, czy panel z prawej strony wyświetla informacje związane z wybranym wierzchołkiem.
Wynik	Pozytywny.

Tab. 3: Raport z trzeciego testu

Testowane wymaganie	<i>Dla drzew do 100 000 wierzchołków wizualizacja nie powinna zajmować więcej niż 3s.</i>
Kroki testowe	<ol style="list-style-type: none"> 1. Z menu głównego aplikacji wybierz ścieżkę do pliku <i>tree_100k.csv</i>. 2. Naciśnij przycisk <i>Inspect tree</i>.
Wynik	Pozytywny - deserializacja, ulepszony algorytm Walkera i wyświetlenie drzewa z pliku zajęło 2.802s.

Tab. 4: Raport z czwartego testu

Testowane wymaganie	<i>Dla drzew do 250 000 wierzchołków wizualizacja nie powinna zajmować więcej niż 5s.</i>
Kroki testowe	<ol style="list-style-type: none"> 1. Z menu głównego aplikacji wybierz ścieżkę do pliku <i>tree_250k.csv</i>. 2. Naciśnij przycisk <i>Inspect tree</i>.
Wynik	Pozytywny - deserializacja, ulepszony algorytm Walkera i wyświetlenie drzewa z pliku zajęło 4.626s.

Tab. 5: Raport z piątego testu

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

5 Użyte 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. *PyQt5* - nakładki na bibliotekę Qt, umożliwiającą tworzenie interfejsu graficznego. Dla projektów takich jak praca inżynierska, *PyQt* dystrybuowana jest na zasadach *GNU General Public License*.