Wizualizacja drzewa stanów algorytmu UCT

Dokumentacja powykonawcza

Patryk Fijałkowski
Grzegorz Kacprowicz

19 grudnia 2019

Streszczenie

Poniższy dokument zawiera dokumentację powykonawczą projektu, którym było stworzenie aplikacji pozwalającej na wizualizację drzewa stanów algorytmu UCT. Pozwala ona 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 systemu połączony z instrukcją użytkowania. Zawiera on również opis interfejsu użytkownika, dokładnie opisujący najistotniejsze okna aplikacji. Dokument pozwala zaznajomić się z architekturą programu oraz opisem i schematami modułów aplikacji – zaczynając od tego odpowiedzialnego za wizualizację. Pierwszy moduł opiera 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. Następnie można zobaczyć raport z przeprowadzonych testów akceptacyjnych. W ostatnim rozdziale dokumentu wymienione są użyte technologie wraz z ich wersjami i licencjami.

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 użytkowania i opis funkcjonalności **4**
2.1 Okno startowe aplikacji 4
2.2 Okno podglądu drzewa 6
2.3 Okno gry i wizualizacji 8

3 Architektura systemu **9**
3.1 Diagram klas głównych komponentów 9
3.2 Diagram stanów aplikacji 10
3.3 Diagram sekwencji rozgrywki 11
3.4 Diagram sekwencji eksportu drzewa 12
3.5 Diagram sekwencji wizualizacji 13

4 Raport z testów akceptacyjnych **14**

5 Użyte technologie **16**

6 Użyte grafiki **17**

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:

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

- na Linuxie:

```
pip install -U pip
```

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

```
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:

```
python main_application_window.py
```

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

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

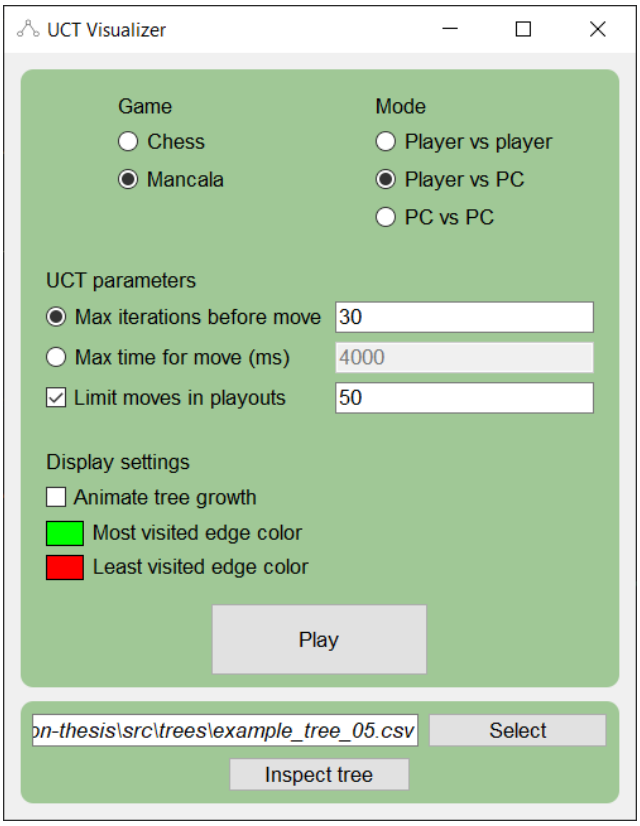
Po uruchomieniu powinniśmy zobaczyć główne okno aplikacji o nazwie *UCT Visualizer*.

2 Poradnik użytkowania i opis funkcjonalności

Dla wygodnej interakcji użytkownika z programem przygotowany został specjalny interfejs graficzny. Możemy wyróżnić trzy główne okna.

2.1 Okno startowe aplikacji

Jest to główne okno programu, w którym użytkownik może wybrać, czy chce zagrać w grę, czy wczytać i obejrzeć przygotowane wcześniej pliki drzew. Wygląd okna jest przedstawiony na rysunku 1.



Rys. 1: Okno startowe aplikacji

Dostępne funkcje:

1. Wybór jednej z dwóch gier:
   - mancala
   - szachy.

2. Wybór jednego z trzech trybów gry:
   - *gracz vs PC* – po wykonanym ruchu gracza algorytm oblicza ruch komputera i wykonuje go
   - *PC vs PC* – użytkownik decyduje, kiedy mają wykonać się ruchy komputera
   - *gracz vs gracz* – rozgrywka dwóch graczy, co wiąże się z brakiem wizualizacji.

3. Ustawienie parametrów ruchu algorytmu UCT:
   - ilość wykonanych iteracji – im więcej ich będzie, tym więcej razy powtórzone będą kroki symulacji kolejnego ruchu przez komputer, co przekłada się na jego większą dokładność i trafność. Minusem jest większy czas oczekiwania na wykonaniu ruchu. (przedział: 1–10000)
   - maksymalny czas na wykonanie ruchu przez komputer – opcja alternatywna dla powyższej. Zamiast podawać odgórnie ilość iteracji, użytkownik może podać czas, w jakim komputer będzie wyznaczał ruch. Po upłynięciu czasu ostatnia symulacja jest dogrywana. (przedział: 1000ms–30000ms).

4. Ucięcie iteracji po przekroczeniu arbitralnie ustalonej liczby ruchów danego "playoutu" (symulacji rozgrywki w obrębie iteracji). Opcja może być włączona lub wyłączona. Dla tego drugiego przypadku gra kończy się tylko wtedy, gdy symulacja gry zakończy się w naturalny sposób.

5. Włączenie animacji rozrostu drzewa – pozwala obserwować jak zmieniało się drzewo w trakcie symulacji ruchu – drzewo wyświetlane jest po każdej iteracji algorytmu. W przypadku wybrania opcji maksymalnego czasu na wykonanie ruchu przez komputer ta opcja uwzględni również czas rysowania wszystkich drzew, co wiąże się ze stratą wydajności algorytmu.

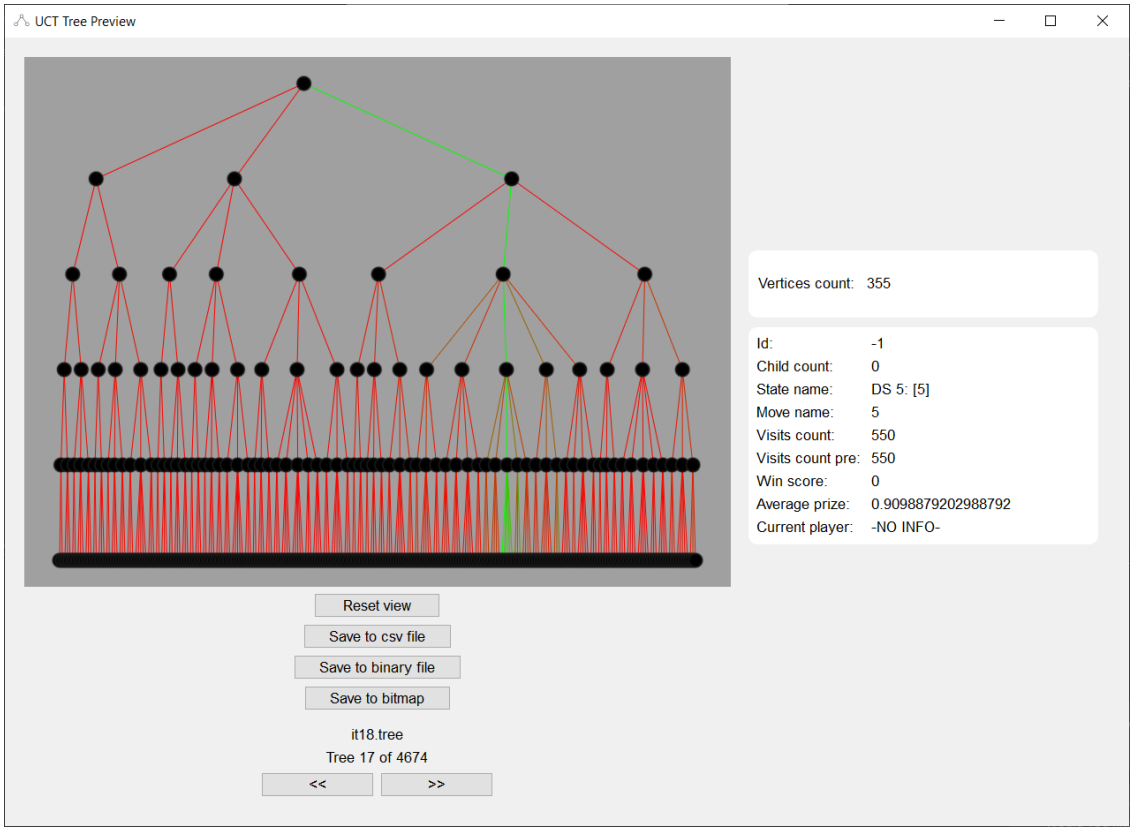
6. Wybór kolorów krawędzi – dla ruchów najczęściej i najrzadziej odwiedzanych przez algorytm. Kolory pozostałych krawędzi będą gradientami podanych kolorów – dla powyższego przykładowego okna, krawędzie częściej odwiedzanych węzłów będą bardziej zielone i mniej czerwone.

7. Wybór drzew do przeglądu – po kliknięciu przycisku *Select* otwiera się dodatkowe okno, które prosi użytkownika o wybranie pliku bądź zestawu plików drzew do analizy. Przyjmowane będą pliki w formacie .csv i binarnym (.tree), w ktorych dane zapisane będą według ustalonego schematu. Zakładamy poprawność wczytywanych plików. W polu tekstowym wyświetli się wówczas ścieżka do wczytanego pliku lub liczba plików w przypadku wczytania ich większej liczby.

Aby rozpocząć rozgrywkę, należy nacisnąć przycisk *Play*. Aby wizualizować wczytane drzewa – *Inspect tree*.

2.2 Okno podglądu drzewa

Okno (rys. 2) umożliwia wizualizację drzew z wczytanych plików oraz sprawne poruszanie się po niej, wraz z możliwością wyświetlania statystyk poszczególnych węzłów.



Rys. 2: Okno podglądu drzewa

Dostępne funkcje:

1. Przybliżanie i oddalanie drzewa za pomocą scrolla. Kursor musi znajdować się nad częścią okna, gdzie wyświetlane jest drzewo.

2. Przesuwanie drzewa – należy przytrzymać prawy przycisk myszy i poruszać myszką.

3. Wyświetlenie statystyk węzła – należy kliknąć lewym przyciskiem myszy na dany węzeł. Po wybraniu węzła, po prawej stronie wyświetlają się jego statystyki:
   - liczba wszystkich wierzchołków drzewa
   - identyfikator węzłą
   - liczba węzłów–dzieci
   - nazwa stanu
   - nazwa ruchu
   - liczba odwiedzin węzła przez algorytm
   - suma wszystkich wygranych (łącznie wyniki pozytywne i negatywne)
   - średnia wygrana – suma wygranych ÷ ilość odwiedzin. Wartość z przedziału $[0, 1]$
   - numer gracza, który wykonał ruch dla danego węzła (1 lub 2).

4. Wyśrodkowanie drzewa – resetuje stopień przybliżenie i przesunięcia.

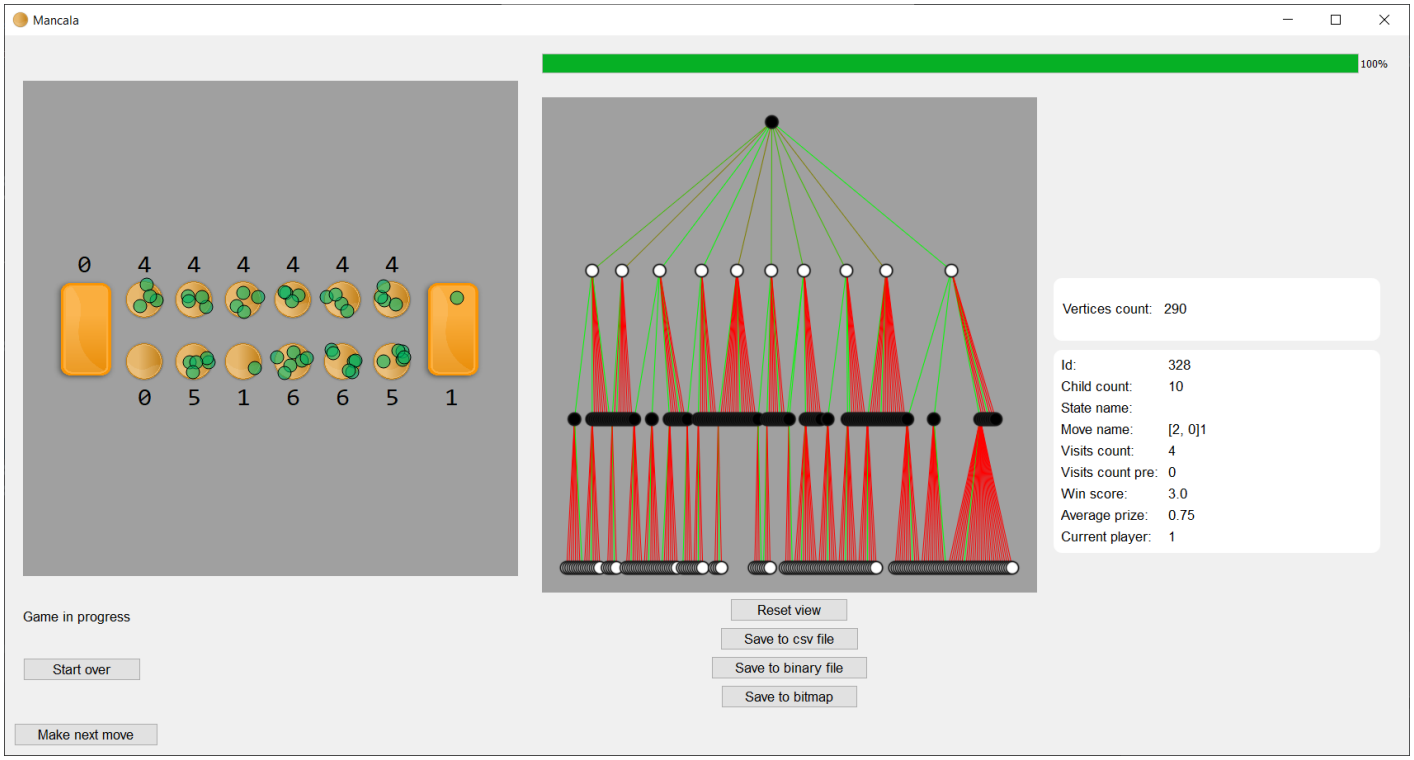
5. Zapis drzewa:
    - w formacie .csv
    - w formacie binarnym (.tree)
    - jako bitmapa (.png).

6. Zmiana drzewa w sekwencji – możemy robić to za pomocą przycisków « i » na ekranie lub za pomocą strzałek na klawiaturze. Wyświetlana jest nazwa aktualnie przeglądanego pliku oraz to, którym jest on plikiem z kolei.

Zamknięcie tego okna nie kończy programu – okno menu głównego jest ciągle aktywne i może być wykorzystane ponownie.

2.3 Okno gry i wizualizacji

Jest to połączenie gry oraz okna podglądu drzewa opisanego powyżej. Znajduje się tu interfejs graficzny toczącej się gry, w którą może ingerować użytkownik i wykonywać swoje ruchy. W górnej części znajduje się pasek postępu, który odzwierciedla ilość wykonanych przez algorytm iteracji podczas wyznaczania następnego ruchu. Po jego wyznaczeniu w środkowej częsci okna ukazuje się wygenerowane drzewo. Jeżeli użytkownik włączył opcję animacji drzew w menu głównym, dodatkowo po każdej iteracji będzie mógł on oglądać rozrost drzewa.



Rys. 3: Okno gry i wizualizacji

Dostępne funkcje:

1. Wszystkie funkcje dostępne w oknie podglądu drzewa poza sekwencjami.

2. Wykonanie ruchu z pomocą GUI gry, np. kliknięcie danego pola w szachach lub dołka w mancali. Nie dotyczy trybu *gracz vs PC*. Po zakończonej rozgrywce napis *Game in progress* pod polem gry zmienia wartość i oznajmia użytkownikowi, która strona wygrała (lub czy jest remis).

3. Rozegranie gry od nowa – kliknięcie przycisku *Start over*.

4. Wykonanie następnego ruchu za pomocą przycisku – tylko w trybie *PC vs PC*. Pozwala na spowodowanie postępu w symulacji rozgrywki komputera z samym sobą. Ruchy za pomocą GUI są wówczas zablokowane.

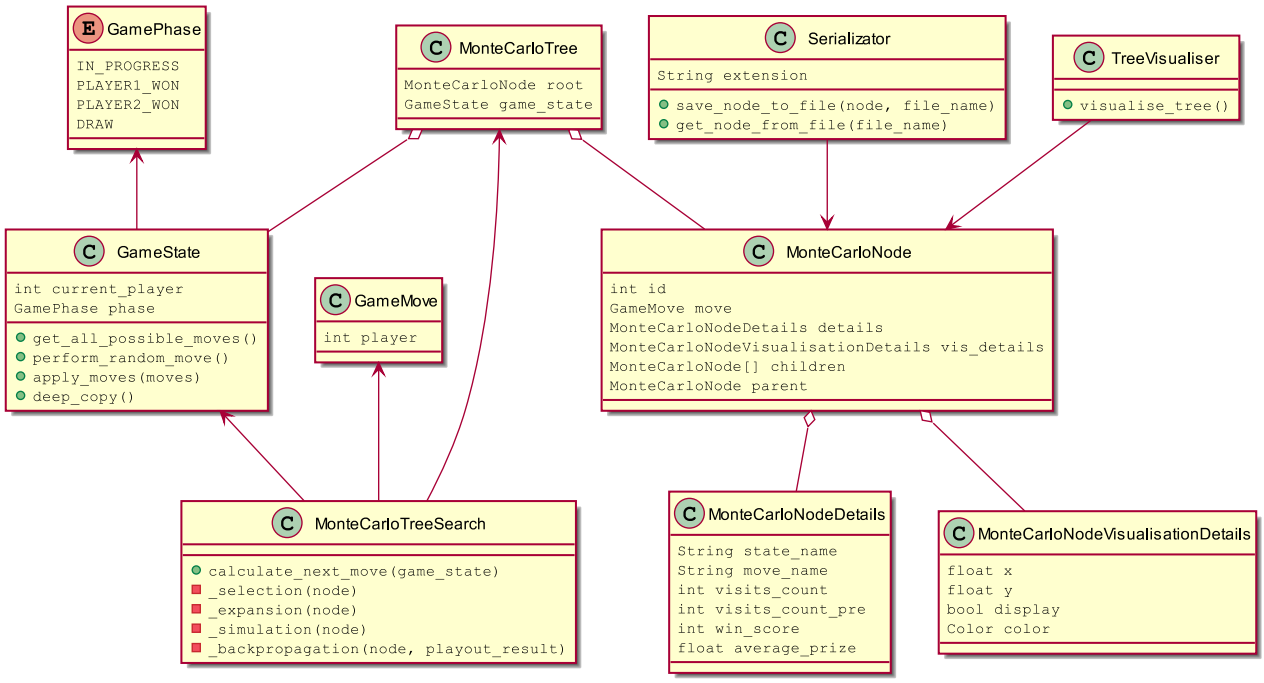
Zamknięcie tego okna również nie kończy programu – okno menu głównego jest ciągle aktywne i może być wykorzystane ponownie.

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 4 ukazuje diagram klas najważniejszych komponentów związanych z modułami *Algorytm*, *Wizualizacja* i *Serializacja*.



Rys. 4: Diagram klas głównych komponentów

Zgodnie z diagramem, klasy `MonteCarloTreeSearch`, `TreeVisualiser` oraz `Serializator` są pośrednio lub bezpośrednie 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.

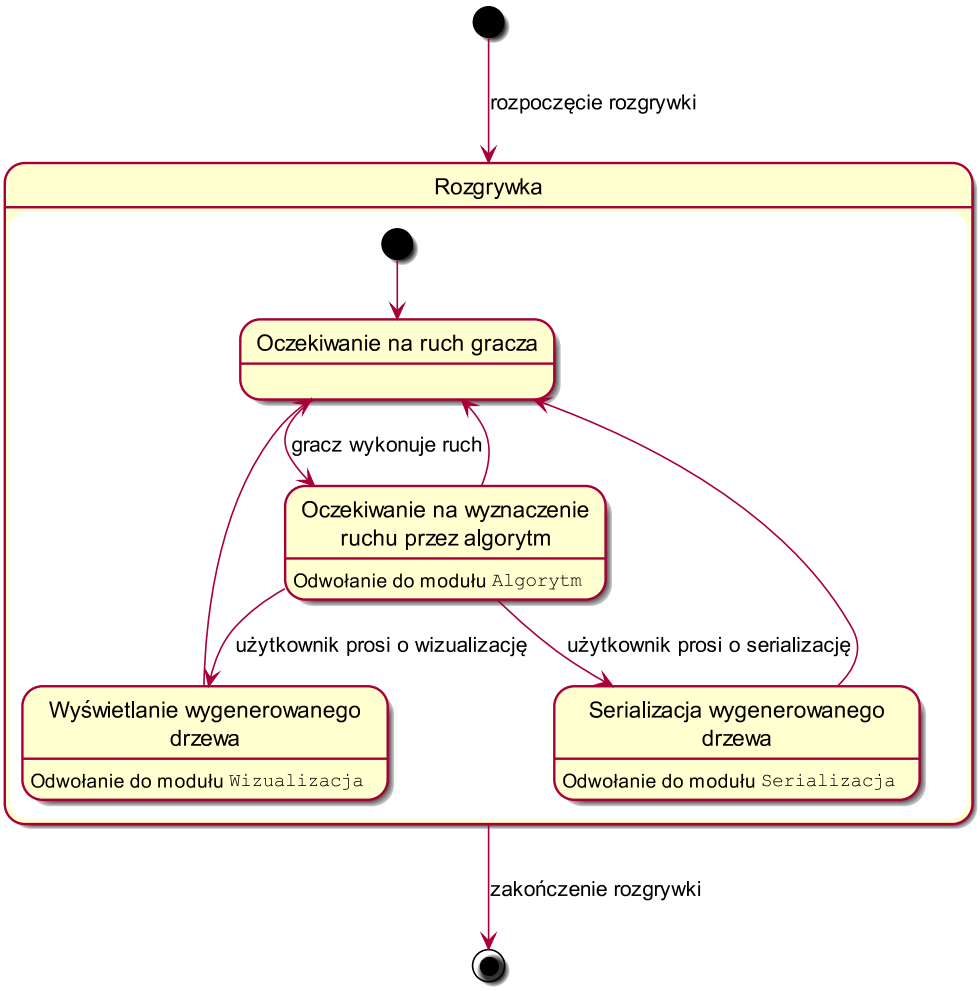
Metoda `calculate_next_move` klasy `MonteCarloTreeSearch` odpowiada za wykonanie kolejnych iteracji algorytmu. Algorytm zapisuje informacje o rozgrywanych playoutach w polach klasy `MonteCarloNodeDetails` analizowanych wierzchołków. Ruch oraz stan analizowanej gry są opisane odpowiednio przez klasy `GameMove` 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 5 ukazuje diagram stanów aplikacji w przypadku rozgrywki w trybie *człowiek kontra maszyna*.



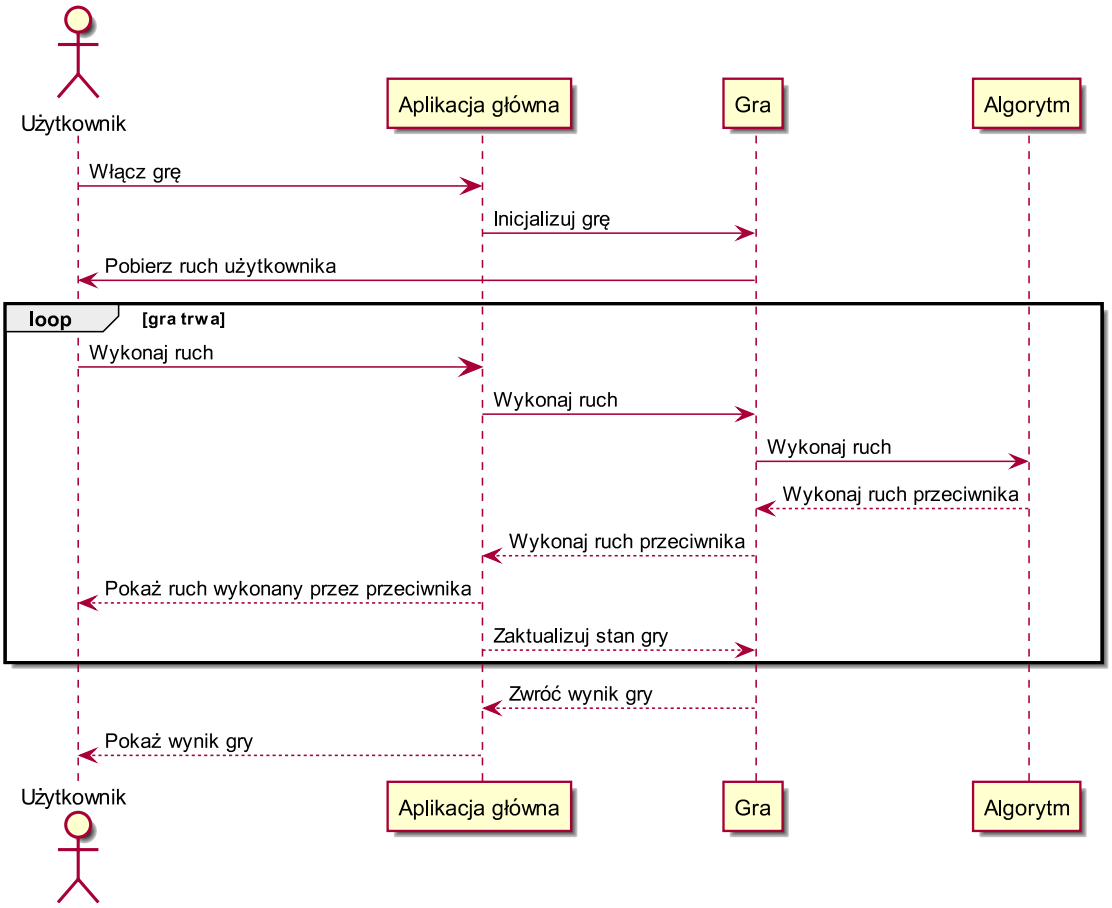
Rys. 5: Diagram stanów aplikacji

Zgodnie z diagramem, aplikacja po rozpoczęciu rozgrywki przechodzi do obszernego stanu *Rozgrywka*, zawierającego cztery wewnętrzne stany. Będac 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 6 ukazuje diagram sekwencji rozgrywki w trybie *człowiek kontra maszyna*.



Rys. 6: Diagram sekwencji rozgrywki

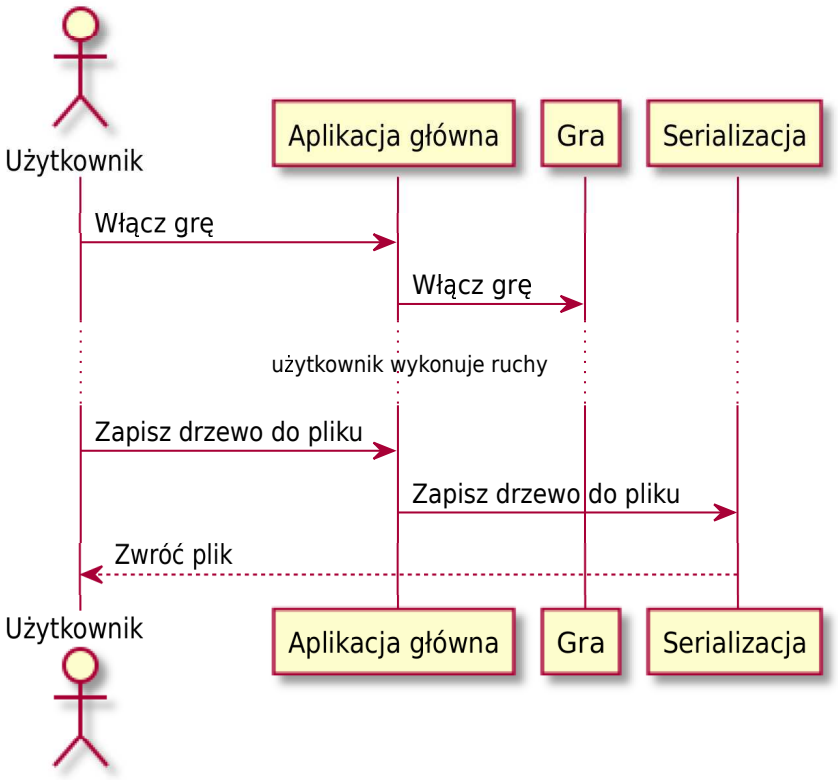
Istotne jest, jak w tej sytuacji komunikują się ze sobą moduły *Aplikacja główna*, *Gra* i *Algorytm*. Zgodnie z założeniami, *Aplikacja główna* jest interfejsem użytkownika do korzystania z pozostałych modułów.

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 modułem *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 7 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 6.



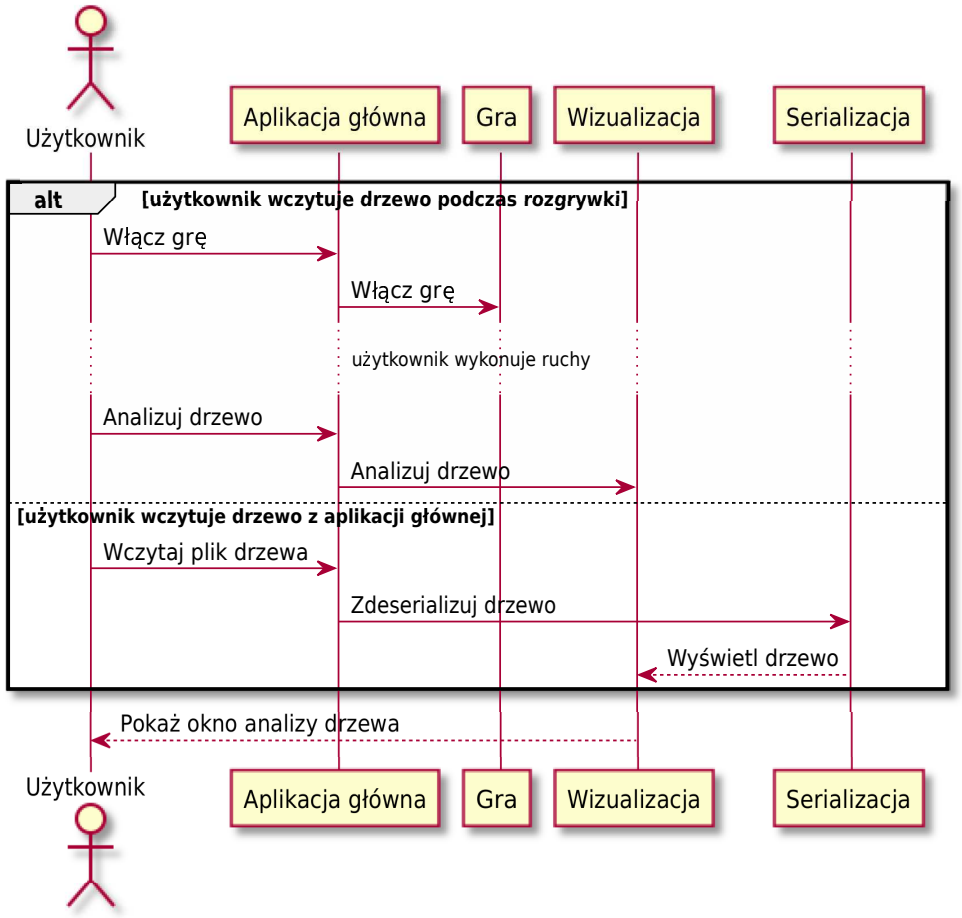
Rys. 7: Diagram sekwencji eksportu drzewa

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 modułem odpowiedzialnym za serializację, który zapisuje drzewo do pliku. Plik drzewa zapisywany jest do specjalnego folderu na tego typu pliki 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 8 przedstawia proces uruchamiania wizualizacji drzewa przez użytkownika jako współpracę poszczególnych komponentów aplikacji. Ponownie, proces uruchamiania gry i wykonywania ruchów wygląda tak jak na rysunku 6.



Rys. 8: 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	*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.

Tab. 1: Raport z pierwszego 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.

Tab. 2: Raport z drugiego 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.

Tab. 3: Raport z trzeciego 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.

Tab. 4: Raport z czwartego 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.

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*.

6 Użyte grafiki

1. Bierki w szachach – `https://icons8.com/`
2. Dołki w mancali – `https://www.wpclipart.com/`