

Wizualizacja drzewa stanów algorytmu UCT

Plan projektu

Patryk Fijałkowski
Grzegorz Kacprowicz

4 listopada 2019

Streszczenie

Poniższy dokument zawiera ogólny zarys projektu. Aplikacja ma w zamyśle pozwalać na oglądanie i dokładną analizę rozgrywki z komputerem w jedną z dwóch gier planszowych. Jest ona potencjalnie rozszerzalna o kolejne gry wpasowujące się w założenia algorytmu UCT. W dokumencie przedstawione są wszystkie moduły aplikacji wraz z pełnionymi funkcjami oraz ich dokładny opis i diagramy UML. Z dokumentu można dowiedzieć się też o zachowaniu poszczególnych komponentów względem siebie oraz jakie opcje są dostępne dla użytkownika.

Historia zmian

Wersja	Data	Autor(zy)	Zmiany
1.0	3.11.2019	PF, GK	stworzenie pierwszej wersji dokumentu

Spis treści

1	Architektura aplikacji	3
1.1	Wizualizacja	3
1.2	Algorytm	3
1.3	Gry	3
1.4	Serializacja	4
1.5	Aplikacja główna	4
2	Główne komponenty aplikacji	5
2.1	Diagram klas	5
2.2	Diagram stanów	6
2.3	Diagram sekwencji rozgrywki	7
2.4	Diagram sekwencji serializacji	8
2.5	Diagram sekwencji wizualizacji	9
3	Interfejs użytkownika	10
3.1	Menu główne	10
3.2	Analiza drzewa	11
3.3	Rozgrywka	12
4	Wybrane technologie	13

1 Architektura aplikacji

Aplikacja będzie podzielona na pięć oddzielnych modułów: algorytm, serializacja, wizualizacja, gra pierwsza i gra druga, które będą funkcjonować w obrębie aplikacji głównej. Cele każdego z modułów i zadania powierzone im są przedstawione poniżej.

1.1 Wizualizacja

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

Dla czytelnych wizualizacji, poczyniliśmy następujące założenia:

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

Aby wyznaczyć układ wierzchołków na płaszczyźnie, spełniając powyższe 6 założeń, skorzystamy z usprawnionego algorytmu Walkera, który działa w czasie liniowym względem liczby wierzchołków. Algorytm, który zaimplementujemy, został opisany w pracy *Improving Walker's Algorithm to Run in Linear Time*¹.

1.2 Algorytm

Moduł “Algorytm” jest implementacją algorytmu Monte Carlo Tree Search, korzystającą z usprawnienia 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.

Aby gra była poprawnie obsługiwana przez moduł “algorytm”, musi spełniać następujące założenia:

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

Rozdział trzeci zawiera dokładniejszy opis funkcjonalności, które należy zapewnić, by moduł “Algorytm” mógł wyznaczać kolejne ruchy danej gry.

1.3 Gry

Aplikacja będzie udostępniała 2 gry planszowe, umożliwiające przetestowanie efektywności wizualizacji oraz algorytmu. Obie gry będą umożliwiały 3 tryby rozgrywki, opisane poniżej.

- **Człowiek kontra człowiek:** decyzje obojga graczy są podejmowane przez użytkownika aplikacji.
- **Człowiek kontra maszyna:** decyzje jednego z graczy są podejmowane przez użytkownika, natomiast drugi gracz podejmuje decyzje najoptymalniejsze z punktu widzenia algorytmu UCT.
- **Maszyna kontra maszyna:** decyzje obojga graczy są podejmowane przez algorytm.

¹“Improving Walker's Algorithm to Run in Linear Time” - Christop Buchheim, Michael Jünger, Sebastian Leipert, Universität zu Köln, Institut für Informatik

1.4 Serializacja

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

Serializacja binarna

W serializacji binarnej przyjmujemy opisany niżej schemat.

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

Serializacja do plików csv

W serializacji do plików csv przyjmujemy, że każdy kolejny wiersz odpowiada kolejnemu wierzchołkowi drzewa, a kolejne wartości opisujące wierzchołek oddzielamy przecinkami. Ostatnią wartością jest liczba wierzchołków potomnych. Jeśli wierzchołek v ma k potomków, to pod wierszem opisującym wierzchołek v będzie k wierszy opisujących jego potomków. Każdy wierzchołek serializujemy do wiersza postaci:

R, O, O2, W, S, D

Oznaczenia:

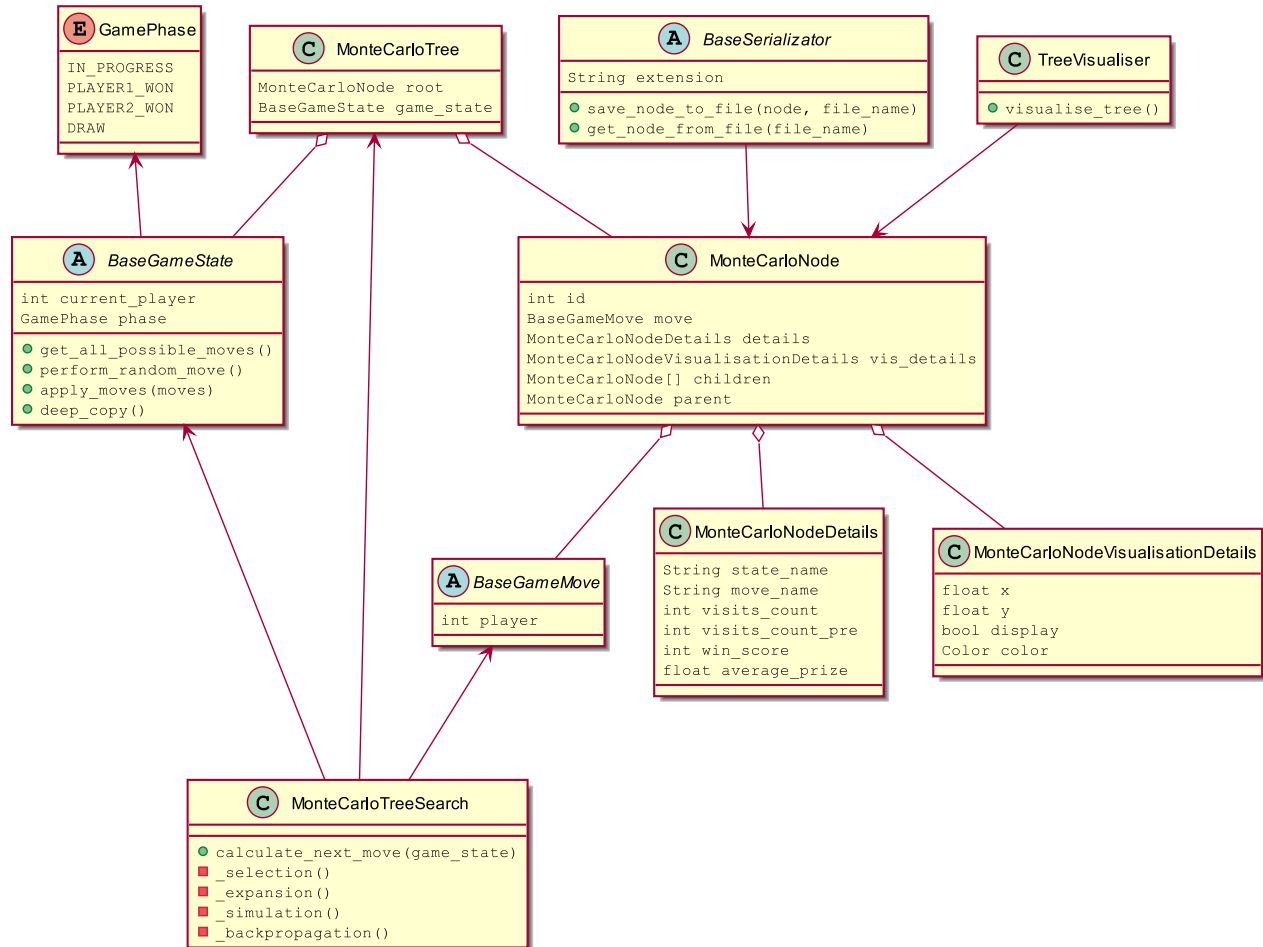
- R - nazwa ruchu
- O - licznik odwiedzin
- O2 - dodatkowy licznik odwiedzin
- W - średnia wypłata algorytmu za ruch
- S - nazwa stanu
- D - liczba wierzchołków potomnych

1.5 Aplikacja główna

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

2 Główne komponenty aplikacji

2.1 Diagram klas



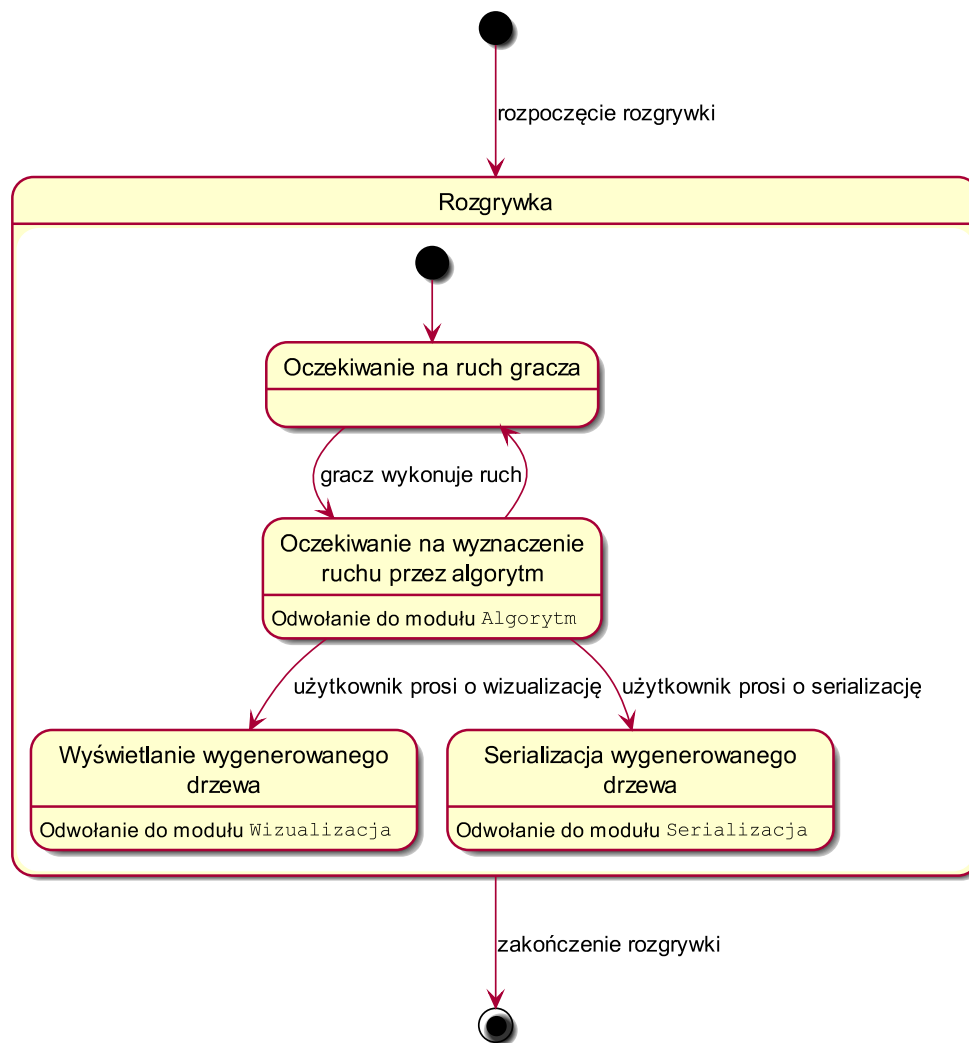
Rys. 1: Diagram klas głównych modułów

Rysunek 1 ukazuje diagram klas najważniejszych komponentów związanych z modułami “Algorytm”, “Wizualizacja” i “Serializacja”. Opis wierzchołka drzewa jest częścią wspólną dla każdego z tych modułów. Zgodnie z diagramem, klasy `MonteCarloTreeSearch`, `TreeVisualiser` oraz `BaseSerializer` są pośrednio lub bezpośrednio zależne od klasy `MonteCarloNode`.

Metoda `calculate_next_move` klasy `MonteCarloTreeSearch` odpowiada za wykonanie kolejnych iteracji algorytmu. Ruch oraz stan analizowanej gry są opisane odpowiednio przez klasy `BaseGameMove` i `BaseGameState`. Daje to użytkownikowi możliwość testowania również innych gier poprzez zaimplementowanie tych dwóch klas i ich metod. 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 zawarte są w `MonteCarloVisualisationDetails`.

2.2 Diagram stanów

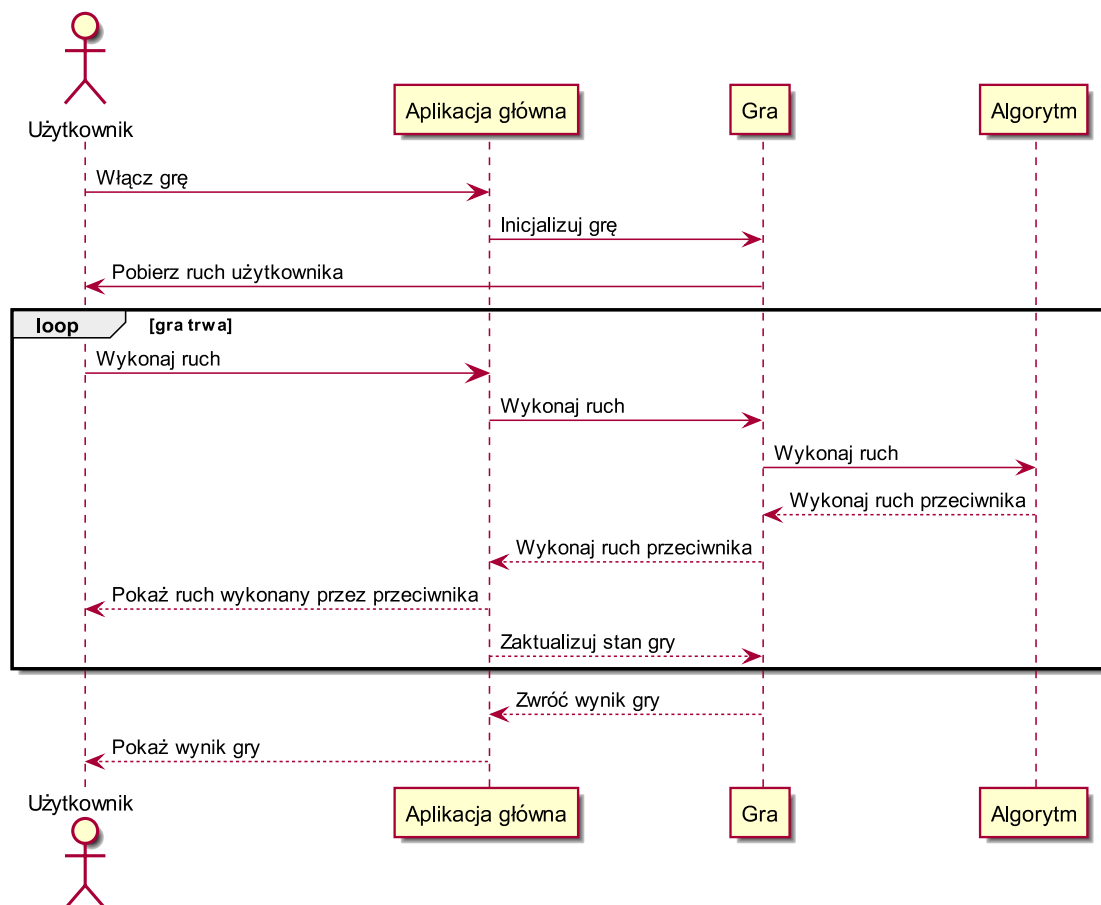


Rys. 2: Diagram stanów aplikacji

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

2.3 Diagram sekwencji rozgrywki

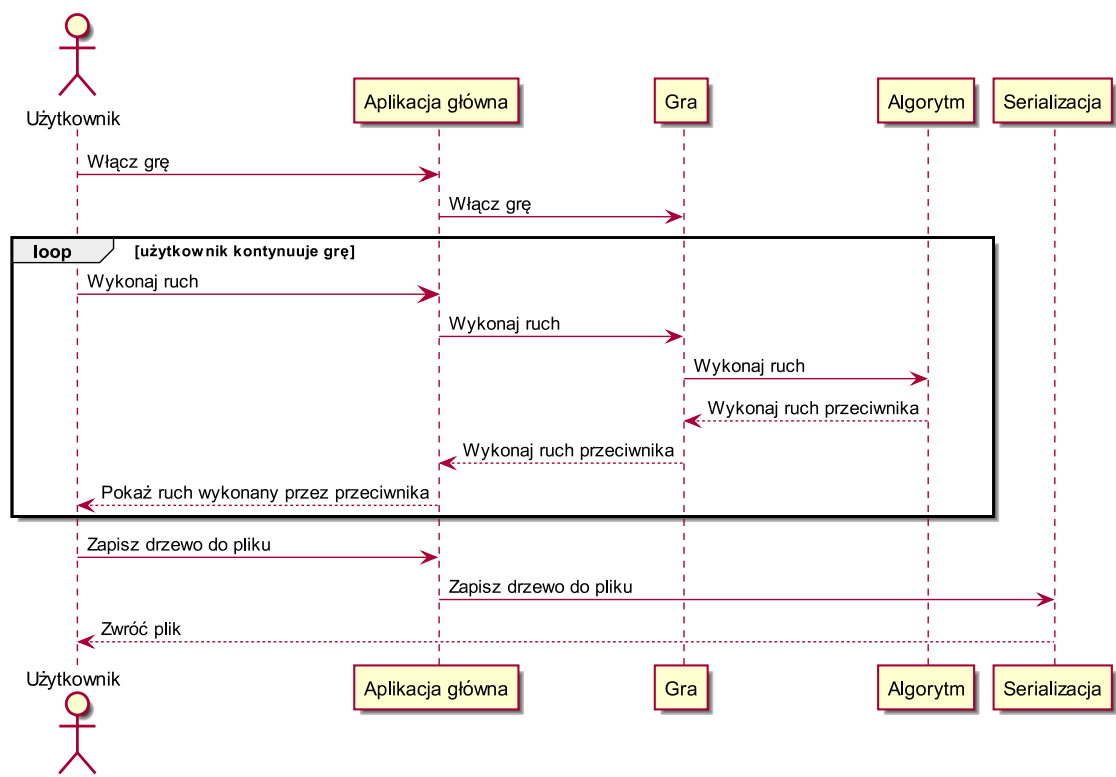


Rys. 3: Diagram sekwencji rozgrywki

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

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

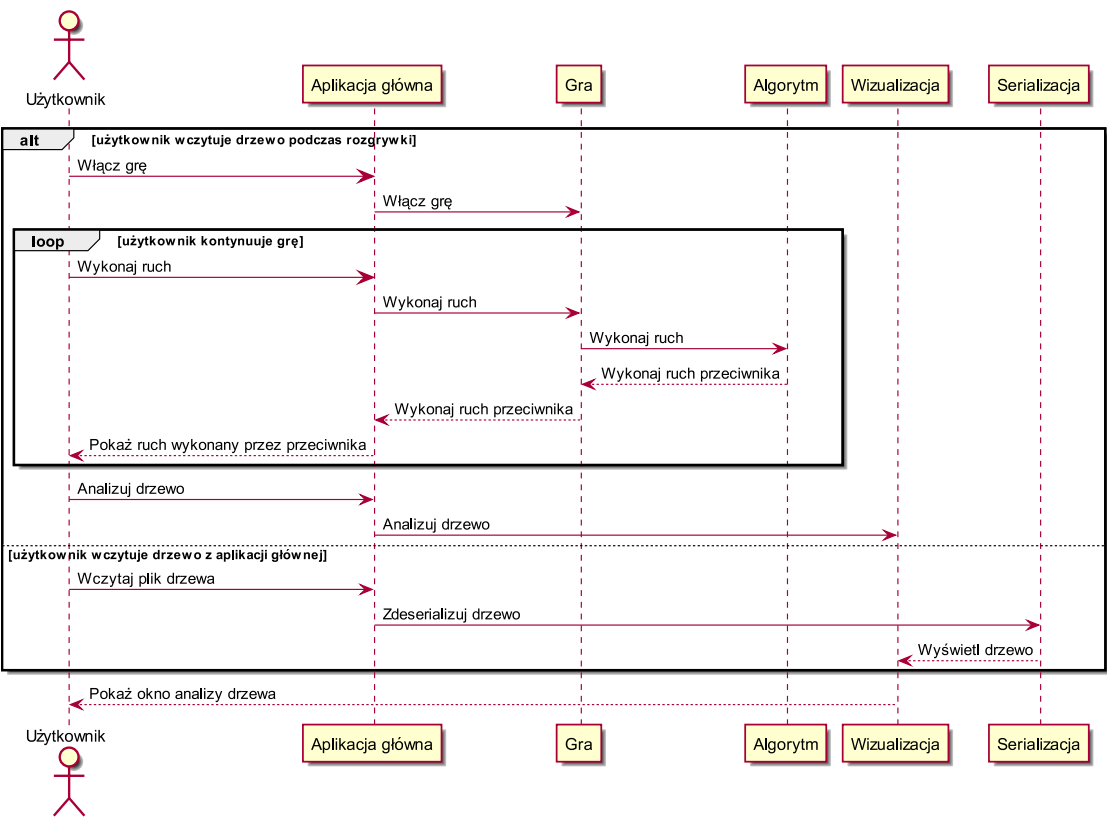
2.4 Diagram sekwencji serializacji



Rys. 4: Diagram sekwencji serializacji

BBB fh ab bsdhjf hb hds hjafd b fdhbsj hsd jfj adfjsabjdsfh hfdsdjf hbasdjfhsbd fjdbsa jhbsjaf bds hfb jfb hadfsdj b hjbs

2.5 Diagram sekwencji wizualizacji



Rys. 5: Diagram sekwencji wizualizacji

BBB fh ab bsdhjf hb hds hjafd b fdhbsj hsd jfj adfsabjdsfh hfsdjf hbasdjfhsbd fjdbsa jhbsjaf bds hfb jfb hadfsdj b hjbs

3 Interfejs użytkownika

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

3.1 Menu główne

Menu

Game:

game 1
game 2

Number of iterations before move:

Max time for move (seconds):

Mode:

☒ Player vs PC
☐ PC vs PC

Play

Select file Load

Analyze tree

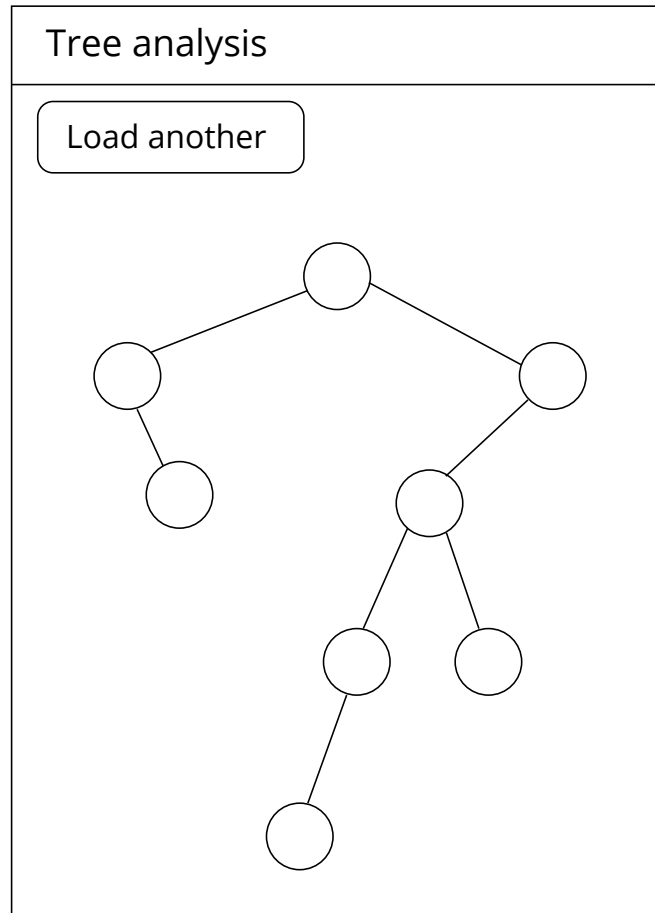
Rys. 6: Okno menu głównego

Ukazane na rysunku 6 menu główne będzie głównym oknem aplikacji i będzie to pierwsza rzecz, którą zobaczy użytkownik po uruchomieniu programu. Dwa moduły, do których można przejść z tego okna, to rozgrywka i analiza drzewa. Żeby rozegrać grę, należy nacisnąć na przycisk *Play*. Powyżej tego przycisku znajdować się będzie szereg opcji, który pozwoli użytkownikowi ustawić parametry gry dostosowane do jego preferencji, w tym między innymi:

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

Analiza drzewa będzie dostępna po naciśnięciu przycisku *Analyze tree* i uprzedniego wczytania pliku z zserializowanym drzewem (.tree, .csv).

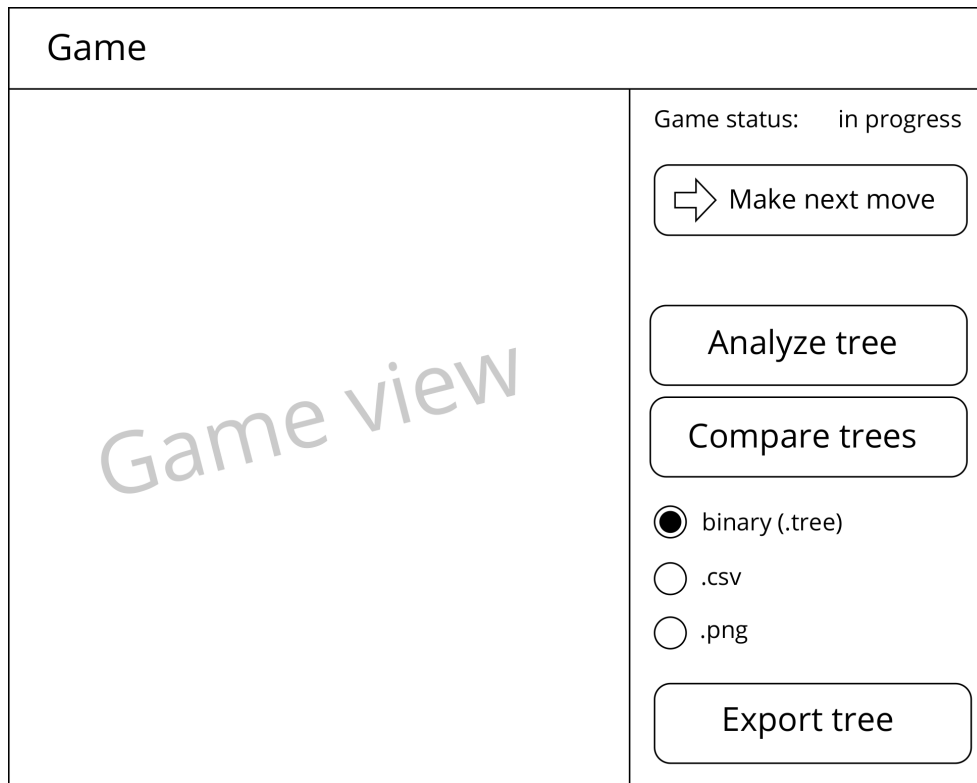
3.2 Analiza drzewa



Rys. 7: Okno analizy drzewa

W oknie ukazanym na rysunku 7 będziemy mogli oglądać wczytane lub wygenerowane drzewo. Kluczową funkcją będzie tutaj możliwość dynamicznego przybliżania i oddalania go wraz z możliwością klikania poszczególnych węzłów w celu pozyskania stanu rozgrywki w danym momencie. Widoczna będzie także informacja o tym, ile razy algorytm odwiedził dany węzeł, ile razy doprowadził on do wygranej oraz średnią nagrodę za ruch w danym węźle.

3.3 Rozgrywka



Rys. 8: Okno rozgrywki

Widok rozgrywki będzie podzielony na dwie części. Gra zawierać się będzie w wyżej pokazanym oknie po lewej stronie. To tutaj użytkownik za pomocą przygotowanego do gier GUI będzie mógł wykonać ruch. Po prawej zaś będą różne opcje związane z aktualnym stanem rozgrywki, między innymi:

- informacja o tym, czy gra się toczy, lub czy się zakończyła
- wykonaj kolejny ruch - wyłącznie w trybie rozgrywki PC vs PC. Użytkownik będzie miał możliwość kontrolowania wykonywanych przez komputer ruchów, aby samemu móc powodować postęp w rozgrywce.
- przeanalizuj powstałe drzewo - będzie to przycisk otwierający drugie okno z opisaną już poprzednio analizą drzewa
- porównaj powstałe drzewo z ostatnim - to samo co wyżej, jednak z wyraźnym zaznaczeniem zmian, które zaistniały w nowym drzewie względem starego
- wyeksportuj drzewo do pliku (csv, png lub binarnego)

4 Wybrane technologie

Wybraną przez nas technologią do napisania aplikacji tj.: gier, algorytmu i wizualizacji jest język programowania **Python** w stabilnej wersji 3.7. Do implementacji gier będziemy posługiwać się biblioteką **PyGame** (w wersji stabilnej). Wizualizacja będzie wykorzystywać w znacznym stopniu bibliotekę **VisPy** (OpenGL), w której najbardziej przydatną dla nas funkcją będzie możliwość pisania kodu w języku **C++** i stosunkowo łatwa integracja z głównym językiem projektu - Pythonem. Wykorzystana przez nas wersja tej biblioteki również będzie wersją stabilną.

Python został przez nas wybrany ze względu na swoją wszechstronność. Posiada on bardzo szeroki zakres bibliotek, co pozwoli nam napisać zdecydowaną większość kodu w jednym języku i przyspieszyć wymianę informacji między komponentami (np. kodem gier a kodem algorytmu MCTS).

VisPy jest nową technologią, która jest wciąż rozwijana, jednak została przez nas wybrana głównie ze względu na:

- współpracę z GPU, co będzie niezbędne podczas wizualizacji setek tysięcy wierzchołków grafu,
- obszerną dokumentację.

Wybór na PyGame padł ze względu na:

- łatwość pisania kodu i przemyślane API,
- popularność i dobrą dokumentację.