Wizualizacja drzewa stanów algorytmu UCT

Plan projektu

Patryk Fijałkowski
Grzegorz Kacprowicz

4 listopada 2019

Streszczenie

Poniższy dokument zawiera ogólny zarys projektu-aplikacji. Składa się on z opisu architektury systemu i poszczególnych komponentów wraz z pełnioną funkcją. Opisane są tu interfejsy składających się na ten projekt modułów, użyte biblioteki i komunikacja między nimi. Znajdują się tu też diagramy UML, które opisują klasy w projekcie i ich zachowanie względem siebie. Dokument zawiera również przykładowy interfejs graficzny dla użytkownika. Co więcej, opisane są użyte technologie i wymagania sprzętowe.

Historia zmian

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

Spis treści

1 Moduły aplikacji

1.1 Wizualizacja

Moduł "wizualizacja" udostępnia funkcjonalność wizualizacji dostarczonych drzew. Użytkownik będzie miał również możliwość przybliżania, oddalania oraz poruszania się po wizualizacji. Opisana 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 najwęziej.

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

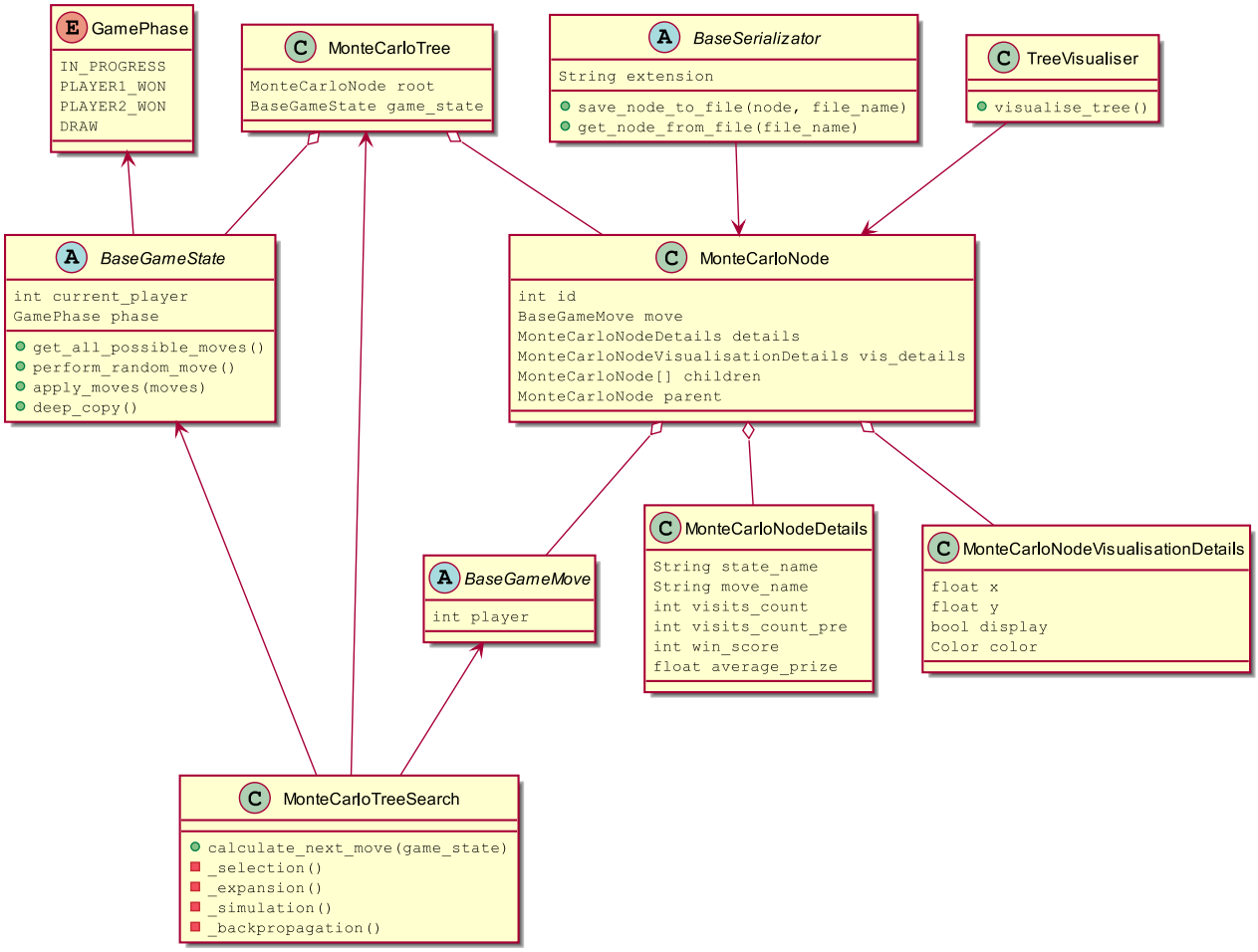
Oznaczenia:

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

1.5 Aplikacja główna

"Aplikacja główna" jest modułem 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

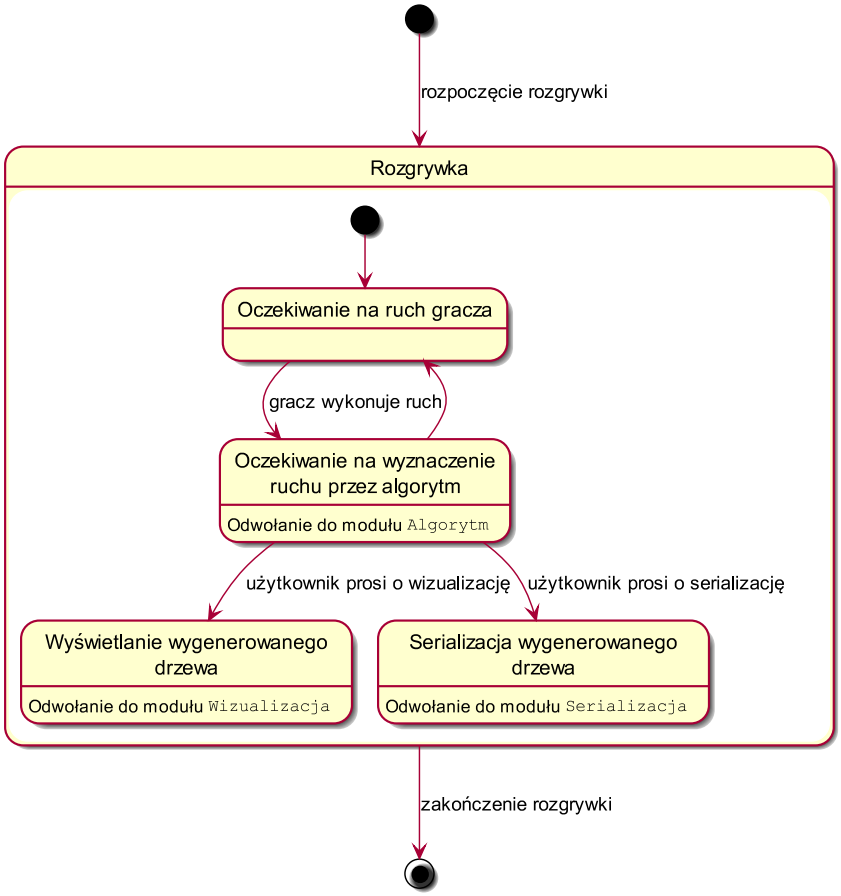


Rys. 1: Diagram klas głównych modułów

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 `BaseSerializator` są pośrednio lub bezpośrednie 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`.

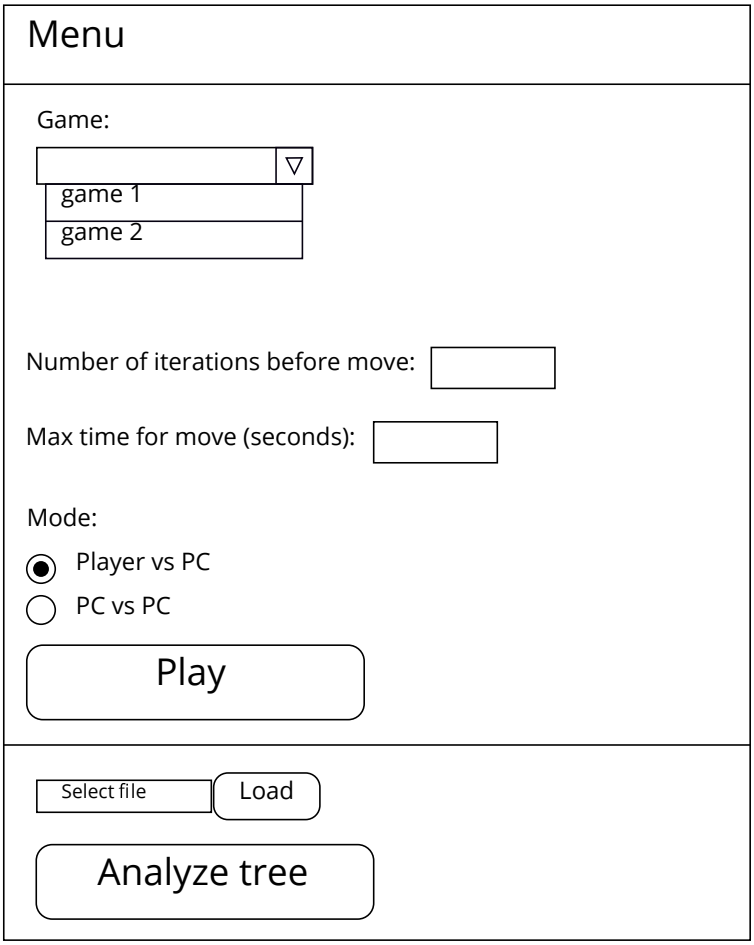


Rys. 2: Diagram stanów aplikacji

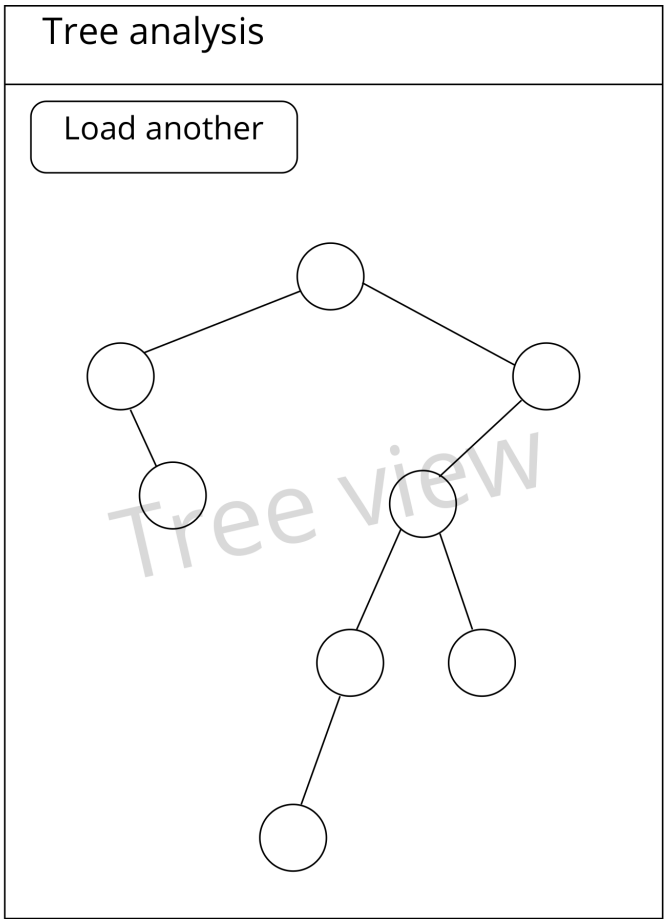
Rysunek 2 ukazuje diagram stanów aplikacji w przypadku rozgrywki "człowiek kontra maszyna". TODO

3 Interfejs użytkownika

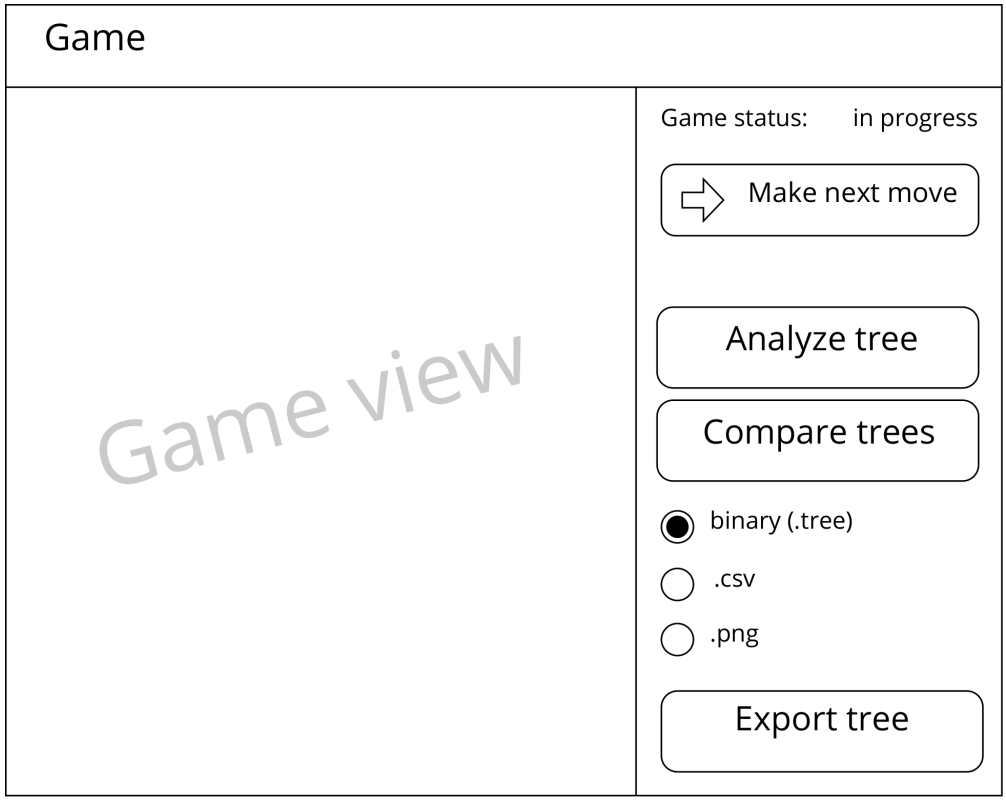
Menu:



Analiza drzewa:



Rozgrywka:


Game
Game view
Game status: in progress
Make next move
Analyze tree
Compare trees
binary (.tree)
.csv
.png
Export tree

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,
- brak lepszej alternatywy.

Wybór na PyGame padł ze względu na:
- łatwość pisania kodu i przemyślane API,
- popularność i dobrą dokumentację.

Program ma w założeniu działać na komputerze osobistym, który posiada kartę graficzną.
Aplikacja działa na systemach operacyjnych Windows i Linux.