Optymalizacja algorytmu *wave function collapse* pod kątem czasu wykonania

Filip Kowalski

Akademia Górniczo-Hutnicza, Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej, Kraków

Streszczenie: Algorytm *wave function collapse* (WFC) ma zalety, dzięki którym stanowi narzędzie przydatne w generowaniu proceduralnym, jednak często jest wymagający obliczeniowo, co ogranicza możliwości jego praktycznego stosowania. W niniejszej pracy przeanalizowano czas wykonania algorytmu WFC i zaproponowano metody jego optymalizacji. Z przeprowadzonych testów wynika, że wąskim gardłem jest krok obserwacji, w szczególności obliczanie minimalnej entropii. Aby skrócić czas wykonania algorytmu, wykorzystano dwie techniki optymalizacyjne: zapamiętywanie wartości entropii poszczególnych komórek oraz buforowanie listy indeksów komórek o aktualnie najniższej entropii. Testy przeprowadzone na siatkach o różnych rozmiarach wykazały, że zaproponowane usprawnienia pozwoliły na ponad 100-krotne przyspieszenie algorytmu, co znacząco poprawia jego użyteczność w zastosowaniach wymagających większej wydajności.

Słowa kluczowe: *wave function collapse*, WFC, optymalizacja, generowanie proceduralne

OPTIMIZATION OF THE WAVE FUNCTION COLLAPSE ALGORITHM FOR EXECUTION TIME

Abstract: The wave function collapse (WFC) algorithm, despite its advantages in procedural generation, is often computationally expensive, limiting its practical applications. The following paper analyzes the execution time of WFC and proposes methods for its optimization. It identifies the observation step, specifically the calculation of minimum entropy, as a key bottleneck. Two successive optimization techniques are introduced: memoization of individual cell entropy values and caching of the list of cell indices with the currently lowest entropy. Tests conducted on grids of varying sizes demonstrate that the proposed improvements achieve speedups exceeding 100×, significantly enhancing the algorithm's utility in performance-demanding applications.

Keywords: wave function collapse, WFC, optimization, procedural generation

https://doi.org/10.7494/978-83-68219-80-7_3

1. Wprowadzenie

Proceduralne generowanie treści (ang. *procedural content generation* – PCG) to technika tworzenia danych za pomocą algorytmów celem zautomatyzowania tego procesu tak, aby wymagał on minimalnego udziału człowieka (Lazaridis i Fragulis 2024). Od lat stanowi ona istotne narzędzie w wielu obszarach informatyki i jest powszechnie wykorzystywana do generowania terenów i miast oraz systemów zadań w grach komputerowych, jak również tekstur, obrazów czy wszelakich animacji (Smith 2021).

Klasyczne przykłady implementacji PCG opierają się na wykorzystaniu technik redukcji szumu, L-systemów lub licznych algorytmów przeszukiwania (Togelius i in. 2011, Maleki i Zhao 2024). Spośród tego rodzaju technik w ostatnich latach na popularności zyskał algorytm *wave function collapse* (WFC) zaprezentowany przez Maxima Gumina (2020). Wyróżnia się on na tle innych metod tym, że daje gwarancję lokalnej spójności generowanych struktur, a także prostotą implementacji i możliwością generowania zasobów na podstawie niewielkiej próbki wejściowej, która niekoniecznie musi być narzucona z góry przez programistę.

Mimo tych zalet algorytm WFC nie zawsze nadaje się do zastosowania w bardziej złożonych zadaniach, ponieważ jest on wymagający obliczeniowo. Czasochłonność algorytmu staje się wyzwaniem szczególnie w przypadku generowania obiektów składających się z dużej liczby pól lub potencjalnych stanów. W niniejszym artykule przedstawiono analizę działania algorytmu WFC oraz propozycje jego optymalizacji pod kątem czasu wykonania, co ma kluczowe znaczenie dla możliwości jego praktycznego wykorzystania przez producentów gier, dla których bardzo istotnym aspektem jest ich płynne działanie, bez konieczności stosowania ekranów ładowania.

1.1. Zasada działania algorytmu WFC

Fundamentem algorytmu WFC jest reprezentująca generowany system siatka wypełniona komórkami, z których każda ma przypisany jeden lub więcej stanów, jakie może przyjąć. Wpływają one na entropię komórki stanowiącą miarę używaną do wyznaczania komórek, które zostaną przetworzone w danym kroku algorytmu. Każdy ze stanów ma zestaw złącz (odpowiadający liczbie kierunków i zwrotów w danej przestrzeni – dla przestrzeni dwuwymiarowej są to cztery złącza) definiujący reguły sąsiedztwa danego stanu z każdym innym. Algorytm rozpoczyna się od inicjalizacji systemu w stanie maksymalnej entropii – każda z komórek może osiągnąć każdy z możliwych stanów. Wraz z wykonywaniem kolejnych kroków algorytmu entropia systemu maleje, aż stanie się on stabilny.

Każdy krok algorytmu polega na wyborze komórki z najniższą entropią i poddaniu jej dwóm procesom: obserwacji i propagacji. Szczegóły tych procedur zostały opisane w kolejnej sekcji[1].

1.2. Stan wiedzy na temat optymalizacji WFC

Problematyka czasu wykonania algorytmu WFC jest przedmiotem badań i tematem licznych publikacji. Istniejące podejścia do optymalizacji można podzielić na usprawnienia algorytmiczne kroku propagacji oraz modyfikacje heurystyk obserwacji. Pierwszy rodzaj podejścia reprezentują Fehr i Courant (2018), którzy wykazali, że adaptacja algorytmu AC-4 (z licznikami wsparcia) pozwala znacząco zredukować złożoność obliczeniową względem naiwnych metod sprawdzania spójności.

Podejmowane są także próby skalowania algorytmu przez jego dekompozycję na warstwy abstrakcji, co w metodzie hierarchicznej zaproponowali Beukman i współpracownicy (2023). Inni autorzy zauważają jednak, że proste metody przyspieszania obliczeń przez zrównoleglenie mają istotne ograniczenia. Jak wskazują López i Chover (2025), silne zależności skutkujące tym, że zmiana w jednej komórce może kaskadowo wpłynąć na całą mapę, czynią dekompozycję na niezależne wątki szczególnie skomplikowanym zadaniem. Wobec trudności z efektywnym zrównolegleniem kluczowa pozostaje optymalizacja sekwencyjna kroku obserwacji. W tym obszarze standardem jest wykorzystanie kopców, jednak w niniejszym artykule zaproponowano mniej wymagającą obliczeniowo technikę buforowania indeksów, dostosowaną do charakterystyki WFC.

2. Cel i metodyka badań

Celem prezentowanego artykułu jest: 1. analiza czasu wykonania algorytmu dla różnych zestawów parametrów wejściowych; 2. zidentyfikowanie kroków algorytmu, które są wymagające obliczeniowo; 3. zaproponowanie sposobów optymalizacji poszczególnych operacji.

2.1. Założenia algorytmu

Algorytm WFC wyróżnia się uniwersalnością. Może on działać w dowolnej liczbie wymiarów, wspierać symetryczne i niesymetryczne złącza poszczególnych stanów, a także pozwala na analizę próbki wejściowej zamiast wykorzystania określonego zestawu

[1] Więcej na temat działania algorytmu zob. Karth i Smith 2017.

zasad. Przygotowując implementację algorytmu na potrzeby badania, przyjęto następujące założenia:

- Algorytm działa w środowisku dwuwymiarowym.
- Algorytm wspiera tylko symetryczne połączenia poszczególnych stanów.
- Szansa wystąpienia każdego ze stanów może być inna.
- Reguły sąsiedztwa są niezmienne i określone przed uruchomieniem algorytmu.

Przedstawione ograniczenia implementacji algorytmu pozwoliły uprościć proces analizy i optymalizacji, jednocześnie zachowując kluczowe aspekty i kroki algorytmu.

2.2. Metodyka badań

W celu uzyskania obiektywnych wyników pomiar czasu wykonania algorytmu przeprowadzono w następujący sposób:

- Czas wykonania każdego z kluczowych etapów pojedynczego kroku został zmierzony za pomocą funkcji bibliotecznych udostępnionych w ramach języka C#.
- Po zakończeniu działania algorytmu WFC dla danego systemu zmierzono czas wykonania algorytmu 1, który został uznany za bazową jednostkę czasu dla tego pomiaru.
- Czas wykonania każdego z kluczowych etapów pojedynczego kroku został podzielony przez jednostkę bazową czasu.
- Przed analizą dane zostały poddane przetworzeniu celem usunięcia błędów pomiarowych. Pomiar był uznawany za poprawny, jeśli mieścił się w przedziale:

$$Q_1 - 5 \cdot \mathrm{IQR} \le x \le Q_3 + 5 \cdot \mathrm{IQR} \tag{1}$$

gdzie x to wartość pomiaru, Q_1 oraz Q_2 to odpowiednio dolny i górny kwartyl zestawu danych, a IQR to rozstęp ćwiartkowy wyrażony wzorem:

$$\mathrm{IQR} = Q_3 - Q_1 \tag{2}$$

Opisany proces pomiarowy gwarantuje otrzymanie wyników, które są niezależne od parametrów jednostki obliczeniowej oraz stanu urządzenia w momencie uruchomienia algorytmu. Nawet jeśli maszyna zostanie chwilowo obciążona, kalkulacje bazowej jednostki czasu zostaną spowolnione w tym samym stopniu co wygenerowanie treści.

```
int[] baselineArr = new int[10000];
int baselineCount = baselineArr.Length;

for (int i = 0; i < baselineCount; i++) {
  baselineArr[i] = i;
}
```

Algorytm 1. Obliczenie bazowej jednostki czasu

2.3. Przeprowadzanie testów

Każdy test szybkości działania algorytmu został przeprowadzony z wykorzystaniem takiego samego zestawu stanów oraz zasad sąsiedztwa. Zestaw składa się z 32 różnych możliwych do osiągnięcia stanów oraz 8 unikalnych identyfikatorów złącz. Generowanie treści na potrzeby badań wykonano dla siatek o następujących rozmiarach:

- 50 × 50 komórek (w sumie 2500),
- 100 × 100 komórek (w sumie 10 000),
- 200 × 200 komórek (w sumie 40 000).

3. Implementacja algorytmu w formie bazowej

Algorytm WFC został zaimplementowany z wykorzystaniem języka programowania C#, w środowisku Unity. Wybór tego środowiska pozwolił na wykorzystanie gotowych metod wizualizacji rezultatów działania algorytmu bez dodatkowego wydłużania czasu jego działania. Na potrzeby badań stworzony został modularny i prosty w obsłudze system implementujący określone założenia algorytmu WFC.

3.1. Kluczowe kroki algorytmu

Algorytm WFC w istocie sprowadza się do wykonywania naprzemiennie dwóch procedur, aż układ osiągnie stan stabilny. Warunek ten jest spełniony wtedy i tylko wtedy, gdy wszystkim komórkom w układzie przypisany został konkretny stan. Główna pętla algorytmu została przedstawiona za pomocą algorytmu 2.

```
while(true) {
 Cell cell = Collapse();

 if(cell != null) {
  Propagate(cell);
 } else {
  break;
 }
}
```

Algorytm 2. Główna pętla przetwarzania algorytmu WFC

3.1.1. Obserwacja (*collapse*)

Krok obserwacji rozpoczyna się od wytypowania komórki układu o najmniejszej entropii. W przypadku istnienia kilku komórek spełniających ten warunek algorytm wybiera ją z tego zbioru losowo. Następnie przypisywany jest jej losowy dostępny stan

(na podstawie rozkładu prawdopodobieństwa ich wystąpienia). Jeśli cały system osiągnął już w równowagę, krok ten zwraca wartość pustą.

W kroku obserwacji często wyróżnia się również podkrok związany z obliczaniem minimalnej entropii i wyborem komórki, która zostanie poddana dalszej części procesu obserwacji.

3.1.2. Propagacja (*propagate*)

W tym kroku komórki sąsiadujące (w znaczeniu sąsiedztwa von Neumanna) z komórką wybraną na wejściu poddawane są procesowi redukcji stanów, które mogą przyjąć. Nowy zestaw stanów jest wynikiem aplikacji reguł sąsiedztwa w ujęciu stanów komórki, z której następuje propagacja, oraz komórki docelowej.

Jeśli w wyniku przeprowadzenia tego kroku komórka sąsiednia zmniejszy swoją liczbę dostępnych stanów, ona również zostaje poddana procesowi propagacji. Oznacza to, że redukcja, która właśnie zaszła, może wymusić redukcję stanów kolejnych granicznych komórek.

Aby możliwe było zakończenie procesu propagacji w jak najmniejszej liczbie kroków, stosowana jest pomocnicza tablica określająca, czy dana komórka została w tym kroku algorytmu odwiedzona. Ze względu na specyfikę tej operacji odwiedzenie każdej komórki więcej niż raz nie spowoduje dodatkowej redukcji stanów.

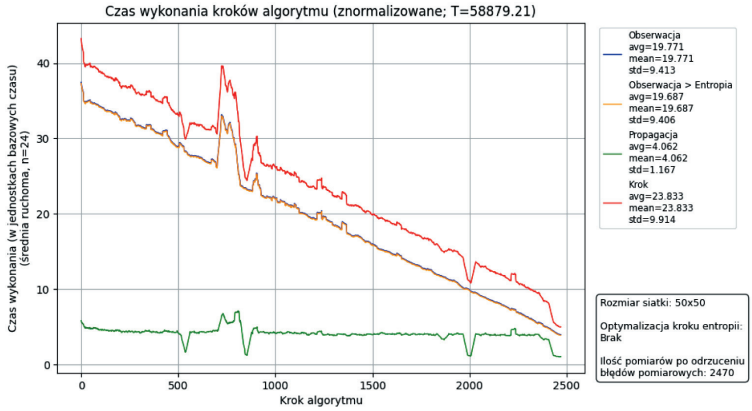
3.2. Czas wykonania

Algorytm uruchomiony bez żadnych dodatkowych optymalizacji pozwala na generowanie treści niewielkich rozmiarów niemalże w czasie rzeczywistym. Widoczne jest jednak znaczne pogorszenie wydajności w przypadku większych rozmiarów generowanej planszy, co zostało ujęte w tabeli 1.

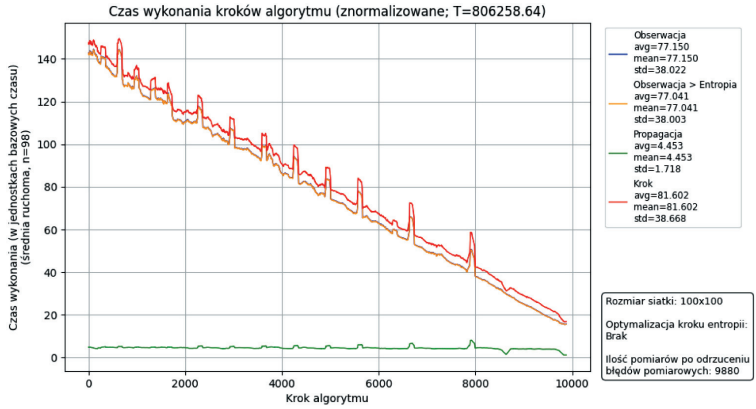
Tabela 1
Czas wykonania algorytmu WFC bez optymalizacji

Wysokość siatki	Szerokość siatki	Czas wykonania (w jednostkach bazowych czasu)
50	50	58 879,21
100	100	806 258,64
200	200	17 580 450,19

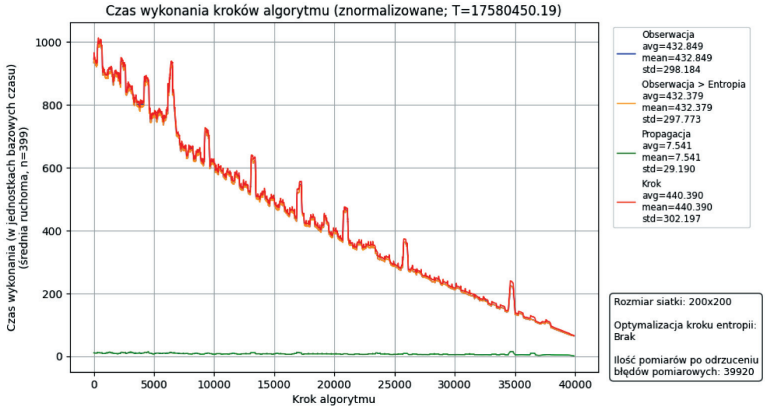
Celem dokładniejszej analizy przebiegu algorytmu oraz znalezienia sposobów optymalizacji procesu na rysunkach 1–3 przedstawiono czas wykonania poszczególnych kroków algorytmu z rozbiciem na etapy obserwacji (z wyszczególnieniem podkroku wyliczania minimalnej entropii) i propagacji.



Rys. 1. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 50 × 50, brak optymalizacji kroku entropii)



Rys. 2. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 100 × 100, brak optymalizacji kroku entropii)



Rys. 3. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 200 × 200, brak optymalizacji kroku entropii)

Z rysunków 1–3 wynika, że w przypadku kroku obserwacji niemal cały czas wykonania przypada na podkrok obliczania entropii. Świadczą o tym znacznie zbliżone do siebie parametry wartości środkowej danych oraz odchylenia standardowego.

Po przeanalizowaniu sposobu działania algorytmu łatwo zauważyć, że to właśnie krok związany z wyliczaniem minimalnej entropii oraz wybieraniem komórki będzie najbardziej wymagający obliczeniowo. Do przeprowadzenia procedury konieczne jest sprawdzenie entropii każdego z pól celem określenia wartości minimalnej, następnie wyznaczenie komórek, które cechują się taką entropią, oraz ostatecznie losowe wybranie tej z nich, która zostanie poddana obserwacji.

Warto również zauważyć, że czas wykonania kroku entropii jest krótszy wraz z kolejnymi krokami algorytmu. Dzieje się tak dlatego, że w procesie obliczania minimalnej entropii i wybierania kandydata do obserwacji uczestniczą komórki, którym nie został przypisany żaden określony stan. Ponieważ w każdym kroku algorytm przypisuje jednej z komórek konkretny stan (podczas kroku obserwacji), naturalne jest, że liczba komórek uwzględnionych w kroku entropii zacznie spadać. Krok ten jest zatem kluczowy dla optymalizacji algorytmu.

4. Optymalizacja algorytmu

Rozdział ten poświęcono przedstawieniu technik optymalizacji algorytmu WFC z wykorzystaniem danych zebranych w wyniku uruchomienia algorytmu w wersji bazowej.

4.1. Zapamiętywanie wartości entropii komórek

Entropia w ujęciu teorii informacji jest opisywana następującym wzorem:

$$H(X) = \sum_{i=1}^{n} p(x_i) \log \frac{1}{p(x_i)} = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{3}$$

gdzie x_i odpowiada konkretnemu stanowi, a $p(x_i)$ określa prawdopodobieństwo jego wystąpienia.

Obliczanie tej wartości na nowo dla każdej komórki jest czasochłonne, szczególnie że nie może się ona zmienić, jeśli komórka nie przeszła procesu redukcji osiągalnych stanów. Pierwszym sposobem na optymalizację podkroku entropii jest zatem ograniczenie liczby wywołań procedur związanych z obliczaniem entropii poszczególnych pól.

Implementacja tego usprawnienia może zostać wykonana na kilka sposobów. W przypadku prezentowanego algorytmu została wykorzystana pomocnicza tablica przechowująca obliczoną wartość entropii każdej komórki.

4.1.1. Implementacja

W celu wdrożenia opisanej optymalizacji wprowadzono następujące zmiany w kodzie:

- Stworzono jednowymiarową tablicę pomocniczą, która przechowuje wartości entropii poszczególnych komórek.
- Zmodyfikowano krok obserwacji, wprowadzając w tablicy pomocniczej aktualizację wartości odpowiadającej obserwowanej komórce. W momencie dokonania obserwacji do tablicy zostaje wpisana maksymalna wartość osiągana przez zmienną typu *float*[2].
- Rozszerzono krok propagacji o logikę, która aktualizuje w tablicy wartość odpowiadającą obecnie przetwarzanej komórce. Dzieje się to wtedy i tylko wtedy, gdy liczba stanów osiągalnych przez komórkę ulegnie zmniejszeniu.
- Zmieniono krok wyliczania entropii, implementując użycie tablicy pomocniczej według algorytmu 3.

```
float lowestEntropy = entropyCache.Min();

if (lowestEntropy == float.MaxValue) {
 // Zwrócenie wartości pustej oznacza osiągnięcie stabilnego systemu
 return null;
}

int randomIndex = Enumerable.Range(0, elementCount).Where(i =>
 entropyCache[i] == lowestEntropy).Random();
return cells[randomIndex];
```

Algorytm 3. Optymalizacja kroku obliczania entropii i wyboru komórki

4.1.2. Czas wykonania

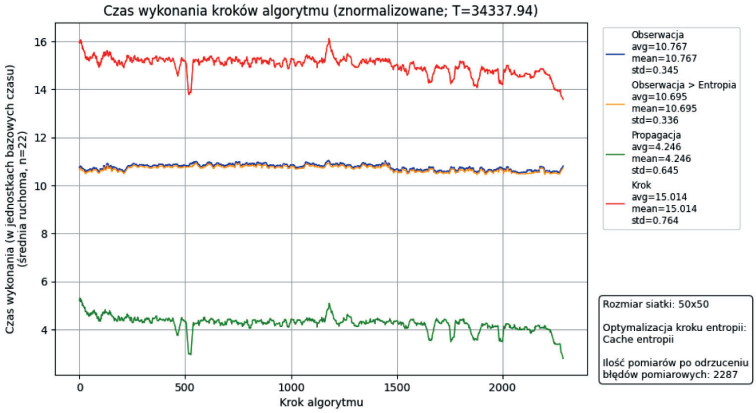
Podobnie jak w przypadku algorytmu w wersji bazowej algorytm z zastosowaniem tablicy pomocniczej do przechowywania obliczonych wartości entropii poszczególnych komórek został poddany testom generowania treści, których czasy wykonania zostały zebrane w tabeli 2.

[2] Warto zadać pytanie, dlaczego nadawana jest taka wartość entropii, jeśli obserwowana komórka ma określony stan, a więc jej entropia równa jest 0. Jest to uproszczenie, które pozwala na łatwiejsze znalezienie komórki z minimalną entropią. Komórki, które zostały poddane obserwacji, są efektywnie usuwane z puli poprzez nadanie im wartości entropii, która jest nie mniejsza niż wartość entropii minimalnej w całym systemie.

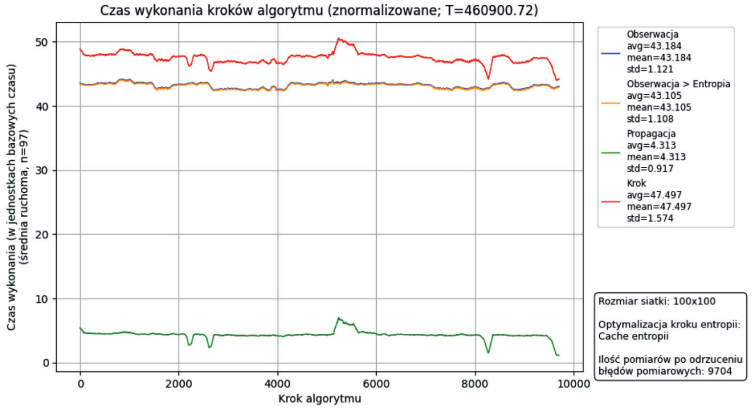
Tabela 2
Czas wykonania algorytmu WFC z zastosowaniem zapamiętywania entropii komórek

Wysokość siatki	Szerokość siatki	Czas wykonania (w jednostkach bazowych czasu)
50	50	34 337,94
100	100	460 900,72
200	200	6 354 468,99

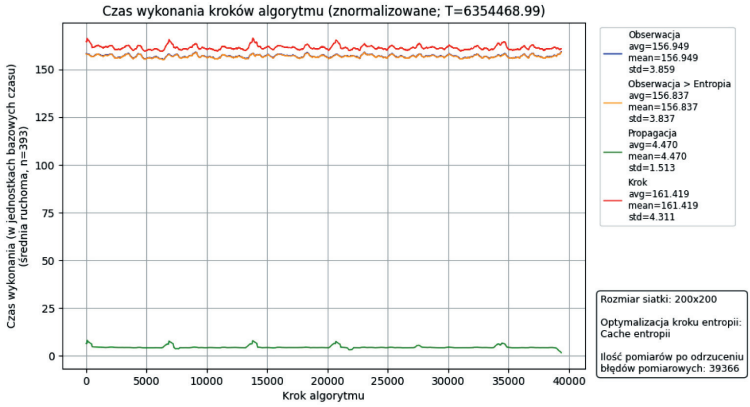
Dodatkowo zmierzono czas wykonania poszczególnych kroków algorytmu, a rezultaty przedstawiono na rysunkach 4–6.



Rys. 4. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 50 × 50, optymalizacja z zastosowaniem tablicy pomocniczej do przechowywania wartości entropii komórek)



Rys. 5. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 100 × 100, optymalizacja z zastosowaniem tablicy pomocniczej do przechowywania wartości entropii komórek)



Rys. 6. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 200 × 200, optymalizacja z zastosowaniem tablicy pomocniczej do przechowywania wartości entropii komórek)

Na rysunkach 4–6 widoczne jest wyraźne zmniejszenie czasu wykonania kroku obliczania entropii, jak również tendencja do utrzymywania stałego czasu egzekucji tej procedury wraz z postępem algorytmu dla danego systemu. Warto jednak zauważyć, że rozwiązanie to nadal jest obciążone istotną wadą: udział obliczeń związanych z entropią w całkowitym czasie iteracji algorytmu diametralnie wzrasta wraz ze zwiększaniem liczby komórek w systemie.

4.2. Zapamiętywanie indeksów komórek z najniższą entropią

Warto ponownie zastanowić się nad zagadnieniem entropii komórki oraz krokiem propagacji. Należy podkreślić, że entropia komórki zależna jest wyłącznie od liczby stanów, które może ona przyjąć, a liczba ta może zostać zredukowana tylko podczas procedury propagacji. Dzięki temu możliwe jest wykonanie dodatkowych optymalizacji, z których najistotniejsze to:

– zapamiętywanie minimalnej entropii w systemie;
– zapisywanie komórek, które osiągają minimalną entropię.

4.2.1. Implementacja

Wprowadzenie optymalizacji polegało na zdefiniowaniu dwóch nowych zmiennych pomocniczych oraz zmodyfikowaniu dwóch procedur:

– Procedura propagacji została rozszerzona o logikę zapamiętywania minimalnej wartości entropii oraz komórek, które ją osiągają, według algorytmu 4.
– Procedura obliczania minimalnej entropii oraz wyboru komórki została zastąpiona nowymi instrukcjami według algorytmu 5.

```
float entropy = entropyCache[index];

if (entropy < systemLowestEntropy) {
 systemLowestEntropy = entropy;
 lowestEntropyIndices.Clear();
}

if (entropy != float.MaxValue && entropy == systemLowestEntropy) {
 lowestEntropyIndices.Add(index);
}
```

Algorytm 4. Optymalizacja kroku propagacji

```
if (lowestEntropyIndices.Count == 0) {
 return GetLowestEntropyOld();
}

float lowestEntropy = systemLowestEntropy;

if (lowestEntropy == float.MaxValue) {
 return null;
}

int randomIndex = lowestEntropyIndices.Random();
lowestEntropyIndices.Remove(randomIndex);

if (lowestEntropyIndices.Count == 0) {
 systemLowestEntropy = float.MaxValue;
}

return cells[randomIndex];
```

Algorytm 5. Optymalizacja procedury obliczania minimalnej entropii i wyboru komórki

Warto zwrócić uwagę na dwa ważne elementy nowej logiki. Po pierwsze, w przypadku pustego zbioru indeksów o minimalnej entropii algorytm zwraca taki sam wynik, jaki zwracał przed zmianą. Po drugie, opróżnienie tablicy pomocniczej skutkuje przywróceniem minimalnej entropii do maksymalnej wartości typu *float*. Gwarantuje to, że każda nowa wartość obliczona w kroku propagacji zostanie uznana za globalne minimum. Podczas implementacji tej optymalizacji wykorzystano oczywiście elementy wprowadzone w ramach poprzedniego usprawnienia.

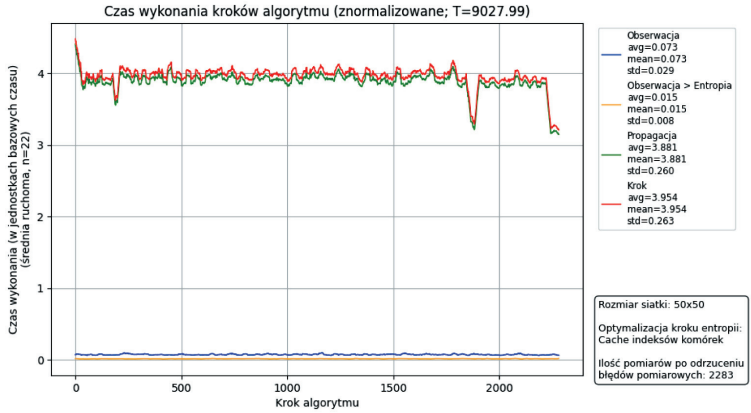
4.2.2. Czas wykonania

W celu weryfikacji finalnej propozycji usprawnienia (z wykorzystaniem listy do zapamiętywania indeksów komórek osiągających minimalną entropię) algorytm został poddany testom generowania treści, których czasy egzekucji zostały zebrane w tabeli 3.

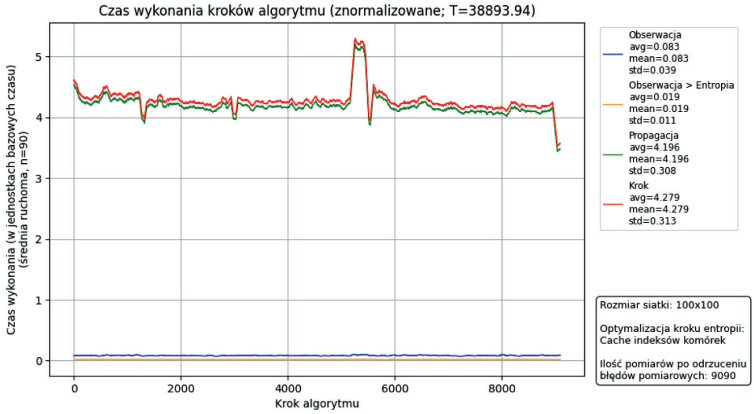
Przedstawione na rysunkach 7–9 czasy wykonań poszczególnych kroków wyraźnie pokazują, skąd wzięło się tak znaczne przyspieszenie algorytmu.

Tabela 3
Czas wykonania algorytmu WFC z zapamiętywaniem indeksów komórek osiągających minimalną wartość entropii

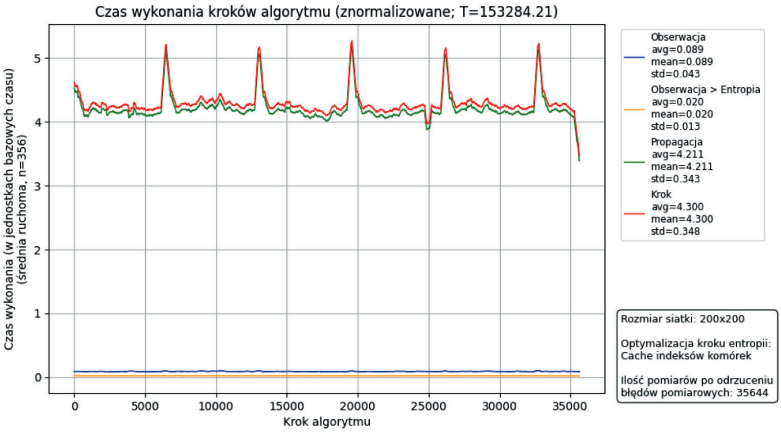
Wysokość siatki	Szerokość siatki	Czas wykonania (w jednostkach bazowych czasu)
50	50	9 027,99
100	100	38 893,94
200	200	153 284,21



Rys. 7. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 50 × 50, optymalizacja z zapamiętywaniem indeksów komórek osiągających minimalną wartość entropii)



Rys. 8. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 100 × 100, optymalizacja z zapamiętywaniem indeksów komórek osiągających minimalną wartość entropii)



Rys. 9. Czas wykonania poszczególnych kroków algorytmu WFC (siatka 200 × 200, optymalizacja z zapamiętywaniem indeksów komórek osiągających minimalną wartość entropii)

Po raz pierwszy zdarzają się przypadki, w których to krok propagacji zajmuje większą część czasu obliczeń w danym kroku algorytmu, a wykonanie kroku obserwacji wymaga nieznacznej ilości czasu.

5. Analiza wyników

Wyniki działania algorytmu w formie bazowej oraz po zaimplementowaniu zaproponowanych optymalizacji zostały zestawione w tabeli 4, która dokładnie obrazuje skrócenie czasu wykonania procedur w każdym z wariantów zmodyfikowanego algorytmu. Wyraźnie widać, że wariant z finalną optymalizacją charakteryzuje się najwyższą wydajnością, a jego przewaga tylko powiększa się wraz ze zwiększaniem rozmiaru siatki.

Tabela 4
Porównanie czasu wykonania poszczególnych wariantów algorytmu WFC

Rozmiar siatki	Wariant algorytmu	Czas wykonania (w jednostkach bazowych czasu)	Krotność poprawy szybkości działania (względem algorytmu bazowego)
50 × 50	podstawowy	58 879,21	1,000
	cache entropii	34 337,94	1,715
	cache indeksów	9 027,99	6,522

Tabela 4 cd.

100 × 100	podstawowy	806 258,64	1,000
	cache entropii	460 900,72	1,749
	cache indeksów	38 893,94	20,730
200 × 200	podstawowy	17 580 450,19	1,000
	cache entropii	6 354 468,99	2,767
	cache indeksów	153 284,21	114,692

Warto również przeanalizować zestawienie średniego czasu wykonania pojedynczego kroku algorytmu dla każdego z jego wariantów przedstawione w tabeli 5. W przypadku finalnej optymalizacji widoczna jest tendencja do osiągania niemalże stałego czasu wykonania pojedynczego kroku algorytmu niezależnie od liczby komórek. Oznacza to, że możliwe jest oszacowanie czasu wykonania algorytmu dla siatki dowolnych rozmiarów.

Tabela 5
Porównanie czasu wykonania pojedynczego kroku dla różnych wariantów algorytmu

Wariant algorytmu	Rozmiar siatki	Czas wykonania (w jednostkach bazowych czasu)	Średni czas wykonania jednego kroku (w jednostkach bazowych czasu)
Podstawowy	50 × 50	58 879,21	23,838
	100 × 100	806 258,64	81,605
	200 × 200	17 580 450,19	440,392
Cache entropii	50 × 50	34 337,94	15,014
	100 × 100	460 900,72	47,496
	200 × 200	6 354 468,99	161,420
Cache indeksów	50 × 50	9 027,99	3,954
	100 × 100	38 893,94	4,279
	200 × 200	153 284,21	4,300

6. Podsumowanie i dalsze badania

Analiza czasu wykonania algorytmu WFC z rozbiciem na poszczególne procedury pozwoliła zaproponować dwie techniki optymalizacji oraz zbadać ich skuteczność.

Zoptymalizowany algorytm (wykorzystujący tablicę pomocniczą do przechowywania entropii poszczególnych komórek oraz zapamiętujący indeksy komórek osiągających minimalną jej wartość) cechował się o wiele wyższą wydajnością niż algorytm w podstawowej wersji, a ponadto wykazywał tendencję do utrzymywania stałego czasu wykonania, który był wprost proporcjonalny do liczby komórek w systemie.

Choć przedstawione techniki optymalizacji pozwalają na wielokrotne przyspieszenie algorytmu, nadal możliwe jest dodatkowe skrócenie czasu obliczeń wymaganych do wygenerowania treści. Można to osiągnąć, modyfikując sposób przeprowadzania procedury propagacji, na którą w finalnym wariancie algorytmu przypadała największa ilość czasu wykonania (o wiele więcej niż na krok obserwacji). Wartym zbadania sposobem optymalizacji jest wykorzystanie programowania dynamicznego w celu przechowywania rezultatów procesu redukcji stanów w zależności od stanów wejściowych.

Literatura

Beukman M., Ingram B., Liu I., Rosman B., 2023, *Hierarchical WaveFunction collapse*, Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 19, no. 1, s. 23–33. https://www.doi.org/10.1609/aiide.v19i1.27498.

Farrokhi Maleki M., Zhao R. (2024). *Procedural content generation in games: A survey with insights on emerging LLM integration*, Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 20, no. 1, s. 167–178. https://doi.org/10.1609/aiide.v20i1.31877.

Fehr M., Courant N., 2018, *fast-wfc*, GitHub. https://github.com/math-fehr/fast-wfc [dostęp: 30.11.2025].

Gumin M., 2020, *WaveFunctionCollapse*, GitHub. https://github.com/mxgmn/WaveFunctionCollapse [dostęp: 4.06.2025].

Karth I., Smith A.M., 2017, *WaveFunctionCollapse is constraint solving in the wild*, [w:] Canossa A., Harteveld C., Zhu J., Sicart M., Deterding S. (eds.), *Proceedings of the 12th International Conference on the Foundations of Digital Games. FDG '17. Cape Cod, MA, August 14–17, 2017*, Association for Computing Machinery. https://doi.org/10.1145/3102071.3110566.

Lazaridis L., Fragulis G.F., 2024, *Creating a Newer and Improved Procedural Content Generation (PCG) Algorithm with Minimal Human Intervention for Computer Gaming Development*, Computers, vol. 13, iss. 11, art. 304. https://www.doi.org/10.3390/computers13110304.

Smith G., 2021, *The future of procedural content generation in games*, Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 10, no. 3, s. 53–57. https://www.doi.org/10.1609/aiide.v10i3.12748.

Togelius J., Yannakakis G.N., Stanley K.O., Browne C., 2011, *Search-based procedural content generation: a taxonomy and survey*, IEEE Transactions on Computational Intelligence and AI in Games, vol. 3, iss. 3, s. 172–186. https://www.doi.org/10.1109/tciaig.2011.2148116.

Villar López M.B.V., Chover M., 2025, *Procedural generation of 3D maps with Wave Function Collapse: optimization and advanced constraints*, [w:] Fellner D.W. (ed.), *XXXIV Spanish Computer Graphics Conference, Jaén, Spain, June 2–4, 2025*, The Eurographics Association. https://www.doi.org/10.2312/ceig.20251107.