

Streszczenie

Temat: Tworzenie map do gier komputerowych za pomocą Algorytmu załamania funkcji falowej

Autor: Rafał Wołyniec 19431

Data Opracowania: 15.05.2023

[1. Wprowadzenie](#)

[2. Algorytm załamania funkcji falowej](#)

[2.1. Faza wejściowa](#)

[2.2. Tworzenie modułów](#)

[2.3. Generowanie wyjścia](#)

[3. Implementacja algorytm do tworzenia map w grach komputerowych](#)

[3.1. Wygenerowane bitmapy](#)

[3.2. Sposób wykorzystania bitmapy do renderowania map](#)

[4. Zakończenie](#)

1. Wprowadzenie

Mapy w grach komputerowych służą stworzeniu przestrzeni, w której gracz jest w stanie przeprowadzać rozgrywkę. Tradycyjnie mapy te były tworzone w całości przez ludzi, proces ten składał się z zaplanowania skali mapy, stworzenia topologii, ustalenia pozycji kluczowych dla rozgrywki, nałożenia tekstur oraz przetestowania mapy. Natomiast w niektórych przypadkach gier, takich jak gry typu sandbox, twórcy gier mogą zdecydować się na wykorzystanie algorytmów generujących mapy. Jednym z takich algorytmów jest algorytm załamania funkcji falowej, który ma za zadanie generowanie dwuwymiarowych obrazów. Te obrazy, zapisane w postaci bitmapy, mogą być dalej przetwarzane przy użyciu różnych technik, aby stworzyć grywalne mapy do gier dwu- lub trójwymiarowych.

2. Algorytm załamania funkcji falowej

2.1. Faza wejściowa

Działanie algorytm zaczyna się od przyjęcia 5 danych:

1. Przykładowa mapa wejściowa w postaci bitmapy: to jest plik bitmapowy o rozmiarze równym lub większym od rozmiaru modułu. Niezależnie od zawartości pliku, algorytm jest w stanie wygenerować plik wyjściowy w postaci bitmapy o większej skali.
2. Wielkość modułu: jest to liczba naturalna N , która określa rozmiar wzorów na mapie wejściowej.
3. Ilość iteracji: to liczba prób, po których algorytm określa, że nie jest w stanie stworzyć deterministycznej mapy dla danego wejścia.
4. Rozmiar bitmapy wyjściowej: jest to liczba pikseli
5. Seed: jest to liczba całkowita, która jest losowo wybierana na początku algorytmu lub podawana jako wejście. Wykorzystanie wartości seeda pozwala uzyskać ten sam wynik końcowy, ponieważ można go użyć do generowania ciągu liczb pseudolosowych, który będzie zawsze identyczny, dopóki wartość seeda nie zostanie zmieniona.

2.2. Tworzenie modułów

Po przyjęciu danych wejściowych, algorytm rozpoczyna proces tworzenia listy modułów, które są podstawowymi elementami konstrukcyjnymi mapy wyjściowej. Każdy moduł składa się z pikseli pobranych z mapy wejściowej. Przykładowo modułami o wielkości 1 piksela dla jakiegokolwiek bitmapy jest każdy piksel znajdujący się na bitmapie wejściowej, natomiast gdy wartość tę zwiększymy do 2 to, każdy moduł będzie miał wielkość 2×2 , czyli 4 pikseli sąsiadujących ze sobą. Wycinanie modułów algorytm zaczyna z pozycji (0,0) na mapie wejściowej. W momencie gdy algorytm wyodrębni wszystkie możliwe moduły oraz przypisze każdemu z nich

możliwe moduły sąsiadujące, inicjalizowany jest plik wyjściowy, który będzie zawierał bitmapę o podanej wielkości.

2.3. Generowanie wyjścia

Generując plik wyjściowy, należy rozpocząć od podzielenia bitmapy na przestrzenie, które będą przyjmować moduły, wychodzi z tego że miejsca powinny mieć taką samą wielkość jak moduły. Następnie algorytm wykorzystując wylosowany lub podany seed losowo wybiera wydzieloną przestrzeń, czyli komórkę w siatce i przypisuje jej losowy moduł. Przed przypisaniem pierwszego modułu, każda komórka może przyjąć dowolny moduł. Jednak, gdy komórka zmienia swój stan na zdefiniowany (czyli przypisany zostaje do niej moduł, który pozostaje niezmienny przez resztę obecnej iteracji), algorytm przechodzi do propagacji tej informacji, rozpoczynając od sąsiadujących komórek. Propagacja polega na ustalaniu jaki moduł może zostać przyjęty przez komórkę. W momencie gdy wszystkie komórki na mapie wyjściowej uaktualniły listę możliwych modułów, dochodzi do znalezienia komórki o najmniejszym stopniu entropii (czyli komórki, której ilość potencjalnych modułów jest najmniejsza). Po ustaleniu komórki o najniższym stopniu entropii, algorytm, używając seeda, losuje kolejne liczby losowe w celu wybrania modułu jaki ta komórka ma przyjąć. W przypadku gdy tych komórek jest więcej niż jedna komórka wybierana jest losowo. Proces ten się powtarza aż do wygenerowania w pełni określonej (zdeteminowanej) bitmapy wyjściowej. Jednak w trakcie trwania algorytmu może dojść do sytuacji, gdy stopień entropii jednej z komórek osiągnie 0. W tym momencie obecna iteracja się zakańcza, a algorytm powtarza generowanie bitmapy od momentu inicjalizacji w kolejnej iteracji. Aby upewnić się, że algorytm jest skończony, na wejściu wymagane jest podanie ilości iteracji, po których algorytm stwierdzi, że dla podanych danych wejściowych niemożliwe jest wygenerowanie bitmapy.

3. Implementacja algorytm do tworzenia map w grach komputerowych

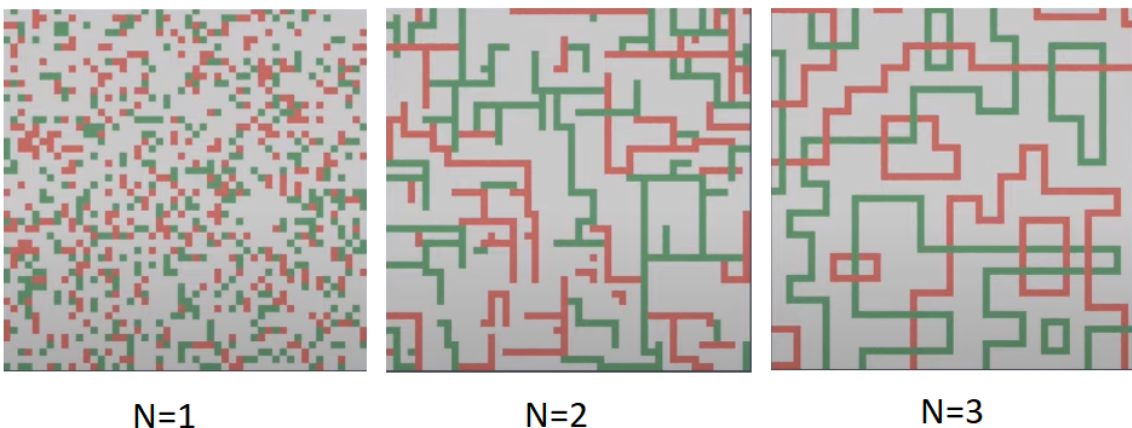
3.1. Wygenerowane bitmapy

Najważniejszymi czynnikami wpływającymi na wygenerowanie bitmapy są mapa wejściowa oraz wielkość modułu (N). Zaczynając od wielkości modułu, zmienna ta definiuje, jak duży fragment wejścia algorytm ma analizować. Aby stworzyć wyjście, w którym można zauważyć zachowane wzory, ważne jest odpowiednie dobranie wartości N (rozmiar modułu). W przypadku, gdy wartość N jest

zbyt mała, obraz wyjścia będzie przypominał szum, natomiast gdy wartość N jest zbyt duża, bitmapa wyjściowa będzie głównie składała się z wzorów znalezionych na wejściu. W praktyce, optymalne dobranie wielkości modułu N zależy od konkretnej sytuacji i charakterystyki danych wejściowych.



rys. 1 Przykładowa bitmapa wejściowa



rys. 2 Przykłady wyjścia oparte na wejściu (rys. 1) w zależności od wielkości modułu i rozmiarze 300 x 300 pikseli

Warto zrozumieć, że wybór odpowiedniej wartości N (rozmiaru modułu) ma istotny wpływ na jakość i charakter wygenerowanej bitmapy. Przy małych wartościach N algorytm może bardziej skupić się na pojedynczych pikselach i generować bardziej kreatywne, ale też bardziej chaotyczne wyniki. Natomiast dla większych wartości N , algorytm bardziej generalizuje i szuka większych wzorców, co może prowadzić do bardziej spójnych i przejrzystych struktur, ale jednocześnie zbyt mocno skupić się na nich i tworzyć niemalże identyczne wyniki. Dlatego też, ważne jest, aby eksperymentować z różnymi wartościami N i odpowiednio dopasować rozmiar modułu do charakterystyki danych wejściowych, aby uzyskać pożądane efekty i generować bitmapę wyjściową, która najlepiej odzwierciedla oczekiwane wzorce i struktury na mapie gry.

3.2. Sposób wykorzystania bitmapy do renderowania map

Jak widać algorytm dobrze sobie radzi w generowaniu obrazów zawierających struktury bądź wzory złożonych z pikseli. Natomiast jak otrzymaną bitmapę zamienić na mapę do potencjalnej gry. W zależności od oczekiwań dotyczących jak skomplikowana ma być mapa lub jak wiele obiektów różnorodnych ma się na niej znajdować, proces ten może się różnić. Biorąc na przykład mapę

wejściową na której znajdują się wyłącznie 2 kolory pikseli czyli wiele pikseli szarych oraz wiele pikseli czarnych. Po wygenerowaniu bitmapy zawierającej kolory, najprostszym podejściem jest zakwalifikowanie koloru szarego jako pustej przestrzeni (np. tła), a koloru czarnego jako ściany. Po uzyskaniu takiej interpretacji mapy, można ją wyrenderować na scenie w grze, wykorzystując odpowiednie biblioteki, takie jak Tilemaps do renderowania map dwuwymiarowych lub bibliotekę UnityEngine do generowania obiektów dla każdego piksela. W przypadku bardziej skomplikowanych map, istnieje potrzeba bardziej zaawansowanych reguł interpretacji kolorów, aby poprawnie rozróżnić różne elementy na mapie. Jednolity kolor może reprezentować różne elementy, w zależności od kontekstu i otaczających go pikseli. Dlatego algorytm przetwarzania musi brać pod uwagę nie tylko wartość danego piksela, ale również wzorce i relacje między sąsiednimi pikselami. W przypadku zielonych pikseli, na przykład, pojedynczy piksel może oznaczać pojedyncze drzewo, ale wiele zielonych pikseli zbliżonych do siebie może tworzyć las. Podobnie, niebieskie linie mogą reprezentować rzeki, a skupiska takich linii mogą stanowić większe formacje wodne, takie jak jeziora lub morza.

4. Zakończenie

Ostatecznie, algorytm załamania funkcji falowej nie generuje bezpośrednio plików wyjściowych z mapami dla gier komputerowych, ale znacząco ułatwia proces ich tworzenia. Zastosowanie tego algorytmu pozwala zastąpić krok tworzenia topologii mapy oraz ustalania kluczowych pozycji w procesie tworzenia mapy dla gry komputerowej.

Ilość technik, z których można skorzystać do przetworzenia otrzymanej bitmapy na mapę, jest ogromna i wymaga wyspecjalizowanego podejścia w zależności od oczekiwań programisty. Warto zauważyć, że technika załamania funkcji falowej jest jednym z wielu narzędzi używanych do generowania map i grafiki w grach komputerowych. Po uzyskaniu spójnej i realistycznej reprezentacji za pomocą tego algorytmu, można skorzystać z różnych technik przetwarzania, np. za pomocą bibliotek do generowania obiektów na mapie, aplikowania tekstur, dodawania detali, tworzenia kolizji i wielu innych. Wybór konkretnych technik i podejść zależy od wymagań projektu i preferencji programisty. Algorytm załamania funkcji falowej stanowi jednak potężne narzędzie do efektywnego generowania map, szczególnie gdy zastosowana jest odpowiednia kombinacja technik, co pozwala na tworzenie różnorodnych, interesujących i unikatowych światów w grach komputerowych."