Warianty gry w życie Conwaya

(Conway's Game of Life variants)

Marcin Rogala

Praca inżynierska

Promotor: dr hab. Jan Otop

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

25 grudnia 2020

Streszczenie

Przedstawiona praca opisuje implementację projektu, umożliwiającego tworzenie i wizualizację wariantów gry w życie Conwaya opartych na planszy zbudowanej na podstawie diagramów Woronoja. Części składowe projektu to generator diagramów Woronoja oparty na algorytmie S. Fortune'a, wizualizacja gry przy zastosowaniu API OpenGl oraz ewaluator języka opisu gry.

This paper describes the project, which alowes, creation and visualization of Conway's Game of Life variants, on a a board made of Voronoi diagrams. Project consist of Voronoi diagrams generator using S. Fortune's algoritm, game visualisation using OpenGl API and game description language evaluator.

Spis treści

Rozdział 1.

Wprowadzenie

1.1. Gra w życie

Gra w życie, została zaprezentowana w 1970 roku przez Johna Conwaya. Jest to gra bez graczy co oznacza, że jej ewolucja zależy tylko od stanu początkowego bez późniejszych ingerencji człowieka. Planszą jest dwu wymiarowa siatka kwadratowych komórek, z których każda ma 8 sąsiadów. Każda komórka może być żywa lub martwa. W standardowej grze obowiązują tylko trzy reguły:

- martwa komórka rodzi jeśli ma dokładnie 3 żywych sąsiadów.
- żywa komórka umiera z zatłoczenia, jeśli ma więcej niż 3 żywych sąsiadów
- żywa komórka umiera z samotności jeśli ma mniej niż 2 żywych sąsiadów

Przedstawione przez Conwaya proste zasady pozwoliły zachować równowagę pomiędzy rozrostem i zanikaniem struktur.

Gra w życie ma wiele ciekawych właściwości. Jedną z nich jest równoważność maszynie Turinga, co oznacza, że ma takie same możliwości obliczeniowe jak komputer z nieskończoną pamięcią i brakiem ograniczeń czasowych. Przy użyciu gry w życie możliwa jest konstrukcja bramek logicznych oraz implementacja różnych systemów komputerowych. Gra może przebiegać chaotycznie lub według jednego z ustalonych wzorców:

- stabilny – pozostają niezmienne bez względu na kolejne przekształcenia
- oscylatory – zmieniają się cyklicznie, regularnie wracają do poprzednich stanów
- statki – oscylatory zmieniające swoją pozycję na planszy

Gra Conwaya jest przykładem automatu komórkowego, które często wykorzystywane są do przeprowadzania symulacji komputerowych. Max Brenner w swoim artykule [1] przedstawił symulację rozprzestrzeniania się wirusa COVID-19, co bardzo dobrze pokazuje możliwości, jakie dają nam modyfikacje gry w życie oraz bardziej rozbudowane automaty komórkowe.

1.2. Opis projektu

Przedstawiona praca opisuje projekt pozwalający na definiowanie zasad gry w życie używając prostego języka. Gra odbywa się na planszy będącej diagramem Woronoja, co istotnie narusza jedną z właściwości automatów komórkowych, mianowicie komórki automatu różnią się od siebie. Pozwala to wprowadzić element nieregularności oraz upodabnia grę do prawdziwego życia, w którym elementy symulacji mają istotne różnice wynikające se swojej budowy lub środowiska. Weźmy za przykład ludzi, którzy mają różną liczbę osób w swoim otoczeniu, co wpływa na wiarygodność symulacji przeprowadzonych na standardowej siatce kwadratów.

Pierwszą częścią pracy jest przedstawienie algorytmu Fortune pozwalającego na generowanie diagramów Woronoja. Algorytm przyjmuje na wejściu zbiór punktów oraz opierając się na technice zamiatania generuje na ich podstawie diagram. Algorytm Fortune rozpatruje względem współrzędnej y następujące wydarzenia:

- site event - miotła napotyka kolejny punkt z wejściowego zbioru
- circle event - miotła napotyka wydarzenie będące niwelacją jednej z paraboli wykorzystywanej do wyznaczania krawędzi diagramu.

Po rozpatrzeniu wszystkich wydarzeń wynikiem działania algorytmu są krawędzie oraz graf sąsiedztwa pól diagramu Woronoja.

Drugą częścią pracy jest opis gramatyki oraz mechanizmu prasowania i ewaluacji języka opisu gry. Parser tworzonych przez użytkowników programów wykorzystuje bibliotekę PEGTL, która opiera się na gramatycę PEG oraz parsowaniu z góry na dół. Wynikiem parsowania jest abstrakcyjne drzewo rozkładu języka, które jest następnie ewaluowane do odpowiednich funkcji opisujących grę w życie.

Kolejna część, to opis wykorzystania API OpenGl do przeprowadzenia wizualizacji gry.

Pracę uzupełnia opis istniejących narzędzi i algorytmów do przeprowadzania symulacji z użyciem gry w życie.

Rozdział 2.

Diagramy Woronoja

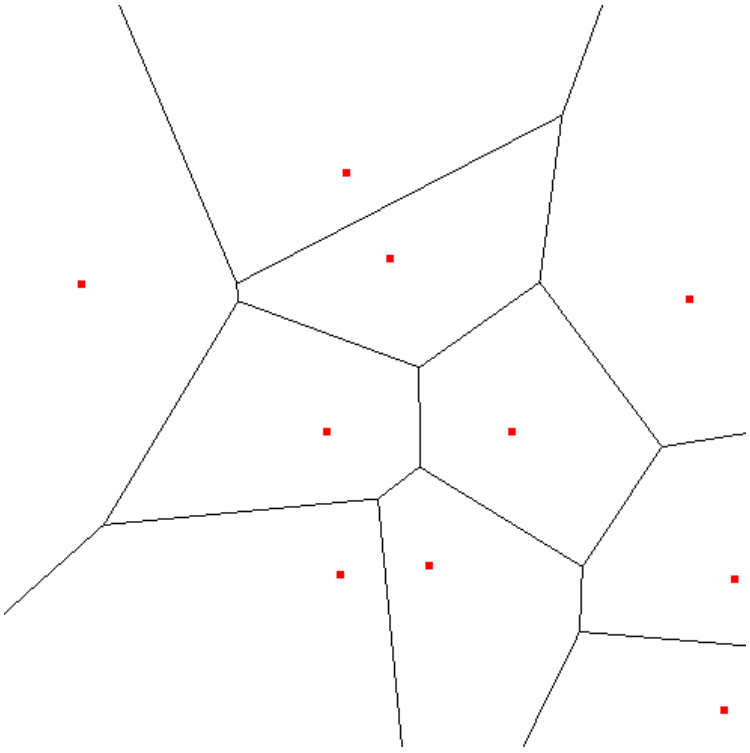
2.1. Definicja

Niech S będzie skończonym zbiorem parami różnych dwuwymiarowych punktów o współrzędnych rzeczywistych. Punkty zbioru S, nazywane będą centrami.

Definicja 1. *Polem diagramu Woronoja* odpowiadającemu punktowi $p \in S$ oznaczamy $Vor_S(p) = \{x \quad | \quad x \in \mathbb{R}^2 \wedge \forall p' \in S \quad dist(x,p) \leq dist(x,p')\}$, gdzie $dist$ oznacza odległość euklidesową dwóch punktów.

Definicja 2. *Diagram Woronoja*, to zbiór $\{Vor_S(p) \quad | \quad p \in S\}$.

Przykład 2.1. *Przykładowy diagram Woronoja z zaznaczonymi* 10 *centrami.*



2.2. Algorytm Fortunea

2.2.1. Opis algorytmu

Jednym ze sposobów generowania diagramów Woronoja jest algorytm zaproponowany przez Johna Fortunea oparty na technice zamiatania [2].

Definicja 3. *Miotłą* nazywamy poziomą linię poruszającą się od dołu do góry planszy. Wszystkie punkty poniżej miotły zostały już rozpatrzone przez algorytm, natomiast punkty powyżej miotły zostaną rozpatrzone, gdy miotła dojdzie do ich wysokości.

Najważniejszą strukturą algorytmu jest towarzysząca miotle linia brzegowa. Składa się ona z parabol, rozpinających się na rozpatrzonymi przez miotłę punktami oraz nieukończonych krawędzi diagramu. W czasie trwania algorytmu parabole rozszerzają się, a ich przecięcia wyznaczają krawędzie. Parabola, która tworzy pole z centrum $p = (x_p, y_p)$ jest zbiorem $\{x \quad | \quad x \in \mathbb{R}^2 \wedge dist(p, x) = dist(p_s, x)\}$, gdzie przez p_s oznaczamy punkt na miotle znajdujący się najbliżej x.

Obserwacja 4. *Punkt przecięcia sąsiednich parabol jest równo odległy od centrów, które definiują te parabole.*

Wyznaczymy wzór $f(x)$ parabol znajdujących się w linii brzegowej. Niech y_s oznacza wysokość, na której znajduje się miotła, a (x_c, y_c) będzie centrum pola, które tworzy parabola.

$$\begin{aligned}
dist((x_c, y_c), (x, f(x))) &= dist((x, y_s)(x, f(x))) \\
\sqrt{(x_c - x)^2 + (y_c - f(x))^2} &= \sqrt{(x - x)^2 + (y_s - f(x))^2} \\
(x_c - x)^2 + (y_c - f(x))^2 &= (y_s - f(x))^2 \\
(x_c - x)^2 &= (y_s - f(x))^2 - (y_c - f(x))^2 \\
(x_c - x)^2 &= 2f(x)y_c - 2f(x)y_s + y_s^2 - y_c^2 \\
(x_c - x)^2 + (y_c^2 - y_s^2) &= 2f(x)(y_c - y_s) \\
f(x) &= \frac{(x - x_c)^2}{2(y_c - y_s)} + \frac{y_c^2 - y_s^2}{2(y_c - y_s)} \\
f(x) &= \frac{(x - x_c)^2}{2(y_c - y_s)} + \frac{y_c + y_s}{2}
\end{aligned}$$

Korzystając z otrzymanego wzoru, możemy w łatwy sposób wyliczać między innymi przecięcia parabol z innymi obiektami w linii brzegowej.

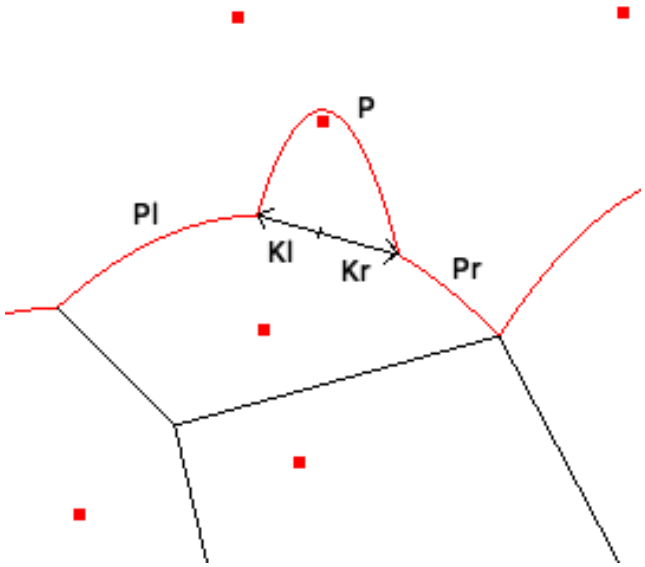
Poruszająca się przez planszę miotła napotyka dwa rodzaje wydarzeń.

- site event – dotarcie do nowego puntu ze zbioru S
- circle event – zniwelowanie jednej z parabol w linii brzegowej

Site event – obsługa

Napotykając nowe centrum, musimy dodać do linii brzegowej nową parabolę, dzieląc jedną z istniejących na dwie części. W tym celu przeszukujemy linię brzegową poszukując paraboli leżącej bezpośrednio pod nowym centrum. To przeszukiwanie będzie miało kluczowy wpływ na złożoność algorytmu.

Przykład 4.1. *Zmienione elementy linii brzegowej po obsłużeniu nowego centrum.*



Odnaleziona parabola zostaję zastąpiona ciągiem elementów:

$$Pl, Kl, P, Kr, Pr \tag{2.1}$$

Gdzie Pl, Pr to kopie rozdzielanej paraboli. P to nowa parabola utworzona przez dodawane centrum. Kl, Kr to nowe krawędzie rozszerzające się w przeciwnych kierunkach. Punktem startowym dodawanych krawędzi jest punkt na rozdzielanej paraboli znajdujący się pod nowym centrum.

Pozostaje sprawdzić, czy parabole, które dodaliśmy do linii brzegowej zostaną kiedyś zniwelowane i czy konieczne jest dodanie odpowiednich wydarzeń. Parabola zostanie zniwelowana przez swoich sąsiadów jeśli zachodzą dwa warunki:

- parabola nie jest skrajnie lewym lub skrajnie prawym elementem linii brzegowej
- krawędzie (półproste) będące sąsiadami paraboli, przecinają się

Aby dodać nowe wydarzenie polegające na ściśnięciu paraboli przez jej sąsiadów, musimy poznać jego współrzędną y. Weźmy ciąg elementów linii brzegowej:

$$P1, K1, P2, K2, P3 \tag{2.2}$$

Punkt $p_i = (x_i, y_i)$ będący punktem przecięcia krawędzi $K1$ i $K2$ jest też miejscem przecięcia parabol $P1$ i $P3$, co czyni y_i wysokością na której $P2$ zostanie ściśnięte przez $P1$ i $P3$. W tym momencie p_i będzie równo odległe od centrów parabol $P1, P2, P3$. Centra te będą leżały na okręgu o środku p_i, skąd wzięła się nazwa wydarzenia.

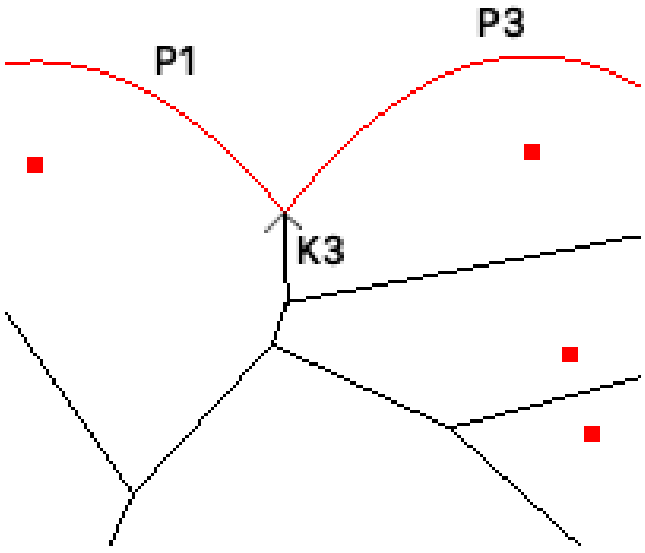
Circle event – obsługa

Rozpatrzmy następujący ciąg elementów w linii brzegowej:

$$P1, K1, P2, K2, P3 \tag{2.3}$$

Załóżmy, że parabola $P2$ zostaje ściśnięta i musimy usunąć ją z linii brzegowej. Leżące po jej bokach krawędzie $K1, K2$ nie będą już rosły. Powinny zostać usunięte i oznaczone jako gotowe krawędzie diagramu. Pozostaje dodać nową krawędź pomiędzy parabolami $P1$ i $P3$ oraz podobnie jak podczas obsługi nowego centrum sprawdzić, czy konieczne jest dodanie wydarzeń niwelacji $P1$ i $P3$.

Przykład 4.2. *Obsłużone wydarzenie, usunięto parabolę $P2$ oraz krawędzie $K1$ i $K2$. W ich miejsce dodana została krawędź $K3$.*



2.2.2. Implementacja

Pseudokod

Poniższy pseudokod pokazuje, bardzo uproszczony sposób rozpatrywania wydarzeń. Wydarzenia trzymane są na stosie, zaimplementowanym przy użyciu *priority_queue* ze standardowej biblioteki języka $C++$.

Algorithm 1: Pseudokod algorytmu

```
Kolejka wydarzeń eventQueue zawiera wszystkie wydarzenia typu site event.
while eventQueue nie jest pusta do
    currentEvent = eventQueue.top();
    if currentEvent jest typu site event then
        handleSiteEvent(currentEvent);
    else
        handleCircleEvent(currentEvent);
    end
end
```

Linia brzegowa

Linia brzegowa – *Beachline*, jest najważniejszą strukturą całego algorytmu. Zawiera ona listę struktur *BeachlineField*, przechowującą elementy znajdujące się w linii brzegowej. Wszystkie funkcje, których złożoność nie została określona, działają ze stałą złożonością. Najważniejsze funkcje *Beachline* to:

- *findParabolaForNewSite* – wyszukująca parabolę z linii brzegowej, która znajduje się bezpośrednio pod nowym centrum. Funkcja oblicza przecięcia parabol, z ich sąsiadami co pozwala poznać przedział na jakim się one znajdują. Złożoność tej funkcji to $O(n)$, gdzie n to rozmiar linii brzegowej.
- *handleSiteEvent* – wykonująca niezbędne operacje opisane w Site event – obsługa. Złożoność tej funkcji to $O(n)$, gdzie n to rozmiar linii brzegowej.
- *handleCircleEvent* – wykonująca niezbędne operacje opisane w Circle event – obsługa.
- *addCircleEvents* – funkcja dodająca niezbędne wydarzenia typu *circle event*, dla nowych elementów linii brzegowej.

Beachline zawiera dużo funkcji pomocniczych, pomagających obsługiwać wydarzenia oraz wykonywać niezbędne obliczenia na parabolach i krawędziach. Niektóre z tych funkcji to:

- *addNewField* – funkcja dodająca nowy element do linii brzegowej.

- *removeField* – funkcja usuwająca element z linii brzegowej.
- *parabolaHalfEdgeIntersection* – funkcja wyznaczająca punkt przecięcia paraboli z półprostą.
- *edgesIntersection* – funkcja wyznaczająca przecięcie półprostych.
- *completePolygons* – funkcja czyszcząca linię brzegową po zakończonej pracy oraz oznaczająca ostatnie krawędzie jako zakończone. Złożoność tej funkcji to $O(n)$, gdzie n to rozmiar linii brzegowej.

Beachline przechowuje także wynikowe krawędzie algorytmu oraz uzyskany graf połączeń pól diagramu.

Pozostałe struktury

- *Event* – przechowuje informacje dotyczące wydarzeń przechowywanych w *eventQueue*.
- *Parabola* – reprezentuje parabolę przechowywaną w linii brzegowej.
- *HalfEdge* – reprezentuję nieukończoną krawędź (półprostą), która rozszerza się w trakcie trwania algorytmu.
- *Edge* – reprezentuje ukończona krawędź.

2.3. Złożoność

Obsługa każdego wydarzenia typu *site event* dodaje dwie nowe krawędzie oraz dwie nowe parabole do linii brzegowej. Załóżmy, że każda parabola ma przypisane wydarzenie typu *circle event*. Obsługa wydarzenia *circle event* zmniejsza liczbę parabol oraz krawędzi. Oznacza to, że dla n pól diagramu w linii brzegowej będzie $O(n)$ elementów oraz w *eventQueue* będzie $O(n)$ wydarzeń. Ze względu, na złożoność wyszukiwania parabol w linii brzegowej – $O(n)$, złożoność całego algorytmu to $O(n^2)$. Możliwe jest osiągnięcie złożoności $O(nlog(n)$, przez zastosowanie zbalansowanego drzewa binarnego do przechowywania elementów linii brzegowej.

2.4. Inne algorytmy

2.4.1. Algorytm dziel i zwyciężaj

Innym podejściem do generowania diagramów Woronoja jest zastosowanie techniki dziel i zwyciężaj. Podczas każdego kroku algorytmu dzielimy zbiór, z którego

generujemy diagram na dwa równe, rozłączne podzbiory. Po rekurencyjnym wywołaniu algorytmu i wygenerowaniu diagramu dla mniejszych zbiorów, wynikowe diagramy są ze sobą łączone. Złożoność takiego algorytmu to $O(nlogn)$.

2.4.2. Algorytm inkrementacyjny

W tym podejściu do gotowego diagramu Woronoja dodajmemy kolejno nowe punkty, aktualizując istniejący diagram. Początkowo dla jednego punktu diagramem jest ciała płaszczyzna. Złożoność algorytmu to $O(n^2)$.

2.4.3. Triangulacja Delone

Definicja 5. Triangulacja Delone zbioru P oznaczana jako $DP(P)$ to taka triangulacja, że żaden punkt $p \in P$ nie leży w środku okręgu opisanego na dowolnym trójkącie należącym do $DP(P)$.

Algorytmy wyznaczające triangulację Delone działają w złożoności $O(nlogn)$. Korzystając z wyznaczonej triangulacji można w łatwy sposób uzyskać diagram Woronoja, łącząc punkty okręgów opisanych na trójkątach.

Rozdział 3.

Język opisu gry

3.1. Główne programy

3.2. Konstrukcje językowe

Rozdział 4.

Parser

Rozdział 5.

Przykłady użycia

Rozdział 6.

Inne narzędzia

6.1. Golly

6.1.1. Opis

6.1.2. Algorytm

6.2. PlayGameOfLife

Bibliografia

[1] Max Brenner. Simulating covid-19 with cellular automata, 2020. [Online; accessed 29-August-2020].

[2] Wikipedia contributors. Sweep line algorithm, 2020. [Online; accessed 17-August-2020].