Cellular automata - modeling sound wave

Natalia Luberda

Marzec 2023

1 Introduction

Raport przedstawia wyniki pracy nad symulacja fali dźwiekowej przy użyciu automatów komórkowych. Celem projektu było stworzenie modelu opisujacego zależność miedzy predkościa czastek a ciśnieniem akustycznym, których zmiany wpływaja na rozchodzenie sie fali dźwiekowej. W trakcie pracy zaimplementowano algorytm oparty na metodzie von Neumanna dla sasiedztwa komórek, a także rozszerzono model o różne typy komórek takie jak ściany i źródła dźwieku. Dodatkowo, wprowadzono możliwość regulacji czestotliwości i amplitudy dźwieku emitowanego przez źródła. W raporcie przedstawiono szczegółowy opis implementacji, a także wyniki i wnioski z przeprowadzonych symulacji.

2 Klasa Point

Klasa Point reprezentuje pojedynczy punkt w siatce automatu komórkowego. Każdy punkt przechowuje informacje o ciśnieniu akustycznym w danym punkcie oraz o predkości czasteczek w otoczeniu punktu. Ponadto, w klasie zdefiniowano zmienne reprezentujace sasiadów punktu w czterech kierunkach :

- północ,
- wschód,
- południe,
- zachód.

W każdym punkcie istnieje również zmienna określajaca typ punktu (0 - powietrze, 1 - ściana, 2 - źródło dźwieku).

```
public class Point {

        public Point nNeighbor;
        public Point wNeighbor;
        public Point eNeighbor;
```

```
public Point sNeighbor;
public float nVel;
public float eVel;
public float wVel;
public float sVel;
public float pressure;
public int sinInput = 0;

public static Integer []types ={0,1,2};
int type;
```

Metody w klasie Point odpowiadaja za aktualizowanie wartości ciśnienia akustycznego oraz predkości czasteczek w otoczeniu punktu.

1. Metoda updatePressure() aktualizuje ciśnienie akustyczne w danym punkcie w zależności od typu punktu - jeśli typ punktu to 0 (powietrze), to ciśnienie akustyczne w punkcie jest uaktualniane na podstawie wartości predkości czasteczek w otoczeniu punktu. W przypadku typu punktu równego 2 (źródło dźwieku), ciśnienie akustyczne w punkcie jest aktualizowane w oparciu o sinusoidalna funkcje zmian ciśnienia o regulowalnej amplitudzie i czestotliwości. Zgodnie ze wzorem:

```
public void updatePresure() {
// TODO: pressure update
sinInput += 3;
float amplituda = 3;
float frequency = 3;
if(this.type == 0){
        this.pressure = (float) /
  ((float)this.pressure - 0.5(nVel+sVel+eVel+wVel));
}else if(this.type == 2){
        double radians = Math.toRadians(sinInput);
        this.pressure = (float) /
  ((float) amplituda*(Math.sin(2*Math.PI*frequency*radians)));
        };
}
```

2. Metoda updateVelocity() aktualizuje wartości predkości czasteczek w otoczeniu punktu w oparciu o wartości ciśnienia akustycznego w sasiadujacych punktach oraz w danym punkcie. W ten sposób symuluje sie propagacje fali dźwiekowej w siatce automatu komórkowego. Zgodnie zw wzorem:

```
public void updateVelocity() {
        if(this.type == 0){
        this.nVel = this.nVel -
 (nNeighbor.pressure-this.pressure);
```

```
            this.sVel = this.sVel -
    (sNeighbor.pressure-this.pressure);

            this.eVel = this.eVel -
    (eNeighbor.pressure-this.pressure);

            this.wVel = this.wVel -
     (wNeighbor.pressure-this.pressure);
            };
}
```

3. Metoda clear() w klasie Point jest odpowiedzialna za ustawienie predkości oraz ciśnienia w punkcie na wartość 0. Jest to cześć algorytmu symulacji mechaniki płynów, gdzie w każdej iteracji należy wyzerować wpływ poprzednich wartości predkości i ciśnienia.W metodzie ustawiane sa wartości predkości w kierunkach północnym (nVel), południowym (sVel), wschodnim (eVel) oraz zachodnim (wVel) na 0. Podobnie, wartość ciśnienia w punkcie (pressure) również jest ustawiana na 0.

```
    public void clear() {
                // TODO: clear velocity and pressure
                this.nVel = 0;
                this.sVel = 0;
                this.eVel = 0;
                this.wVel = 0;

                this.pressure = 0;
        }
```

Klasa Point jest podstawowym elementem modelu symulujacego propagacje fali dźwiekowej w automacie komórkowym. W ramach tego modelu punkty reprezentuja fragmenty ośrodka, w którym fala dźwiekowa sie propaguje, a metody w klasie Point odpowiadaja za aktualizacje stanu tego ośrodka na podstawie jego stanu poprzedniego.

3 Klasa Board:

Główna klasa Board dziedziczy po JComponent i implementuje kilka interfejsów, aby obsłużyć zdarzenia myszy i zmiany rozmiaru komponentu. W konstruktorze, inicjalizowane sa pola points - dwuwymiarowa tablica obiektów Point, o długości length i wysokości height. Nastepnie sa ustawiane listenery dla zdarzeń myszy i zmiany rozmiaru, ustawiane tło na białe i ustawiana flaga opaque na true.

1. Metoda iteration() - jest odpowiedzialna za aktualizowanie predkości i ciśnienia punktów w siatce. Po aktualizacji, wywołuje metode repaint() na obiekcie Board, aby odświeżyć widok.

2. Metoda clear() - ustawia ciśnienie punktów w siatce na 0 i wywołuje metode repaint().

3. Metoda initialize() - tworzy obiekty Point w każdym punkcie siatki, nastepnie przypisuje sasiadów każdego punktu.

4. Metoda paintComponent() - rysuje siatke i kolory w zależności od typu punktu w siatce.

5. Metoda drawNetting() - rysuje siatke w tle, a nastepnie iteruje po wszystkich punktach w siatce i rysuje je w odpowiednim kolorze w zależności od ich typu.

6. Metoda mouseClicked() i mouseDragged() - obsługuja zdarzenia myszy - pobieraja współrzedne klikniecia/dragu i ustawiaja flage lub typ punktu w siatce na podstawie editType, a nastepnie wywołuja repaint().

7. Metoda componentResized() - obsługuje zmiane rozmiaru komponentu - pobiera nowe wymiary komponentu, oblicza nowa długość i wysokość siatki i inicjalizuje siatke z nowymi wymiarami.

Klasa Point jest używana jako element siatki - zawiera pola predkości i ciśnienia oraz odniesienia do sasiadów. Zawiera również metode updateVelocity() i updatePressure(), która aktualizuje predkość i ciśnienie punktu na podstawie predkości i ciśnienia jego sasiadów. Metoda clicked() zmienia flage clicked punktu na true.

4 Zadanie I:

4.1 Co trzeba było zrobić?

W zadaniu należało stworzyć klase Point, w której utworzone zostały 4 zmienne przechowujace informacje o sasiadach w kierunkach północnym, południowym, wschodnim i zachodnim. Nastepnie, w klasie Board, w metodzie initialize() należało zainicjować sasiadów dla każdej komórki, używajac tzw. sasiedztwa von Neumanna, ale bez inicjowania komórek brzegowych.

```
public Point nNeighbor;
        public Point wNeighbor;
        public Point eNeighbor;
        public Point sNeighbor;

for (int x = 1; x < points.length-1; ++x) {
  for (int y = 1; y < points[x].length-1; ++y) {
    points[x][y].eNeighbor = points[x+1][y];
    points[x][y].nNeighbor = points[x][y-1];
    points[x][y].sNeighbor = points[x][y+1];
    points[x][y].wNeighbor = points[x-1][y];
```

```
  }
}
```

W klasie Point należało również utworzyć 4 zmienne przechowujace informacje o predkości czasteczek w kierunku odpowiedniego sasiada.

```
public float nVel;
public float eVel;
public float wVel;
public float sVel;
```

W metodzie clear() należało napisać kod, który resetuje wartości predkości czasteczek i ciśnienia akustycznego.

```
public void clear() {
    // TODO: clear velocity and pressure
    this.nVel = 0;
    this.sVel = 0;
    this.eVel = 0;
    this.wVel = 0;

    this.pressure = 0;
}
```

Nastepnie należało zaimplementować metode updateVelocity() zgodnie z równaniem, które opisuje zwiazek predkości czasteczki z ciśnieniem akustycznym i predkościa czasteczki w danym kierunku w określonej komórce. Podobnie należało zaimplementować metode updatePresure(), która opierała sie na równaniu zwiazanym z maksymalna predkościa fali i przepływem ciśnienia akustycznego w danym miejscu.

```
public void updateVelocity() {
    this.nVel = this.nVel -
    (nNeighbor.pressure-this.pressure);

    this.sVel = this.sVel -
    (sNeighbor.pressure-this.pressure);

    this.eVel = this.eVel -
    (eNeighbor.pressure-this.pressure);

    this.wVel = this.wVel -
    (wNeighbor.pressure-this.pressure);
}

public void updatePresure() {
        this.pressure = (float)
((float)this.pressure - 0.5*(nVel+sVel+eVel+wVel));
}
```

Ostatecznie, należało uruchomić i przetestować symulacje.

4.2 Jaki był efekt?

Po uruchomieniu programu z jednym punktem symulujacym źródło fali, w tle pojawił sie efekt zmiany koloru na biały i czarny, który zilustrował ruch fali dźwiekowej. Przy dodaniu wielu punktów, fale zaczeły interferować ze soba, co doprowadziło do złożonych wzorców falowych, które były zależne od liczby i rozmieszczenia punktów źródłowych.

5 Zadanie 2:

5.1 Co należało zrobić?

Zadanie drugie polegało na rozszerzeniu symulacji fali dźwiekowej poprzez dodanie różnych typów komórek, takich jak powietrze, ściany czy źródło dźwieku.

W tym celu w klasie Point została dodana statyczna tablica typów komórek oraz zmienna typu int.

```
public static Integer []types ={0,1,2};
int type;
```

W konstruktorze klasy Point ustawiono domyślny typ na 0 (powietrze). W klasach Board i GUI zostały odkomentowane fragmenty kodu, które umożliwiaja wybór i rysowanie różnych typów komórek na siatce za pomoca listy rozwijanej.

```
public Point() {
    clear();
    this.type = 0;
}
```

Aby zaimplementować ściany, trzeba było zapobiec wywoływaniu metod updatePresure() i updateVelocity() w komórkach o typie innych niż 0 (powietrze).

```
public void updateVelocity() {
    // TODO: velocity update
    if(this.type == 0){
    this.nVel = this.nVel -
    (nNeighbor.pressure-this.pressure);

    this.sVel = this.sVel -
    (sNeighbor.pressure-this.pressure);

    this.eVel = this.eVel -
    (eNeighbor.pressure-this.pressure);

    this.wVel = this.wVel -
    (wNeighbor.pressure-this.pressure);
    };
}
```

```
public void updatePresure() {
                if(this.type == 0){
                        this.pressure = (float)
    ((float)this.pressure - 0.5*(nVel+sVel+eVel+wVel));
}
```

5.2 Jaki był efekt?

Podczas symulacji zaobserwowano, że fala dźwiekowa odbijała sie od ściany, co wskazuje na poprawne zaimplementowanie funkcjonalności ścian. Dzieki zastosowaniu różnych typów komórek, w tym typu reprezentujacego ściany, możliwe było wizualne odróżnienie różnych cześci siatki oraz zaobserwowanie, jak fale dźwiekowe oddziałuja z poszczególnymi elementami środowiska. To pozwoliło na lepsze zrozumienie zachowania sie fal dźwiekowych w złożonych systemach akustycznych.

6 Zadanie 3:

6.1 Co należało zrobić?

W zadaniu "Sources of sound" należało dodać zmienna "sinInput" typu int do klasy Point,

```
int sinInput = 0;
```

a nastepnie w metodzie "updatePressure()" klasy Point, jeśli komórka jest typu 2, to jej ciśnienie powinno sie zmieniać zgodnie z określonymi warunkami sinusoidalnymi.

```
public void updatePresure() {
    // TODO: pressure update
    sinInput += 3;
    float amplituda = 3;
    float frequency = 3;
    if(this.type == 0){
        this.pressure = (float)
((float)this.pressure - 0.5*(nVel+sVel+eVel+wVel));
    }else if(this.type == 2){
        double radians = Math.toRadians(sinInput);
        this.pressure = (float)
((float) amplituda*(Math.sin(2*Math.PI*frequency*radians)));
    };
}
```

Aby regulować amplitude i czestotliwość emitowanego dźwieku, można było zmieniać wartość zmiennej "sinInput" w każdym kroku symulacji.

6.2 Jaki był efekt?

"Jak można regulować amplitude i czestotliwość emitowanego dźwieku?".

Aby odpowiedzieć na to pytanie w programie, można dodać zmienne, które pozwola na regulowanie amplitudy i czestotliwości dźwieku.

Na przykład, aby regulować amplitude, można zmienić wartość mnożnika w wyrażeniu sinusoidalnym, które jest używane do generowania fali dźwiekowej. W przypadku regulacji czestotliwości można zmienić wartość kata w radianach, który jest przekazywany do funkcji sinus.

W programie można dodać pola do klasy Point, które pozwola na ustawienie amplitudy i czestotliwości. Nastepnie można wykorzystać te wartości do generowania fali dźwiekowej.

Aby regulować amplitude i czestotliwość emitowanego dźwieku, można dodać zmienne, które wpłyna na sinusoidalne zmiany ciśnienia w komórce. Przykładowo, zmiana wartości mnożnika może wpłynać na amplitude dźwieku, natomiast zmiana wartości kata w radianach może wpłynać na czestotliwość dźwieku. W programie można dodać pola do klasy Point, które pozwola na ustawienie amplitudy i czestotliwości, a nastepnie wykorzystać te wartości do generowania fali dźwiekowej.

Po dodaniu źródła dźwieku o sinusoidalnym ciśnieniu, efekt symulacji stał sie bardziej kompletny i realistyczny. Można było zaobserwować wizualne zmiany w postaci skurczów i rozkurczów kropek na planszy, które były rezultatem fali dźwiekowej emitowanej przez źródło. Dodatkowo, po ustawieniu odpowiedniego koloru kropek reprezentujacych źródło, zauważono skrócenie ich średnicy, co wskazuje na ich intensywniejsze wytwarzanie.

Regulacja amplitudy i czestotliwości emitowanego dźwieku była możliwa poprzez manipulacje wartościa zmiennej sinInput, co wpłyneło na amplitude i czestotliwość sinusoidalnych zmian ciśnienia w komórce źródła. W rezultacie, można było uzyskać różne efekty dźwiekowe na planszy. Mogły być one szybsze, a kółka wytwarzane przez źródła innej grubości.

7 Podsumowanie:

Program symulujacy fale dźwiekowa w przestrzeni przyniósł interesujace efekty. Poprzez dodanie możliwości różnicowania typów komórek (powietrze, ściana, źródło dźwieku) oraz zastosowanie sinusoidalnych zmian ciśnienia, program pozwala na generowanie fali dźwiekowej i obserwowanie jej propagacji w czasie rzeczywistym.

Dodanie funkcjonalności ścian pozwoliło na zobaczenie, jak fala odbija sie od przeszkód, co jest ważnym aspektem w analizie propagacji dźwieku w otwartej przestrzeni. Dzieki dodaniu źródła dźwieku, użytkownik ma możliwość regulowania czestotliwości i amplitudy emitowanego dźwieku, co daje jeszcze wieksze możliwości eksperymentowania i obserwacji efektów symulacji.

Wprowadzone zmiany, takie jak zweżenie kółeczek reprezentujacych komórki oraz zwiekszenie ich liczby, wpłyneły pozytywnie na czytelność symulacji i zwiek-

szyły jej atrakcyjność wizualna. Całkowity czas potrzebny na wykonanie symulacji jest krótki, co pozwala na szybka analize efektów zmian parametrów symulacji.

Ogólnie rzecz biorac, program jest skutecznym narzedziem do symulacji fali dźwiekowej i pozwala na wizualizacje propagacji dźwieku w czasie rzeczywistym. Wprowadzone zmiany zwiekszyły atrakcyjność wizualna programu oraz umożliwiły bardziej zaawansowane eksperymenty z parametrami symulacji.