Ročníkový projekt - 2D simulace zvuku v místnosti

Gymnázium Arabská

Felix Navrátil

Vyučující: Mgr. Jan Lána

Předmět programování

21. března 2024

Anotace

Tento ročníkový projekt se zabývá simulováním zvuku ve 2D prostoru pomocí soustředných kružnic. V následujících kapitolách je stručně vysvětleno, jak se zvuk v místnosti pohybuje, odráží a nakonec mizí. Dále je zde vysvětleno, jak se zvuk simuluje v mojí aplikaci a jaké zjednodušení je potřeba zakomponovat do programu, aby se zvuk simuloval efektivně.

Abstract

This year's project focuses on simulating sound in a 2D space using concentric circles. The following chapters briefly explain how sound moves, reflects, and eventually fades in a room. Furthermore, they describe how sound is simulated in my application and what simplifications need to be incorporated into the program to generate sound efficiently.

Obsah

1 Úvod

Jako můj čtvrtý ročníkový projekt jsem si vybral simulování zvuku ve 2D místnosti. Toto téma jsem si zvolil kvůli tomu, že mě fascinuje, jak se zvuk pohybuje, a velmi mě zajímalo, jak by se tato pravidla dala efektivně naprogramovat. Aplikaci jsem programoval v programovacím jazyku java a používal jsem framework javaFX. Při programování jsem narazil na mnoho problémů, mezi které patří například, jak realisticky a efektivně simulovat odraz vlny od zdi, jak simulovat amplitudu vlny v dané vzdálenosti od místa vytvoření vlny, jak mám výslednou vlnu zobrazit na obrazovku nebo jak se bude simulovat interference vln. Při programování jsem kvůli zohlednění efektivity musel zvuk zjednodušit, ale i přes to aplikace ve výsledku umí vytvořit semirealistickou reprezentaci zvukové vlny a to i se simulováním odrazu a interference vln.

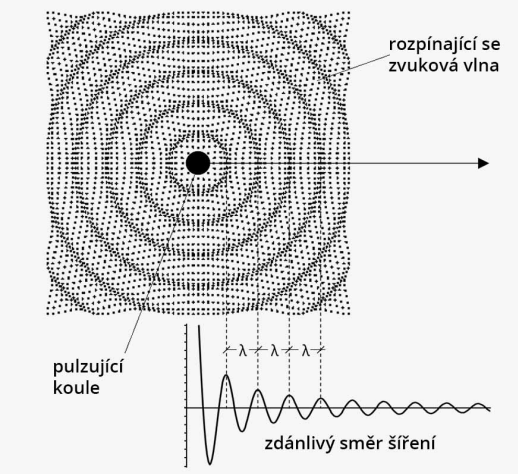
2 Pohyb a odraz zvuku

2.1 Definice zvuku

,, Zvuk je mechanické vlnění v látkovém prostředí, které je schopno vyvolat sluchový vjem. Frekvence tohoto vlnění, které je člověk schopen vnímat, jsou značně individuální a leží v intervalu přibližně 16 Hz až 20 000 Hz. Mechanické vlnění mimo tento frekvenční rozsah sluchový vjem nevyvolává, přesto se někdy také označuje jako zvuk. "[4]

2.2 Pohyb zvuku

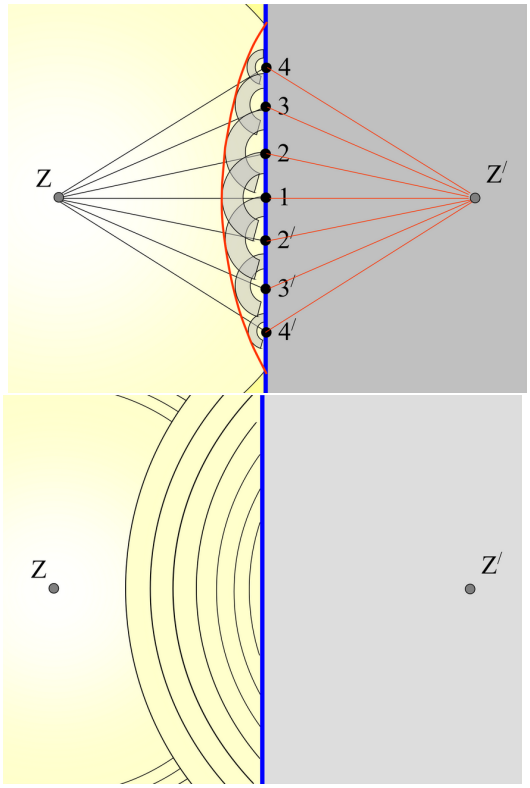
Pokud se zvuk šíří z bodového zdroje (jak simuluje aplikace) šíří ve formě expandujícího torusu, jehož tloušťka odpovídá vlnové délce. Zvukový impuls vytváří prstencovou strukturu, v níž se střídají oblasti s kladnou a zápornou výchylkou, tak jak ukazuje obrázek č. 1. Body stejně vzdálené od středu tvoří kružnici se stejnou výchylkou, označovanou jako vlnoplocha. Vlnění postupně slábne v důsledku zákona zachování energie, přičemž jeho intenzita klesá nepřímo úměrně druhé mocnině vzdálenosti od zdroje impulsu.[5]



Obrázek 1: Zvuk šířící se z bodového zdroje
Zdroj: https://www.audiopro.cz/reseni/co-je-a-jak-se-projevuje-proximity-efekt/

2.3 Odraz zvuku

Když zvuk dopadá na stěnu, tak část jeho energie je absorbována zdí a část je emitovaná z jednotlivých bodů zpět do prostoru. Z těchto bodů se zvuk šíří podle zákonu odrazu, který říká, že úhel dopadu je roven úhlu odrazu. To znamená že pokud se zvuk šíří z bodového zdroje, tak odraz zvuku vypadá jako stejná vlna s menší amplitudou, která je osově symetrická vzhledem k dané stěně viz. obrázek č. 2 a 3. [5]

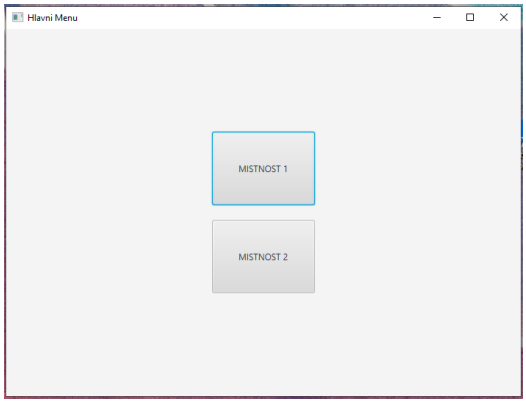


Obrázek 2: schema odrazu zvuku s jednotlivými body

Zdroj: Prezentace: Odraz a lom vlnění (PaedDr. Jozef Beňuška jbenuska@nextra.sk)

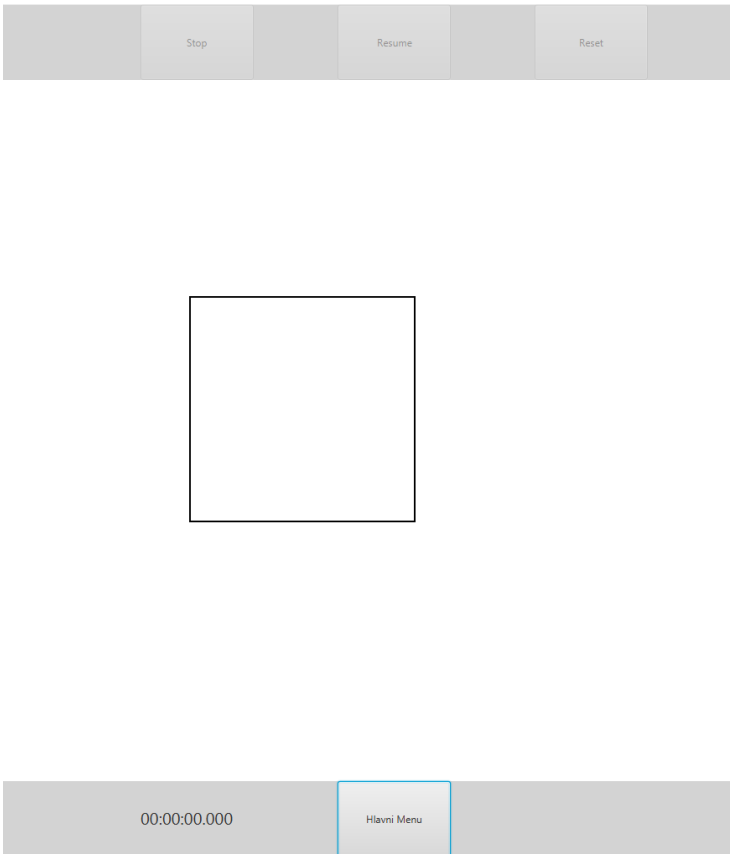
3 Obecné popsání programu

Při spuštění programu se uživateli zobrazí jednoduché a přehledné menu, na kterém jsou 2 tlačítka: *místnost1* a *místnost2* viz obrázek č. 3, pokud uživatel klikne tlačítko je přesměrován do náležité místnosti. *Místnost1* reprezentuje čtvercový tvar a *místnost2* reprezentuje více obdelníkový tvar místnosti. Tak je možné pozorovat různé rozdíly v tom, jak se zvuk šíří v místnostech s různými rozměry.



Obrázek 3: Hlavní menu Zdroj: vlastní

Obě místnosti jsou rozděleny na 3 části grafickou strukturou *VBox*. V první části *VBoxu* jsou tlačítka *Stop*, *Resume*, *Reset*. Tlačítko *Stop* simulaci zastaví, *Resume* ji spustí a *Reset* vymaže všechny vlny z obrazovky. Dále je na scéně zobrazen čtverec který má roli místnosti. A nakonec dole na scéně je tlačítko *Hlavni Menu*, které přesměruje uživatele do hlavního menu a časovač, který slouží na časování toho jak dlouho je simulace spuštěná. Časovač navíc slouží k indikaci, zda program běží plynule, nebo dochází k jeho zasekávání.



Obrázek 4: Místnost 1 Zdroj: vlastní

Pokud uživatel klikne do místnosti vytvoří se vizuální reprezentace expandující zvukové vlny.

3.1 Zjednodušení

Jelikož simulování zvukové vlny je velice náročné na grafický procesor tak bylo nutno vytvořit značně zjednodušený algoritmus, který ale bude simulovat vlnu co nejpřesněji. Mezi tato zjednodušení patří například zanedbání

zmenšování amplitudy nepřímo úměrně s druhou mocninou vzdálenosti od počátku vlny. Toto rozhodnutí jsem udělal, protože by se musela počítat druhá mocnina společně s vzdáleností, což je odmocnina a dvě druhé mocniny, stokrát až tisíckrát za sekundu. Dále jsem byl nucen zanedbat počítání okamžíté výchylky v bodě pomocí vzorečku:

$$y(t) = y_m \sin(2\pi f t + \phi)$$

- y_m - amplituda
- f - frekvence
- t - čas
- ϕ - fáze

Rozhodl jsem se tak provést, protože počítání sinu společně s násobením desetinných čísel skoro pro každý *Pixel* v místnosti je také velmi náročné. Místo toho je vlna rozdělena na několik kružnic, z nichž každá přidá či odebere pixelům okamžitou výchylku (viz kapitoly 4.11 a 4.14). To efektivně vytvoří vizuální relativně přesnou vizuální reprezentaci vlny bez toho, aniž by se program zasekával. Také bylo nutno počítat okamžitou výchylku i ostatní hodnoty důležité pro generaci vlny pouze v integerech/ celých číslech, protože počítání v doublech/ desetinných číslech by také vyústilo v sekání programu. Jelikož je vše počítané v integerech tak ne všechny vlnové délky nebo amplitudy jsou povoleny. Proto jsem se rozhodl tyto hodnoty definovat přímo v kódu jako neměnné konstanty a to na následující hodnoty.

- vlnová délka / $deltaR = 60$
- amplituda = 100

4 Třídy

4.1 Launcher

Třída *Launcher* je potomek třídy *Application*, která je základní třídou potřebnou k vytvoření JavaFX aplikace. To znamená, že musí mít metodu *start*, která je zodpovědná za samotné spouštění aplikace. Tato třída má pouze dvě metody viz. níže.

```
    @Override
public void start(Stage stage) throws IOException {
    MainMenu hlavniMenu = new MainMenu();
    hlavniMenu.setScene(stage);
}

public static void main(String[] args) {
    launch();
}
```

4.2 MainMenu

Třída *MainMenu* se stará o vytvoření scény a definování její velikosti. Vytvořenou scénu následně zobrazí viz. obrázek 1. Tato třída má pouze jednu metodu *setScene(Stage stage)*, která plní následující:

- přiřazení správný *fxml* soubor
- vytvořit scénu *hlavniMenu*

- vytvořit controller *hlavniMenuController*
- předat controlleru velikost scény společně se *stagí*
- zobrazit *stage* a scénu

4.3 MainMenuController

Tato třída slouží jako controller ke třídě *MainMenu*, to znamená že se stará o dynamické aktualizování *VBoxu* (grafické rozhraní) a uživatelské interakce pomocí tlačítek *setSceneDimensions(int height, int width)*, *setStage(Stage stage)*, *updateLayout()*, *handleButtonRoom0Click()* a *handleButtonRoom1Click()*.

setSceneDimensions(int height, int width)

Předává *stage* z třídy *MainMenu* do controlleru. To umožnuje controlleru starat se o změnu *stage* nebo změnu scény.

updateLayout()

Toto je privátní metoda, která je zodpovědná za vypočítávání rozměrů a dynamické aktualizování *VBoxu* vzhledem k velikosti scény.

handleButtonRoom0Click()

Metoda se zavolá pokud uživatel klikne na tlačítko s názvem *MÍSTNOST 1*. Metoda vytvoří novou instanci třídy *Room0* a přesměruje uživatele na příslušnou scénu. To samé dělá i metoda handleButtonRoom1Click(), s rozdílem že tato metoda přesměruje uživatele do jiné místnosti.

4.4 Room0 a Room1

Tyto dvě třídy jsou zodpovědny za inicializaci počáteční velikosti scény, načítání příslušného *FXML* souboru, vytváření a zobrazování scény a *stage*, vytváření controlleru a připisování controlleru k *fxml* souboru. Třídy mají tři metody *setScene(Stage stage)*, která se stará o vše výše zmíněné, *getMistnost0ScreenHeight()*, *getMistnost0ScreenWidth()*. *Room1* vytváří scénu ze 4. obrázku a *Room1* vytváří scénu velmi podobnou s jediným rozdílem a to že místo čtvercové místnosti je tam obdelníková.

4.5 BaseRoomControllerInterface

Interface, který musí implemenovat každý *RoomController*. Implementace je nutná kvůli většímu počtu místností společně s předáváním a získáváním informací ze tříd *SoundWave*, *WaveManager* atd. V interfacu jsou následující metody: *setStage(Stage stage), initialize(), getXMin(), getXMax(), getYMin(), getYMax(), overlayRectangles(), updateLayout(), getRoomWalls(), getRoomCorners() a getStroke()*

4.6 Room0Controller

Tato třída se stará o logiku, rozvržení a interakce se uživatelem. Přiřadí funkci jednotlivým tlačítkám, spravuje časovač a pokud uživatel klikne do místnosti tak pomocí ostatních tříd jako *WaveManager, PixelManager, SoundWave ...* vytvoří novou zvukovou vlnu. Controller má následující metody:

setStage(Stage stage)

Nastaví hlavní okno aplikace pro scénu Room0.

initialize()

Inicializuje scénu, UI komponenty a další proměnné (např. stav obdélníku nebo tlačítek) při načtení ovladače.

createTimeline()

Spustí časovač pro správu časově závislých událostí ve scéně.

initializeRectangle(double x, double y)

Nastaví obdélník v místnosti na základě zadaných rozměrů.

getXMin(), getXMax(), getYMin(), getYMax()

Vrátí minimální a maximální souřadnice x/y místnosti pro určení hranic nebo umístění prvků rozvržení.

updateLayout()

Dynamicky upravuje velikost a pozici tlačítek nebo UI komponent místnosti podle jejích rozměrů.

createOverlay() a overlayRectangles()

Odpovídají za vykreslení překrývajících obdelníků.

initializeLines(double xMin, double xMax, double yMin, double yMax)

Vytváří nové přímky pomocí třídy *Line*, které reprezentují stěny místnosti. Tyto přímky jsou používány při počítání odrazu vlny, konkrétně k vytvoření vlny symetrické vůči dané stěně. Dále vytvoří seznam těchto přímek *roomWalls* a rohů místnosti *roomCorners*.

getRoomWalls()

Vrátí seznam čar představujících stěny místnosti.

getRoomCorners()

Vrátí seznam bodů představujících rohy místnosti.

handleButtonHlavniMenuClick()

Přepne scénu zpět do hlavního menu.

handleButtonStopClick()

Zpracovává uživatelskou interakci k zastavení časovače a vln.

handleButtonResumeClick()

Obnoví pozastavenou aktivitu ve scéně .

handleButtonResetClick()

Obnoví stav místnosti do původního nastavení.

updateTimerLabel()

Tato metoda odpovídá za aktualizování časovače.

handlePaneClick(MouseEvent event)

Zavolá se pokud uživatel klikne na *centerPane*, pokud klikne do místnosti tím je myšleno do vykresleného čtverce tak pomocí třídy *WaveManager* vytvoří zvukovou vlnu. Nejprve se vytvoří animace a potom se volá metoda *updateWaves* ve třídě *WaveManager*.

4.7 Room1Controller

Třída Room1Controller je velmi podobná třídě Room0controller. Metody: *setStage(Stage stage), initialize(), createTimeline(), getStroke(), getRoomWalls(), getRoomCorners(), getXMin(), getXMax(), getYMin(), getYMax(), initializeLines(double xMin, double xMax, double yMin, double yMax), updateLayout(), createOverlay(), overlayRectangles(), updateTimerLabel(), handleButtonHlavniMenuClick(), handleButtonStopClick(), handleButtonResumeClick(), handleButtonResetClick(), handlePaneClick(MouseEvent event)* plní stejnou funkci jako ve třídě Room0Controller viz. kapitola 4.6. Pouze třída *initializeRectangle(double x, double y)* je rozdílná a to v tom, že místo čtverce se vytvoří obdelník.

4.8 Line

Toto je jednoduchá třída, reprezentující přímku v rovině. V programu se používá při reprezentaci stěn a při výpočtu odrazu vlny. Třída má 3 atributy *A, B, C*, které značí hodnoty v obecné rovnici přímky:

$$Ax + By + C = 0$$

Ve třídě je konstruktor, kde se hodnoty nastaví a potom 3 *gettery*, z nichž každý vrací právě jednu hodnotu (*A, B* nebo *C*).

4.9 Point

Point je třída která reprezentuje bod rovině. Má 2 atributy x a y, ty značí souřadnice na scéně, které fungují stejně jako souřadnice v kartézské souřadnicové

soustavě. Tudíž se zde mohou používat stejné výpočty jako v analytické geometrii.

Jelikož má třída dva konstruktory, tak je možné instanci této třídy inicializovat dvěma způsoby. První způsob je pomocí zadání samotných souřadnic a druhý je pomocí průsečíku dvou přímek (*Line*). Průsečík počítá třída Calculator, která je popsána v kapitole 4.10. Dále má třída následující metody: *getX(), getY(), setX(double x), setY(double y), toString(), distance(Point other).*

4.10 Calculator

Calculator je třída, která vypočívává souřadnice bodů. Má dvě metody: *calculateIntersection(Line line1, Line line2)*[2], která zjistí souřadnice průsečíku dvou přímek a *calculateSymetricPoint(Point point, Line line)*[3], jenž zjistí souřadnice osově souměrného bodu vůči danné přímce. Obě metody vrací *Point* a vypadají následovně:

```
public Point calculateIntersection(Line line1, Line line2) {
    // Extract coefficients from each line
    double A1 = line1.getA();
    double B1 = line1.getB();
    double C1 = line1.getC();
    double A2 = line2.getA();
    double B2 = line2.getB();
    double C2 = line2.getC();
    // Calculate determinant
    double determinant = A1 * B2 - A2 * B1;
    // If determinant is 0, the lines are parallel or coincident
    if (determinant == 0) {return null;}
```

```
        // Calculate the intersection point
        double x = (B1 * C2 - B2 * C1) / determinant;
        double y = (A2 * C1 - A1 * C2) / determinant;
        return new Point(x, y);
    }
    public Point calculateSymetricPoint(Point point, Line line){
        // Extract line coefficients
        double A = line.getA();
        double B = line.getB();
        double C = line.getC();
        // Extract point coordinates
        double x = point.getX();
        double y = point.getY();
        // Calculate projection point
        double denominator = A * A + B * B;
        if (denominator == 0) {throw new IllegalArgumentException("Invalid
        line coefficients:A and B cannot both be zero.");}
        double x_proj = (B * (B * x - A * y) - A * C) / denominator;
        double y_proj = (A * (-B * x + A * y) - B * C) / denominator;
        // Calculate symmetric point
        double x_sym = 2 * x_proj - x;
        double y_sym = 2 * y_proj - y;
        // Return the symmetric point
        return new Point(x_sym, y_sym);
    }
```

4.11 Pixel

Třída *Pixel* znázorňuje zvětšený individuální pixel v místnosti. Je potomkem třídy *Rectangle* to proto, aby byla jednoduchá změna barvy či přidávání na

scénu. Třída má následující atributy:

- *gridX/Y* - pozice v gridu neboli 2D poli místnosti
- *realX/Y* - reálná pozice na scéně
- *celkovaVychylka* - celková výchylka v bodě
- *PIXELSIZE* - velikost pixelu
- *rectangle* - vizuální reprezentace pixelu

Celkem má třída 8 metod, z nichž každá plní specifickou funkci.

setPixelSize(int pixelSize)

Nastaví velikost pixelu.

getGridY/X(double realY/X, double y/xMIn)

Vrátí hodnoty *gridX/Y*.

getRectangle()

Vrátí samotní obdelník.

setColor(int celkovaVychylka)

Přiřadí barvu vzhledem k velkové výchylce.

setDefault()

Nastaví celkovou výchylku na 0.

addVychylka(int vychylka)

Přidá výchylku do celkové výchylky.

createColor(int celkovaVychylka)

Přiřadí barvu k celkové výchylce pomocí if - elseif řetězce. Pokud je celková výchylka menší než -20 tak metoda vrátí modrou barvu (čím je to menší tím je modrá víc sytá), když je celková výchylka větší než 20 tak metoda vrátí červenou (čím je výchylka větší tím je červená sytější) a pokud je hodnota mezi -20 a 20 tak se vrátí bílá barva. Metoda přiřadí výchylku do intervalu od 400 do -400. Podle toho jak velká výchylka bude, tak se ji přiřadí příslušná barva.

getCoordinates()

Vrátí instanci třídy *PixelCoordinate* se souřadnicemi pixelu viz. kapitola 4.12.

4.12 PixelCoordinate

Toto je pomocná třída, která slouží pouze jako nástroj pro ukládání souřadnic pixelů v gridu. Má 2 atributy x a y, ty slouží jako souřadnice. Dále má třída 2 *gettery* každý buď pro x nebo y, metodu *equals(Object obj)*, která se používá při lokaci duplikátů v algoritmu na přiřazování pixelů dané kružnici viz. kapitola 4.14. Jako poslední metoda, kterou tato třída má je *hashCode()*, jenž se také používá při lokaci duplikátů.

4.13 PixelManager

Tato třída slouží ke spravování celého 2D pole pixelů. Mezi její funkce patří například inicializace pole, aktualizování jednotlivých pixelů nebo resetování všech neaktivních pixelů. Mezi klíčové atributy patří:

- *Pixel[ ][ ] pixelGrid* - představuje 2D pole pixelů v místnosti
- *int width* - šířka pole
- *int height* - výška pole
- *BaseRoomControllerInterface roomController* - zpřístupňuje metody a proměnné z jednotlivých controllerů např. *Pane centerPane*
- *int PIXELSIZE* - konstanta reprezentující velikost jednoho pixelu

Třída má celkem 6 metod.

initializePixelGrid(int rectWidth, int rectHeight, double rectX, double rectY)

Tato metoda inicializuje *pixelGrid*, to znamená, že se připíšou jednotlivé pixely na správně pozice do místnosti i do *pixelGridu*. Metoda se volá ve třídách *Room0/1Controller*, konkrétně v metodě *initializeRectangle*.

addRectanglesToPane(Pane pane)

Přidá všechny pixely na scénu pomocí *pane*. Je také použita ve třídách *Room0/1Controller* v metodě *initializeRectangle*.

resetPixelGridDisplacement()

Zodpovídá za resetování všech pixelů. To znamená, že pro každý pixel zavolá metodu *setDefault()*. Metoda se volá v *Room0/1Controllerech* v metodě *handleButtonResetClick()*.

resetAllInactivePixels(Set activePixelCoordinates)

Účel této metody je resetování všech pixelů, které nejsou momentálně v žádné vlně. Metoda projde celý *pixelGrid* a pro každý pixel získá jeho *PixelCoordinate* a zjistí zda jsou dané souřadnice napsané v setu *activePixelCoordinates*. Pokud jsou, tak se nic neděje a pokud nejsou, je daný pixel vynulován (zavolá se metoda *setDefault()*)

getPixelSize(), getPixelGrid()

Slouží jako *gettery* pro *PIXELSIZE* a *pixelGrid*.

4.14 SoundWave

Soundwave je potomkem třídy *Circle* a reprezentuje individuální expandující zvukovou vlnu. Zvuková vlna má tvar expandujícího torusu/donutu, který je znázorněn jako oblast mezi vnějším a vnitřním kruhem (*outerCircle* a *innerCircle*). Třída také interaguje s *pixelGridem* pomocí třídy *PixelManager* a *WaveManager* a zjišťuje, jakým pixelům má přidat výchylku. Dále zjišťuje body a momenty odrazu vlny a způsobuje samotnou expanzi. Jelikož je třída velmi obsáhlá (cca 500 řádků kódu a 30 metod), tak zde není popsaná každá metoda zvlášť jako u předchozích tříd, ale je popsaná jen logika.

4.14.1 Inicializace

Nová vlna se vytvoří, pokud uživatel klikne do místnosti, nebo se vlna odrazí od zdi. Když se tvoří nová instance, tak se nejprve zavolá konstruktor, který nastaví střed (vytvoří se *Point* se souřadnicemi středu), poloměr, controller,

stěny místnosti (xMin/Max a yMin/Max), časovač, osy kružnice, tj. 2 přímky (*Line* viz. kapitola 4.8), rohy místnosti a body, kde se vlna odrazí od stěn. Body odrazu se hledají jako průsečíky os kružnice a stěn místnosti. Používá se k tomu metoda *getIntersections()*. Tato metoda zjistí, kde se nachází střed vlny a pomocí třídy *Calculator* najde již zmíněné průsečíky. Průsečíky se vrátí jako pole *Pointů*, toto pole je klíčové k počítání odrazu vlny.

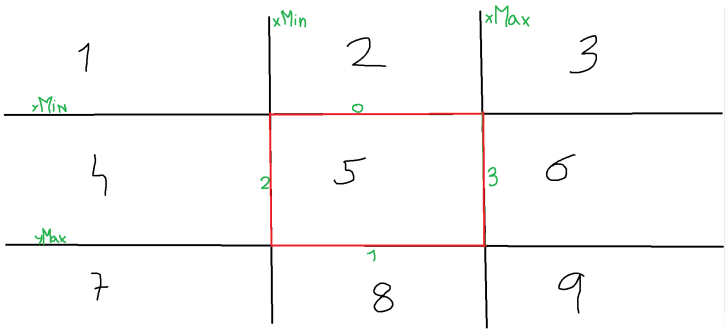
4.14.2 Expanze

O expanzi vlny se stará metoda *grow()*. Tato metoda nejprve zvětší poloměr vnější kružnice *outerrRadius* o 1, potom zjistí, jesli je větší než *deltaR* (rozdíl poloměru vnitřního a vnějšího kruhu) neboli délka zvukové vlny. Jestliže je větší, tak zkontroluje zda existuje vnitřní kruh, pokud ne tak se vnitřní kruh vytvoří a pokud ano tak se poloměr vnitřního kruhu *innterRadius* také zvětší o 1.

Dále se zkontroluje jestli je *outerRadius* dělitelný *PIXELSIZE* a zároveň simulace běží, pokud jsou tyto podmínky splněny, tak se vymažou hodnoty ze setu *savedSetOfPixelCoords*, zavolá se metoda *generateWaveDonut()* (viz. kapitola 4.14.4) a hodnoty ze setu *activePixelCoordinates* se přepíšou do setu *savedSetOfPixelCoords*. Nakonec se zavolá metoda *resetVisitedPixels()*, která vymaže všechny hodnoty ze setů *visitedPixels, duplicatePixels, activePixelCoordinates*.

4.14.3 Odstraňování

Vlna se odstraní pokud je její amplituda menší než nula nebo je starší než *maxAge* (8 sekund). *maxAge* byl zvolen jako 8 sekund, protože tolik trvá



Obrázek 5: Schéma místností Zdroj: vlastní

innerCircle, než se expanduje pryč z místnosti.

4.14.4 Odraz

Zvuková vlna se odrazí, pokud se dotkne stěny. Tato logika v programu funguje následovně. Nejprve se zjistí, při jakých poloměrech se vlna odrazí od zdi, neboli jak moc je vlna vzdálená od každé zdi. Tyto vzdálenosti najde metoda *getReflectionDistances*, která vrací pole celých čísel / integerů. Nejdříve metoda zjistí v jakém sektoru scény (viz. obrázek 5) leží pomyslný střed (v případě kdy je vlna vytvořena kliknutím uživatele, tak reálný střed). Vlna se může nacházet v devíti různých sektorech 1-9, které tvoří 3 logicky podobné skupiny (první je 5, druhá je 2,4,6,8 a třetí je 1,3,7,9). Skupiny jsou si podobné tehdy když mají osy vlny v daných skupinách stejný počet průsečíku se stěnami místnosti. Pro první skupinu jsou průsečíky 4, pro druhou je jeden a pro třetí je jich 0. To, jestli střed leží v daném sektoru, zjišťují metody: *isInRectangle, isBellowRectangle,isAboveRectangle, isLeftOfRectangle, isRightOfRectangle, isAboveOnTheRightOfRectangle, isAboveOnTheLeftOfRectangle, isBellowOnTheRightOfRectangle, isBellowOnTheLeftOfRectan-*

gle. Tyto metody vrací boolean a jsou použity jako podmínky v if-else if řetězci v metodě *getReflectionDistances*.

Jako první a nejjednodušší na počítání je sektor 5, střed vlny je reálný a nachází se v místnosti. Vzdálenosti se tehdy určují pouze jako vzdálenosti průsečíků os se stěnami a středem. Tyto vzdálenosti jsou 4 tím pádem metoda vrátí pole integeru o velikosti 4. Dále s těmito vzdálenostmi pracuje třída *WaveManager*.

Pro druhou skupinu neboli sektory 4,8,6,8 se ve třídě *SoundWave* počítá pouze jedna vzdálenost a to ta od středu k průsečíku opačné strany. V tomto případě metoda vrací pole o velikosti 1 s danou vzdáleností, ostatní vzdálenosti počítá třída *WaveManager*.

Třetí skupina se počítá jinak než první 2. Jelikož je střed vlny diagonálně od místnosti, tak osy vlny nemají se stěnami místnosti žádný průsečík, proto je zde potřeba uplatnit jiný výpočet odrazových vzdáleností. Protože se vlna šíří jakoby z rohu místnosti, tak se vždy dotkne první rohů opačných stěn. Takže je nutné vypočítat pouze vzdálenosti k ostatním rohům místnosti. Počet těchto vzdáleností je 3 takže metoda vrací pole o velikosti 3.

4.14.5 Vykreslování

Zvuková vlna má tvar donutu, tento tvar program chápe jako prostor mezi vnějším (*ounterCircle*) a vnitřním kruhem (*innterCircle*). Prostor je rozdělen na 20 (včetně *innerCircle* a *outerCircle*) podkružnic. Počet těchto podkružnic je roven podílu velikosti pixelu (*PIXELSIZE*) a vlnové délky / rozdílu poloměru vnějšího a vnitřního kruhu (*deltaR*).

Každá z těchto podkružnic zvýší nebo sníží celkovou výchylku daného pixelu

o *amplituda/5*. Prvních 5 ji zvýší, druhých 10 ji sníží a zbytek ji zvýší. Toto vytvoří iluzi sinusoidy s jednoduchou a velmi efektivní implementací v kódu. Konkrétně to funguje následovně.

Pokaždé když se zavolá metoda *grow()* a *outerRadius* je dělitelný *PIXELSIZE*, tak se volá metoda *generateWaveDonut()*. Tato metoda nejprve vypočítá aktuální délku vlny (*width*) a potom vypočítá celkový počet kruhů (*numberOfCircles*), ten se spočítá jako podíl délky vlny a *PIXELSIZE*. Následovně je spuštěn for-cylkus, který běží od nuly do *numberOfCircles*. V tomto for-cyklu se nejprve vypočítá poloměr momentální podkružnice,

```
int radius = innerRadius + ((numberOfCircles - i-1) * PIXELSIZE)
```

potom její okamzitaVychylka (*amplitudeForCirrcle*) pomocí metody *calculateAmplitudeForCircle(int circleIndex)*. Tato metoda vrátí buď *amplitude/5*, pokud je index menší než 5 nebo větší než 15, nebo *-amplitude/5,* pokud je hodnota indexu jiná.

Dále se zavolá metoda *drawCircleWithOkamzitaVychylka(int centerX, int centerY, int radius, int okamzitaVychylkaValue)*[1]. Tato metoda použije Bresenhamův algoritmus na vykreslování kružnic a zjistí na jakých pixelech (reálné celočíselné pixely na obrazovce) momentálně kružnice leží.

```
int x = 0;
int y = radius;
int d = 3 - 2 * radius;
//finds the coordinates of pixels
while (y >= x) {
    if (d > 0) {
```

```
            y--;
            d = d + 4 * (x - y) + 10;
        } else {
            d = d + 4 * x + 6;
        }
        // Set the pixels for this circle
        setCirclePixelsDisplacement
        (centerX, centerY, x, y, okamzitaVychylkaValue);
        x++;
    }
```

Hodnoty jsou předány metodě *setCirclePixelsDisplacement(int centerX, int centerY, int x, int y, int okamzitaVychylkaValue)*, která zjistí zda je daná souřadnice v místnosti. Jelikož Bersenhamův algoritmus funguje na bázi kopírování pixelů symetricky podle středu a os symetrií, tak je potřeba zjistit zda se daný pixel nachází v místnosti. To metoda rozpozná za pomoci 8 podmínek (8 protože kruh je symetrický po 8 osách). Zde je příklad několika z těchto podmínek.

```
if (centerY + y > yMin && centerY + y < yMax &&
    centerX + x > xMin && centerX + x < xMax)
if (centerY + y > yMin && centerY + y < yMax &&
    centerX - x > xMin && centerX - x < xMax)
if (centerY - y > yMin && centerY - y < yMax &&
    centerX + x > xMin && centerX + x < xMax)
if (centerY - y > yMin && centerY - y < yMax &&
    centerX - x > xMin && centerX - x < xMax)
```

Pokud se tyto podmínky splní, tak se zavolá metoda setPixelDisplacement(int x, int y, int okamzitaVychylkaValue), jenž přidá okamžitou výchylku kruhu na jednotlivé pixely. Jako první se konvertují souřadnice na scéně na souřadnice v *pixelGridu* pomocí metody *getGridX/Y* ze třídy Pixel. Dále se zkontroluje, zda jsou gridové souřadnice validní a pokud jsou, tak se vytvoří nová instance třídy *PixelCoordinate* *pixelCoord* se souřadnicemi *gridX/Y*. Poté se zkontroluje, zda daný *pixelCoord* nebyl už jednou přidaný, to znamená jestli není v setu *visitedPixels*. Pokud je tak se přidá do setu *duplicatePixels*, a pokud není, tak se přidá do setů *visitedPixels* a *activePixelCoordinates*, a *Pixelu* se stejnými souřadnicemi v *pixelGridu* se přidá *okamzitaVychylka* kružnice.

4.15 WaveFactory

Tato třída má funkci továrny na vlny. Má pouze jednu metodu *createWave(double x, double y, BaseRoomControllerInterface controller, int radius, int amplitude, int direction)*, která vrátí novou instanci třídy *SoundWave*.

4.16 WaveManager

Třída *WaveManager*, jak z názvu vyplývá, funguje jako manažer pro všechny vlny na scéně. Stará se o aktualizaci scény nebo vytváření, odstraňování a odrazy vln. Dále je zodpovědná za zastavování, spouštění a resetování simulace když uživatel klikne na příslušná tlačítka.

4.16.1 Vytváření

K vytvoření individuální vlny se používá metoda *createWave(double x, double y, BaseRoomControllerInterface controller, int radius, int amplitude)*. Metoda nejdříve zkontroluje jeslti je *centerPane* null a pokud není, tak se vytvoří nová vlna pomocí třídy *WaveFactory*. Poté se připíše to *Arraylistu activeWaves* a přiřadí se jí instance třídy *PixelManager*.

4.16.2 Aktualizace scény

Pro aktualizaci se používá metoda *updateWaves(BaseRoomControllerInterface controller)*. Tato metoda nejdříve vytvoří *ArrayList wavesToRemove* a *HashSet activePixelCoordinates*. Tyto datové struktury slouží na zapisování vln, které je potřeba odstranit a ke sledování *Pixelů*, jenž jsou momentálně ve vlně. Dále se spustí for-cyklus iterující přes každou *SoundWave* v *activeWaves*. Pro každou z těchto vln se nejprve zavolá metoda *grow()*, poté se zkontroluje zda je starší než 8 sekund a jestli je její amplituda nižší nebo rovna nule. Pokud se tyto podmínky naplní, tak se vlna přiřadí do *Arraylistu wavesToRemove* . Následně se do *activePixelCoordinates* přidají všechny pixely dané vlny pomocí metody *wave.getactivePixelCoordinates()*.
Potom, co se for-cylkus ukončí, tak se vynulují všechny pixely, které nejsou v žádné vlně, zavolá se metoda *checkWavesForReflections (BaseRoomControllerInterface controller)* (viz. kapitola 4.16.3) a smažou se všechny vlny, jenž jsou ve *wavesToRemove*.

4.16.3 Odraz

Pokaždé když se aktualizuje scéna pomocí metody *updateWaves()*, tak se zavolá metoda checkWavesForReflections(BaseRoomControllerInterface controller).

checkWavesForReflections(BaseRoomControllerInterface controller)

Tato metoda znovu iteruje přes všechny vlny *wave* v *activeWaves*. Nejdříve vytvoří nový *Point center*, který je stejný jako střed vlny. Potom program zkontroluje zda je *center* v místnosti. Pokud je, tak se z vlny získají *reflectionDistances* a spustí se for-cylkus, jenž iteruje přes všechny tyto vzdálenosti. Pokud se *outerRadius* vlny rovná některé z těchto vzdáleností, tak se vytvoří nová přímka reprezentující danou stěnu, následně se pomocí třídy *Calculator* vytvoří symetrický bod, jenž reprezentuje střed nové vlny a nakonec se vytvoří nová vlna s počátečním poloměrem *outerRadius* a amplitudou o 50 menší.

Pokud střed vlny neleží v místnosti, tak se zavolá metoda *handleOutOfRectangleWave(SoundWave wave, Point center, BaseRoomControllerInterface controller)*.

handleOutOfRectangleWave(SoundWave wave, Point center, BaseRoomControllerInterface controller)

Tato metoda zpracovává vlny, jejichž střed neleží v místnosti a funguje následovně. Nejdříve se z dané vlny získají *reflectionDistances*, dále se zavolají metody *handleOutOfRectangleWaveForCorners(wave, center, controller)*

a handleDiagonalWavesForCorners(wave, center, controller), které zpracovávají vzdálenosti ke středům (viz další odstavce), potom se zkontroluje, zda je střed vlny ve správné pozici a má správný poloměr. Zde je příklad těchto podmínek

```
if(wave.isAboveRectangle(center.getX(), center.getY()) &&
currentRadius == reflectionDistances[0]){...}
else if (wave.isBellowRectangle(center.getX(), center.getY()) &&
currentRadius == reflectionDistances[0]){...}
```

Jestliže jsou tyto podmínky splněny, tak se zavolá metoda *reflectWave* (viz další odstavce), která vytvoří novou symetrickou vlnu.

handleOutOfRectangleWaveForCorners(SoundWave wave, Point center, BaseRoomControllerInterface controller)

Tato metoda se specializuje na odraz vln, které mají středy v sektorech 2,4,6,8 (viz. obrázek 5). Nejprve se zjistí, v jakém z těchto sektorů se střed nachází. Poté se zjistí, zda se *outerRadius* rovná vzdálenosti od středu k některému z rohů a pokud ano tak se zavolá metoda *reflectWave*, která vytvoří novou vlnu symetrickou vůči správné zdi.

handleDiagonalWavesForCorners(wave, center, controller)

Tato metoda se stará o odrazy vln, jejichž středy leží v sektorech 1,3,7,9 (viz. obrázek 5). V této metodě se nejprve inicializují rohy místnosti a poloměr *topLeft, bottomLeft, bottomRight, topRight, currentRadius*. Dále se zjistí v jakém sektoru střed leží. Pokud leží v sektoru jedna, tak se počítají vzdálenosti pouze k *bottomLeft, bottomRight, topRight* a analogicky je to i

se středy v ostatních sektorech. Jakmile se lokalizuje sektor, tak program zkontroluje, zda se *outerRadius* rovná vzdálenosti středu k některému z rohů místnosti, jestli ano tak se vytvoří příslušná symetrická vlna pomocí metody *reflectWave*. Zde je příklad jak taková podmínka vypadá.

```
if (wave.isAboveOnTheLeftOfRectangle(center.getX(),
center.getY())) {
    if (currentRadius==
    (int)wave.getCenter().distance(bottomLeft)){
        reflectWave(wave, controller, 1);
    }
    if (wave.getRadius() ==
    (int)wave.getCenter().distance(bottomRight)) {
        Point pomocnyBod =
        calculator.calculateSymetricPoint
        (center,controller.getRoomWalls().get(1));
        reflectWave(pomocnyBod,
        wave.getOuterRadius(), controller, 3, wave.getAmplitude());
    }
    if (currentRadius==
    (int) wave.getCenter().distance(topRight)){
        reflectWave(wave, controller, 3);
    }
```

reflectWave(SoundWave wave, BaseRoomControllerInterface controller, int wallIndex)

Tato metoda se stará o vytváření odražených vln s pomocí *SoundWave, BaseRoomControllerInterface a wallIndex*. Nejprve se inicializuje střed vlny a přímka, jenž reprezentuje stěnu, od které se vlna odráží. Poté se za pomoci třídy *Calculator* vytvoří symetrický bod *symetricPoint* a vytvoří se nová vlna se středem v symetrickém bodě, počátečním poloměrem *outerRadius* a amplitudou o 50 menší. Metoda *reflectWave(Point center, int currentRadius, BaseRoomControllerInterface controller, int wallIndex, int amplituda)* dělá úplně to samé jen za pomoci jiných parametrů.

Třída *WaveManager* má také metody *resumeWaves(), pauseWaves(), resetWaves()*, tyto metody bud zastaví, znovu spustí nebo resetují animaci. Poslední metoda, kterou tato třída obsahuje, je *removeAllWaves(Pane centerPane)*. Ta odstraní všechny vlny ze scény.

5 Závěr

Jako ročníkový projekt jsem si vybral simulování zvuku, protože jsem předpokládal, že to bude velmi zajímavé. Při programování jsem pochopil, jak se pohybuje zvuk v místnosti a také jsem poprvé musel pracovat na efektivitě, aby program dobře fungoval. Zjistil jsem, že JavaFX je hodně špatný framework na tvoření dynamických simulací, protože všechny operace dělá velmi pomalu oproti ostatním programovacím jazykům.

I přes všechna nutná zjednodušení, program splňuje zadání a správně vizuálně reprezentuje, jak se impulzy zvuku pohybují v místnosti. Celkem program zvládne simulovat dvě vlny zároveň, pokud uživatel přidá na scénu více vln, začne se program sekat a ne všechny pixely se budou přepisovat správně.

Ve výsledku mě programování této aplikace velmi bavilo a všechny problémy, na které jsem narazil, se mi podařilo vyřešit s menšími či většími potížemi.

Seznam obrázků

Reference

[1] I. I. A. Assistent. Can you implement setcirclepixelsdisplacement, which modifies or stores pixel values for the circle drawn using bresenham's algorithm? it should apply okamzitavychylkavalue to each pixel appropriately. the method must integrate correctly with drawcirclewithokamzitavychylka and consider whether we are working with a graphical buffer, a 2d array, or another data structure for managing pixels., 2024.

[2] I. I. A. Assistent. Write a method, that will have two parametrs line line1 and line line2 and based on their general equation it will calculate the intersection of those two lines, 2024.

[3] I. I. A. Assistent. Write me a method that will have two parametrs point point and line line. this method will return a point that will be symetric to the parameter point with the line being the axis, 2024.

[4] Wikipedie. Zvuk, 2025.

[5] F. ústav filozofická fakulta Univerzita Karlova. Zvuk, 2018.