Politechnika
Wrocławska

Sprawozdanie Swing Java

Autorzy:
Jakub Gil
Kacper Zomerfeld
Numery indeksów:
263455, 264314

Prowadzący:
dr inż. Aneta Górniak

8 czerwca 2024

Spis treści

1 Wstęp

Gry komputerowe od dawna stanowią ważny element kultury rozrywkowej, zdobywając coraz większą popularność i przyciągając miliony graczy na całym świecie. Jedną z najbardziej ikonicznych i rozpoznawalnych gier w historii jest Pac-Man, która zadebiutowała w 1980 roku i od tamtej pory zachwyca kolejne pokolenia. Niniejsze sprawozdanie ma na celu przedstawienie procesu tworzenia gry Pac-Man w formie aplikacji desktopowej, wykorzystującej język programowania Java. Gra Pac-Man jest klasyczną grą z gatunku arkadowego, w której gracz steruje postacią Pac-Mana, próbując zbierać punkty na planszy, unikając jednocześnie spotkania z duchami. Gra została przez nas lekko uproszczona Pac-mana gonią tylko trzy duchy jak i pozbyliśmy się tunela z gry.

2 Budowa aplikacji

2.1 Game loop

```
double drawInterval = 1_000_000_000 / FPS;
    double nextDrawTime = System.nanoTime() + drawInterval;

    while (gameThread.isAlive()) {
        if (gameState == loseState) {
            break;
        }

        update();
        repaint();

        try {
            double remainingTime = nextDrawTime - System.nanoTime();
            remainingTime /= 1_000_000;
            if (remainingTime < 0) {
                remainingTime = 0;
            }
            Thread.sleep((long) remainingTime);
            nextDrawTime += drawInterval;
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
```

Jest to głowna petlą w grze, która odpowiada za updatowanie i rysowanie rzeczy na ekranie. Poza tym funkcja ta ograniczna liczbe klatek na sekundę do 60, dzięki timerom. Funkcja ta znajduje się w klasie GamePanel która odpowiada za rzeczy rysowane na ekranie. Każda klasa która chce rysować coś na ekranie musi w swoim kontruktorze zawierać klase GamePanel.

2.2 Rysowanie planszy

Mapa przechowywana jest w pliku txt, każda z liczb odpowiada odpowiedniemu plikowi png.

```
10 24 24 24 24 24 24 24 24 24 24 24 24 14 24 24 24 24 24 24 24 24 24 24 24 24 13
20  0  0  0  0  0  0  0  0  0  0  0  0 20  0  0  0  0  0  0  0  0  0  0  0  0 20
20 0 1 5 5 4 0 1 5 5 5 4 0 20 0 1 5 5 5 4 0 1 5 5 4 0 20
20 0 6 9 9 8 0 6 9 9 9 8 0 20 0 6 9 9 9 8 0 6 9 9 8 0 20
20 0 2 7 7 3 0 2 7 7 7 3 0 21 0 2 7 7 7 3 0 2 7 7 3 0 20
20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 20
20 0 1 5 5 4 0 1 4 0 1 5 5 5 5 5 4 0 1 4 0 1 5 5 4 0 20
20 0 2 7 7 3 0 6 8 0 2 7 7 7 7 7 3 0 6 8 0 2 7 7 3 0 20
20 0 0 0 0 0 0 6 8 0 0 0 0 0 0 0 0 0 6 8 0 0 0 0 0 0 20
20 0 1 5 5 4 0 6 8 0 1 5 5 5 5 5 4 0 6 8 0 1 5 5 4 0 20
20 0 6 9 9 8 0 6 8 0 2 7 7 7 7 7 3 0 6 8 0 6 9 9 8 0 20
20 0 2 7 7 3 0 6 8 0 0 0 0 0 0 0 0 0 6 8 0 2 7 7 3 0 20
20 0 0 0 0 0 0 6 8 0 10 24 25 0 22 24 13 0 6 8 0 0 0 0 0 0 20
15 24 24 24 24 25 0 2 3 0 20 0 0 0 0 0 20 0 2 3 0 22 24 24 24 24 17
20 0 0 0 0 0 0 0 0 0 20 0 22 24 25 0 20 0 0 0 0 0 0 0 0 0 20
20 0 1 5 5 4 0 1 4 0 20 0 0 0 0 0 20 0 1 4 0 1 5 5 4 0 20
20 0 6 9 9 8 0 6 8 0 11 24 24 24 24 24 12 0 6 8 0 6 9 9 8 0 20
20 0 6 9 9 8 0 6 8 0 0 0 0 0 0 0 0 0 6 8 0 6 9 9 8 0 20
20 0 6 9 9 8 0 6 8 0 1 5 5 5 5 5 4 0 6 8 0 6 9 9 8 0 20
20 0 2 7 7 3 0 6 8 0 2 7 7 7 7 7 3 0 6 8 0 2 7 7 3 0 20
20 0 0 0 0 0 0 6 8 0 0 0 0 0 0 0 0 0 6 8 0 0 0 0 0 0 20
20 0 1 5 5 4 0 6 8 0 1 5 5 5 5 5 4 0 6 8 0 1 5 5 4 0 20
20 0 2 7 7 3 0 2 3 0 2 7 7 7 7 7 3 0 2 3 0 2 7 7 3 0 20
20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 20
20 0 1 5 5 4 0 1 5 5 5 4 0 19 0 1 5 5 5 4 0 1 5 5 4 0 20
20 0 6 9 9 8 0 6 9 9 9 8 0 20 0 6 9 9 9 8 0 6 9 9 8 0 20
20 0 2 7 7 3 0 2 7 7 7 3 0 20 0 2 7 7 7 3 0 2 7 7 3 0 20
20 0 0 0 0 0 0 0 0 0 0 0 0 20 0 0 0 0 0 0 0 0 0 0 0 0 20
11 24 24 24 24 24 24 24 24 24 24 24 24 16 24 24 24 24 24 24 24 24 24 24 24 24 12
```

Na podstawie tych wartości rysuje się odpowiedni kwadrat na mapie i nastepnie określane są jego własności, przukładowo wszystkie pola poza 0 są kolizyjne co oznacza że nie można przez nie przejść.

```
while(col < gp.maxScreenCol && row < gp.maxScreenRow) {
        int tileNum = mapTileNum[col][row];
        if(tileNum == 0 || tileNum == -1){

        }
        else {
            g2.drawImage(tiles[tileNum].image, x, y, -
gp.tileSize, gp.tileSize, null);
        }
        col++;
        x+=gp.tileSize;
        if(col == gp.maxScreenCol) {
            col = 0;
            x = 0;
            row++;
            y+=gp.tileSize;
        }
    }
```

Głowna częsci funkcji odowiadającej na wczytaniu wartości z pliku, która jest przechowywana w zmiennej zmiennej mapTileNum, a nastepnie narysowaniu odpowiednich pól w odpowiednich miejscach. Analogicznie postepujemy kiedy chemy rysować punkty lub inne stacjonarne obiekty na planszy.

2.3 Gracz oraz duchy

Zarówno gracz jak i duchy są obiektami dynamicznymi którzy co klatkę zmieniają swoją pozycje, z tego powodu wszysktie te kalsy dziedziczą‘z klasy Entity. Każda z tych klas wyglada analogicznie poza równicami pobieranych plikach oraz algorytmach poruszania. Przykładowo gracz kontroluje swoją postać z klawiatury, gdzie duchy poruszają sie prostym AI.// Kontrola ruchem gracza.

```
if (keyHandler.upPressed) {
        direction = "up";
        gp.collisionChecker.checkTile(this);
        if (collisionOn == false) {
            y -= speed;

        }
     else {
         LastDirection = "up";
         keyHandler.lastButtonPressed(directionBeforePotencialColision);
         direction = directionBeforePotencialColision;
        }
    }
```

```
else if(keyHandler.downPressed){
    direction = "down";
    gp.collisionChecker.checkTile(this);
    if (collisionOn == false) {
        y += speed;

    }
    else {
        LastDirection = "down";
        keyHandler.lastButtonPressed(directionBeforePotencialColision);
        direction = directionBeforePotencialColision;
    }
```

Algorytm sterowania AI w pierwszej kolejnosci wyznacza punkt do którego dąży jest to albo aktualna pozycja Pac-Mana albo ta którąosiągnie za jakiś czas. Na podstawie tego kolejkowane są ruchy które nastepnie są sprawdzane czy nie ma kolizji w ich strone jeżeli taka by sie wydarzyła to ten ruch nie jest brany pod uwagę i jest brany następny najleszy. Poniżej przedstawiono już samo zmienianie kierunków kiedy już zostanie wybrany odpowiedni kierunek.

```
if (!collisionOn) {
     switch (direction) {
         case "up":
         y -= speed;
             break;
         case "down":
             y += speed;
             brek;
         case "left":
             x -= speed;
             break;
         case "right":
         x += speed;
             break;
```

2.4 Detekcja kolizji

Detekacja kolizji opowiązuje tylko obiekty które poruszają się po planysz czytaj duchy, Pac-Man. Ze wzgldy na to tylko dla nich sprawdzamy kolizje. Zaimplementowane są 3 algorytmy kolizji.

1. Ściany
2. Punkty

3. NPC

Ze wzgledy na dużą ilosc ścian, zaimplemetnowany został specjalny algorytm który sprawdza kwadraty w okół obietków typu Entity czy się z nimi stykają, w przypadku gdybyśmy sprawdzali po kolei wszystkie obiekty robilibysmy dużo zbędnych obliczeń przez co nasza gra mogła by słabiej działać.

```
int entityLeftWordX = e.x + e.bounds.x;
    int entityRightWordX = e.x + e.bounds.x + e.bounds.width;
    int entityTopWordY = e.y + e.bounds.y;
    int entityBottomWordY = e.y + e.bounds.y + e.bounds.height;
    int entityLeftCol = entityLeftWordX/gp.tileSize;
    int entityRightCol = entityRightWordX/gp.tileSize;
    int entityTopRow = entityTopWordY/gp.tileSize;
    int entityBottomRow = entityBottomWordY/gp.tileSize;
    int tileNum1, tileNum2;
    switch (e.direction) {
        case "up":
            entityTopRow = (entityTopWordY - e.speed) / gp.tileSize;
            tileNum1 = gp.tileManager.mapTileNum[entityLeftCol][entityTopRow];
            tileNum2 = gp.tileManager.mapTileNum[entityRightCol][entityTopRow];
            if(gp.tileManager.tiles[tileNum1].collision
            gp.tileManager.tiles[tileNum2].collision) {
                e.collisionOn = true;
            }
            else{
                e.collisionOn = false;
            }
            break;
        case "down":
            entityBottomRow = (entityBottomWordY + e.speed) / gp.tileSize;
            tileNum1 = gp.tileManager.mapTileNum[entityLeftCol][entityBottomRow];
            tileNum2 = gp.tileManager.mapTileNum[entityRightCol][entityBottomRow]
            if(gp.tileManager.tiles[tileNum1].collision gp.tileManager.tiles[tile
                e.collisionOn = true;
            }
            else{
                e.collisionOn = false;
            }
            break;
        case "left":
            entityLeftCol = (entityLeftWordX - e.speed) / gp.tileSize;
            tileNum1 = gp.tileManager.mapTileNum[entityLeftCol][entityTopRow];
            tileNum2 = gp.tileManager.mapTileNum[entityLeftCol][entityBottomRow];
            if(gp.tileManager.tiles[tileNum1].collision
```

```
                gp.tileManager.tiles[tileNum2]
                .collision) {
                    e.collisionOn = true;
                }
                else{
                    e.collisionOn = false;
                }
                break;
            case "right":
                entityRightCol = (entityRightWordX + e.speed) / gp.tileSize;
                tileNum1 = gp.tileManager.mapTileNum[entityRightCol][entityTopRow];
                tileNum2 = gp.tileManager.mapTileNum[entityRightCol][entityBottomRow]
                if(gp.tileManager.tiles[tileNum1].collision gp.tileManager.tiles[tile
                    e.collisionOn = true;
                }
                else{
                    e.collisionOn = false;
                }
                break;
        }
```

Ze wzgledu na to że wraz z długosćia gry liczba pubktów do zdobycia się zmniejsza to tutaj nie został zaimplementowany żaden specjalny algorytm, za każdym razem algorytm sprawdza każdy obiekt na mapie ,analogicznie dla kolizcji NPC z graczem.

```
    for(int i=0; i < gp.heart.length; i++){
            if(gp.heart[i] != null){
                //System.out.println(e.bounds.intersects(gp.obj[i].bounds));
                if(e.bounds.intersects(gp.heart[i].bounds)){
                    // System.out.println(gp.obj[i].bounds.x + " " + gp.obj[i].bounds
                    if(gp.heart[i].collision){
                        e.collisionOn = false;
                    }
                    if(player) {
                        index = i;
                    }
                }
            }
        }
```

2.5 Wielowątkowość

Każdy obiekt poruszający sie na mapie wykonuje swoje obliczenia w odzielnych wątkach. W przypadku Pac-Mana nie widać dużych różnic ze wzgledu na to że są to proste obliczenia. Jednakże gdyby gra była by bardziej wymagająca robiło by to sporą różnicę.

Każdy wątek wykonuję funkcje update dla poszczegolnych Entity dla poszczególnego ruchu. Kiedy obliczenia dochodzą do końca to dodaje je do listy. Nastepnie czekamy aż lista wypełni sie do rozmiaru takiego ile jest obiektów które porusza się po mapie. Kiedy to nastąpi nastepuje ruch obiektów na planszy. Nastpenie stworzone są nowe wątki do obliczenia kolejnego położenia obietktów.

```
public synchronized void addPosition(Point position) {
    if (positionQueue.size() < 4) {
        positionQueue.add(position);
    }

    if (positionQueue.size() == 4) {
        allMoved = true;
        notify();
    }
}
\\ uruchomienie wątków.
public void update() {
    if (gameState == playState) {
        new Thread(player::update).start();
        for (int i = 0; i < npc.length; i++) {
            if (npc[i] != null) {
                new Thread(npc[i]::update).start();
            }
        }
    }
}
```

2.6 Diagram klas


KeyHandler
gp GamePanel
leftPressed boolean
upPressed boolean
downPressed boolean
rightPressed boolean
Tile
image BufferedImage
collision boolean
Screens
message String
arial_80B Font
gameOver boolean
messageCounter int
gp GamePanel
messageOn boolean
arial_40 Font
g2 Graphics2D
superObject
wordX int
bounds Rectangle
wordY int
image BufferedImage
name String
collision boolean
Main
AssetSetter
tm TileManager
gp GamePanel
player Player
TileManager
gp GamePanel
mapTileNum int[][]
tiles Tile[]
Heart
Entity
playerDirection String
x int
speed int
boundsY int
gp GamePanel
possibleDirections LinkedList<String>
LastDirection String
up BufferedImage
bounds Rectangle
startTimerForPanicMode boolean
down BufferedImage
boundX int
left BufferedImage
y int
panicMode boolean
PANIC_MODE_DURATION int
right BufferedImage
character BufferedImage
dead boolean
panicModeTimer Timer
collisionOn boolean
direction String
Point
GamePanel
npc Entity[]
collisionChecker CollisionChecker
screens Screens
loseState int
FPS double
screenWidth int
keyHandler KeyHandler
playState int
gameState int
obj superObject[]
pauseState int
tileManager TileManager
tileSize int
positionQueue LinkedList<Point>
allMoved boolean
maxScreenRow int
gameThread Thread
maxScreenCol int
endState int
player Player
heart superObject[]
assetSetter AssetSetter
scale int
originalTileSize int
screenHeight int
CollisionChecker
gp GamePanel
Player
points int
keyHandler KeyHandler
Pinky
scatterTarget Point
player Player
Clyde
mode String
player Player
scatterTarget Point
modeSwitchTimer long
modeSwitchInterval long
Blinky
player Player
mode String
modeSwitchTimer long
modeSwitchInterval long
scatterTarget Point