Сучасні перспективи розвитку науки:
матеріали II Міжнар.
наук.-практ. конф. (Київ,
8-9 вересня 2018 р.). - К. :
МЦНД, 2018. – С. 46-48.

Сухицький А.М.
студент 2 курсу магістратури, спеціальності
«Інженерія програмного забезпечення»
Луганського національного університету
імені Тараса Шевченка
artursuhitskiy@gmail.com

Переяславська С.О.
к. пед. н., доцент кафедри інформаційних технологій
та систем Луганського національного університету
імені Тараса Шевченка
pereyaslav9@gmail.com

Підходи до реалізації спрайтової анімації на різних платформах Java

В статті аналізуються методи створення спрайтової 2D анімації на різних платформах Java (Java SE, JavaFX, фреймворк Java LibGDX). Для більшої наочності наводяться приклади кодів, що демонструють можливості цих платформ при реалізації спрайтової анімації.

Ключові слова: спрайтова, анімація, спрайт, JavaFX, фреймворк LibGDX, Java SE.

У наш час комп’ютерна анімація застосовується у різноманітних комп’ютерних додатках для створення рухомих ефектів 2D або 3D зображень і

підвищення візуальної комунікації з програмою. Тому питання вибору способів реалізації комп'ютерної анімації є досить актуальним.

Мета дослідження полягає у аналізі методів створення спрайтової 2D анімації на різних платформах Java (Java SE, JavaFX, фреймворк Java LibGDX).

В комп’ютерній 2D анімації зміна параметрів двовимірних зображень у часі відбувається у наступні способи: паралельне переміщення на площині, обертання, зміна кольору та форми об’єкту тощо, створення ілюзії руху з використанням статичних зображень (спрайтів). Розглянемо спрайтову анімацію.

В основі спрайтової анімації є поняття спрайта, – двовимірного зображення, яке використовується об’єктами для їх же візуалізації. Кожен об'єкт пов'язаний з асоційованим до себе спрайтом. Це може бути зображення одного об’єкту, або зображення, що складається з багатьох частин (кадрів), які відображають певні елементи руху об’єкту. Ці кадри програма послідовно візуалізує для отримання ефекту руху (анімації) (рис. 1).

Рис.1. Приклад спрайта з багатьма кадрами

Розглянемо методи створення спрайтової анімації у середовищах Java SE, JavaFX та фреймворку Java LibGDX.

Алгоритм створення спрайтової анімації в зазначених платформах однаковий: завантаження текстури, опис процесу анімації, запуск анімації. Однак ці алгоритми реалізуються різними класами та методами, що притаманні цим середовищам.

Опрацювання зображень в JavaFX здійснюється за допомогою двох класів: Image і ImageView. Зокрема, клас Image інкапсулює саме зображення, а клас ImageView керує його візуалізацією, наприклад:

private final Image IMAGE = new

Image(getClass().getResourceAsStream("player.png"));

final ImageView imageView = new ImageView(IMAGE);

Безпосередньо анімація у JavaFX описується завдяки класу Animation та його нащадку Transition [2]. У класі створюється конструктор, який буде отримувати дані про текстуру. Цей клас має метод interpolate (double k), у якому описуються правила зміни кадрів анімації:

//обираємо індекс стовбця матриці спрайта

*final int index = Math.min((int) Math.floor(k * count), count - 1);*

// визначаємо координати кадру з урахуванням його параметрів

*final int x = (index % columns) * width + offsetX;*

*final int y = (index / columns) * height + offsetY;*

//створюємо прямокутник кадра з певними координатами

imageView.setViewport(new Rectangle2D(x, y, width, height));

Реалізується анімація у JavaFX за допомогою методів класу Animation, що задають циклічність, наприклад:

animation.setCycleCount(Animation.INDEFINITE);

LibGDX адаптований до роботи зі спрайтовою анімацією завдяки специфічним класам фреймворку. Аналогічні дії виконуються за допомогою класу Texture, який отримує зображення спрайта з файлу і завантажує його в GPU, та класу SpriteBatch, який відповідає за його відображення. Анімація описується завдяки класу TextureRegion та його методу TextureRegion.split(). На цьому етапі відбувається формування двомірного масиву кадрів з текстури.

Треба звернути увагу на те, що анімація відбувається тільки з використанням одновимірних структур даних [1]. У наступному коді наведено приклад перетворення двомірного масиву кадрів спрайта в одномірний масив. Далі масив додається до об'єкту Animation з параметром швидкості зміни кадрів.

TextureRegion[][] tmp = TextureRegion.split(spriteSheet,

spriteSheet.getWidth()/FRAME_COLS,

spriteSheet.getHeight()/FRAME_ROWS);

```
spriteFrames = new TextureRegion[FRAME_COLS * FRAME_ROWS];
int index = 0;
for (int i = 0; i < FRAME_ROWS; i++){
        for(int j = 0; j < FRAME_COLS; j++){
                spriteFrames[index++] = tmp[i][j];  }
}
spriteAnimation = new Animation(0.025f, spriteFrames);
```

У прикладі, який наведено нижче, змінній currentFrame надається створений масив кадрів, кількість часу, який пройшов з останньої візуалізації (змінна stateTime) та логічна змінна, яка відповідає за циклічність анімації:

```
stateTime += Gdx.graphics.getDeltaTime();
currentFrame = (TextureRegion) spriteAnimation.getKeyFrame(stateTime, true);
```

У фреймворку LibGDX анімація реалізується у методі render(), де за допомогою методу draw(), що належить класу SpriteBatch, відбувається візуалізація кадру. У метод передаються поточний кадр та його координати:

```
spriteBatch.draw(currentFrame, x,y);
```

На відміну від вже розглянутих засобів розробки, Java SE не має специфічних класів для роботи зі спрайтовою анімацією. У даному середовищі анімацію можна створити, використовуючи цикл або таймер.

Розглянемо підходи до створення анімації за допомогою циклів. Для зберігання посилань на кадри використовуються змінні класів ArrayList та String, що зберігатимуть посилання на поточний кадр та лічильник кадрів для їх відображення:

```
Private inti_anim = 0;
ArrayList<String> list = new ArrayList<String>();
private String s;
list.add("src/sp_0");
list.add("src/sp_1");
list.add("src/sp_...");
```

list.add(“src/sp_n”);

Для перебору кадрів спрайта використовується цикл:

i_anim++;

if (i_anim>=l_list.size()) i_anim = 0;

s = l_list.get(i_anim);

Для відображення та реалізації анімації використовується клас Graphics2D, що розширює клас Graphics, щоб забезпечити більш витончений контроль над геометрією, координатами перетворень, керуванням кольорами тощо. Крім того застосовуємо метод класу ImageIcon:

Graphics2D g;

g.drawImage(new ImageIcon(s).getImage(), (int)x, (int)y, null);

При застосуванні таймера для зберігання кадрів використовуються такі самі класи та змінні, що й для циклу. Для перебору кадрів в цьому випадку використовуємо клас Timer:

```
Timer timer = new Timer(25, new ActionListener() {
public void actionPerformed(ActionEvent e){
i_anim++;
s = list.get(i_anim);
if(i_anim>= list.size()-1){
i_anim = 0;    }
    }
});
```

Для запуску і зупинки анімації застосовуємо методи *timer.start()* та *timer.stop()* відповідно. Візуалізація на екрані анімації здійснюється таким же чином, як і при застосуванні циклів анімації.

*Висновки***.** Проведений аналіз довів, що реалізація спрайтової анімації більш легка з точки зору розробки коду на платформах JavaFX та Java фреймворк LibGDX, бо в них вже реалізовані спеціальні класи для роботи з цим типом анімації. У Java SE цей процес має більш складний характер у порівнянні з попередніми середовищами завдяки відсутності специфічних класів. Однак

даний недолік перекривається гнучкістю платформи. Користувач може сам створити класи для роботи з анімацією, які необхідні йому у різних випадках.

Література:

1. 2D Анимация [Електронний ресурс] – Режим доступу до ресурсу: http://www.libgdx.ru/2014/09/2D-Animation.html
2. Creating a Sprite Animation with JavaFX [Електронний ресурс] – Режим доступу до ресурсу: https://netopyr.com/2012/03/09/creating-a-sprite-animation-with-javafx/