SFML

Animated Sprite Class

Animation.cpp

```cpp
#ifndef ANIMATION_INCLUDE
#define ANIMATION_INCLUDE

#include <vector>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/Texture.hpp>

class Animation
{
public:
    Animation();

    void addFrame(sf::IntRect rect);
    void setSpriteSheet(const sf::Texture& texture);
    const sf::Texture* getSpriteSheet() const;
    std::size_t getSize() const;
    const sf::IntRect& getFrame(std::size_t n) const;

private:
    std::vector<sf::IntRect> m_frames;
    const sf::Texture* m_texture;
};

#endif // ANIMATION_INCLUDE
```

Animation.hpp

```cpp
#include "Animation.hpp"

Animation::Animation() : m_texture(NULL)
{

}

void Animation::addFrame(sf::IntRect rect)
{
    m_frames.push_back(rect);
}

void Animation::setSpriteSheet(const sf::Texture& texture)
{
    m_texture = &texture;
}

const sf::Texture* Animation::getSpriteSheet() const
{
    return m_texture;
}

std::size_t Animation::getSize() const
```

```
{
    return m_frames.size();
}

const sf::IntRect& Animation::getFrame(std::size_t n) const
{
    return m_frames[n];
}
```

AnimationSprite.hpp

```
#ifndef ANIMATEDSPRITE_INCLUDE
#define ANIMATEDSPRITE_INCLUDE

#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/System/Time.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Transformable.hpp>
#include <SFML/System/Vector2.hpp>

#include "Animation.hpp"

class AnimatedSprite : public sf::Drawable, public sf::Transformable
{
public:
    explicit AnimatedSprite(sf::Time frameTime = sf::seconds(0.2f), bool paused = false, bool looped = true

    void update(sf::Time deltaTime);
    void setAnimation(const Animation& animation);
    void setFrameTime(sf::Time time);
    void play();
    void play(const Animation& animation);
    void pause();
    void stop();
    void setLooped(bool looped);
    void setColor(const sf::Color& color);
    const Animation* getAnimation() const;
    sf::FloatRect getLocalBounds() const;
    sf::FloatRect getGlobalBounds() const;
    bool isLooped() const;
    bool isPlaying() const;
    sf::Time getFrameTime() const;
    void setFrame(std::size_t newFrame, bool resetTime = true);

private:
    const Animation* m_animation;
    sf::Time m_frameTime;
    sf::Time m_currentTime;
    std::size_t m_currentFrame;
    bool m_isPaused;
    bool m_isLooped;
    const sf::Texture* m_texture;
    sf::Vertex m_vertices[4];

    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;

};

#endif // ANIMATEDSPRITE_INCLUDE
```

AnimationSprite.hpp

```
#include "AnimatedSprite.hpp"
```

```cpp
AnimatedSprite::AnimatedSprite(sf::Time frameTime, bool paused, bool looped) :
    m_animation(NULL), m_frameTime(frameTime), m_currentFrame(0), m_isPaused(paused), m_isLooped(looped), m
{

}

void AnimatedSprite::setAnimation(const Animation& animation)
{
    m_animation = &animation;
    m_texture = m_animation->getSpriteSheet();
    m_currentFrame = 0;
    setFrame(m_currentFrame);
}

void AnimatedSprite::setFrameTime(sf::Time time)
{
    m_frameTime = time;
}

void AnimatedSprite::play()
{
    m_isPaused = false;
}

void AnimatedSprite::play(const Animation& animation)
{
    if (getAnimation() != &animation)
        setAnimation(animation);
    play();
}

void AnimatedSprite::pause()
{
    m_isPaused = true;
}

void AnimatedSprite::stop()
{
    m_isPaused = true;
    m_currentFrame = 0;
    setFrame(m_currentFrame);
}

void AnimatedSprite::setLooped(bool looped)
{
    m_isLooped = looped;
}

void AnimatedSprite::setColor(const sf::Color& color)
{
    // Update the vertices' color
    m_vertices[0].color = color;
    m_vertices[1].color = color;
    m_vertices[2].color = color;
    m_vertices[3].color = color;
}

const Animation* AnimatedSprite::getAnimation() const
{
    return m_animation;
}

sf::FloatRect AnimatedSprite::getLocalBounds() const
{
    sf::IntRect rect = m_animation->getFrame(m_currentFrame);

    float width = static_cast<float>(std::abs(rect.width));
    float height = static_cast<float>(std::abs(rect.height));

    return sf::FloatRect(0.f, 0.f, width, height);
}

sf::FloatRect AnimatedSprite::getGlobalBounds() const
{
    return getTransform().transformRect(getLocalBounds());
}

bool AnimatedSprite::isLooped() const
{
    return m_isLooped;
}

bool AnimatedSprite::isPlaying() const
{
    return !m_isPaused;
}

sf::Time AnimatedSprite::getFrameTime() const
{
    return m_frameTime;
}

void AnimatedSprite::setFrame(std::size_t newFrame, bool resetTime)
{
    if (m_animation)
    {
        //calculate new vertex positions and texture coordiantes
        sf::IntRect rect = m_animation->getFrame(newFrame);

        m_vertices[0].position = sf::Vector2f(0.f, 0.f);
        m_vertices[1].position = sf::Vector2f(0.f, static_cast<float>(rect.height));
        m_vertices[2].position = sf::Vector2f(static_cast<float>(rect.width), static_cast<float>(rect.heigh
        m_vertices[3].position = sf::Vector2f(static_cast<float>(rect.width), 0.f);

        float left = static_cast<float>(rect.left) + 0.0001f;
        float right = left + static_cast<float>(rect.width);
        float top = static_cast<float>(rect.top);
        float bottom = top + static_cast<float>(rect.height);

        m_vertices[0].texCoords = sf::Vector2f(left, top);
        m_vertices[1].texCoords = sf::Vector2f(left, bottom);
        m_vertices[2].texCoords = sf::Vector2f(right, bottom);
        m_vertices[3].texCoords = sf::Vector2f(right, top);
    }

    if (resetTime)
        m_currentTime = sf::Time::Zero;
}

void AnimatedSprite::update(sf::Time deltaTime)
{
    // if not paused and we have a valid animation
    if (!m_isPaused && m_animation)
    {
        // add delta time
        m_currentTime += deltaTime;

        // if current time is bigger then the frame time advance one frame
        if (m_currentTime >= m_frameTime)
        {
            // reset time, but keep the remainder
            m_currentTime = sf::microseconds(m_currentTime.asMicroseconds() % m_frameTime.asMicroseconds())

            // get next Frame index
            if (m_currentFrame + 1 < m_animation->getSize())
                m_currentFrame++;
            else
            {
                // animation has ended
                if (!m_isLooped)
                {
                    m_isPaused = true;
                }
                else
                {
                    m_currentFrame = 0; // reset to start
                }



            }

            // set the current frame, not reseting the time
            setFrame(m_currentFrame, false);
        }
    }
}

void AnimatedSprite::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
    if (m_animation && m_texture)
    {
        states.transform *= getTransform();
        states.texture = m_texture;
        target.draw(m_vertices, 4, sf::Quads, states);
    }
}
```