```
////////////////////////////////////////////////////////////
//
// Copyright (C) 2014 Maximilian Wagenbach (aka. Foaly) (foaly.f@web.de)
//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment
// in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such,
// and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution.
//
////////////////////////////////////////////////////////////

#include "stdafx.h"


#include "AnimatedSprite.hpp"

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_texture(NULL)
{

}

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).first;

    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).first;
        m_frameTime = m_animation->getFrame(newFrame).second;

        /*if (m_frameTime <= sf::seconds(0.f))
        {
            return;
        }*/

            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.
    height));
            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)
        {
            /*if (m_frameTime <= sf::seconds(0.f))
            {
                return;
            }*/

            // reset time, but keep the remainder
            //m_currentTime = sf::microseconds(m_currentTime.asMicroseconds() % m_frameTime.asMicroseconds())↙
    ;
            m_currentTime = sf::seconds(0.f);

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

                if (!m_isLooped)
                {
                    //CHANGED by IE, making the animation stop at last frame if loop is false
                    m_currentFrame = m_animation->getSize() - 1; // reset to start
                    m_isPaused = true;
                }

            }

            // 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);
    }
}

const int AnimatedSprite::getCurrentFrame()const
{
    return m_currentFrame;
}
```