Simon Butson and Samson DeVol
Period 3
March 16, 2017

Tutorial for calculating velocity of particles

Introduction

In this tutorial we will explain how to calculate the velocity vector for a particle simulation. We will be using a particle class which can be found below. The particles will also be accelerating towards a point in a particle system. For this program we will be using SFML for code blocks which can be downloaded below.

Set up

We are using C++ in code blocks and SFML Version 2.1 for visuals. You can download the most recent version of SFML from http://www.sfml-dev.org/download/sfml/2.4.2/. If you need assistance downloading SFML Version 2.1 then this video will help you: https://www.youtube.com/watch?v=vt0CiMGzBo8.

This tutorial assumes that the user is using the SFML particle class template available here https://www.sfml-dev.org/tutorials/2.0/graphics-vertex-array.php (scroll down to particle system example). Also make sure to copy the provided demo so you can test the particle class. You will need to include these directories for the class to function properly <iostream>, <SFML/Graphics.hpp>, <cmath>, <stdio.h>, <stdlib.h>, and <time.h>.

Step by step :

To begin your Particle Class Template should look something like this:

```
class ParticleSystem : public sf::Drawable, public sf::Transformable
{
public:

    ParticleSystem(unsigned int count) :
    m_particles(count),
    m_vertices(sf::Points, count),
    m_lifetime(sf::seconds(1)),
    m_emitter(0, 0)
    {
    }

    void setEmitter(sf::Vector2f position)
    {
        m_emitter = position;
    }

    void update(sf::Time elapsed)
    {
        for (std::size_t i = 0; i < m_particles.size(); ++i)
        {
            // update the particle lifetime
            Particle& p = m_particles[i];
            p.lifetime -= elapsed;

            // if the particle is dead, respawn it
            if (p.lifetime <= sf::Time::Zero)
                resetParticle(i);

            // update the position of the corresponding vertex
            m_vertices[i].position += p.velocity * elapsed.asSeconds();
```

```
            // update the alpha (transparency) of the particle according to its lifetime
            float ratio = p.lifetime.asSeconds() / m_lifetime.asSeconds();
            m_vertices[i].color = (sf::Color(255,255,255));
            m_vertices[i].color.a = static_cast<sf::Uint8>(ratio * 255);
        }
    }

private:

    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
        // apply the transform
        states.transform *= getTransform();

        // our particles don't use a texture
        states.texture = NULL;

        // draw the vertex array
        target.draw(m_vertices, states);
    }

private:

    struct Particle
    {
        sf::Vector2f velocity;
        sf::Time lifetime;
    };

    void resetParticle(std::size_t index)
    {
        // give a random velocity and lifetime to the particle
        float angle = (std::rand() % 360) * 3.14f / 180.f;
        float speed = (std::rand() % 50) + 100.f;
        m_particles[index].velocity = sf::Vector2f(std::cos(angle) * speed, std::sin(angle) * speed);
        m_particles[index].lifetime = sf::milliseconds((std::rand() % 2000) + 1000);

        // reset the position of the corresponding vertex
        m_vertices[index].position = m_emitter;
    }
```

```
    std::vector<Particle> m_particles;
    sf::VertexArray m_vertices;
    sf::Time m_lifetime;
    sf::Vector2f m_emitter;
};

int main()
{
    // create the window
    sf::RenderWindow window(sf::VideoMode(1024, 512), "Particles");

    // create the particle system
    ParticleSystem particles(40000);

    // create a clock to track the elapsed time
    sf::Clock clock;
    srand(time(NULL));

    // run the main loop
    while (window.isOpen())
    {
        // handle events
        sf::Event event;
        while (window.pollEvent(event))
        {
            if(event.type == sf::Event::Closed)
                window.close();
        }

        // make the particle system emitter follow the mouse
        sf::Vector2i mouse = sf::Mouse::getPosition(window);
        if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
        {
        particles.setEmitter(window.mapPixelToCoords(mouse));
        }
        // update it
        sf::Time elapsed = clock.restart();
        particles.update(elapsed);
        // draw it
        window.clear();
        window.draw(particles);
        window.display();
    }
    return 0;
}
```

We will not be modifying most of this code, so we will explain the lines that will be changed or added. Most of the functions are applied to each individual particle, and will be ran for every particle in the system before the screen is drawn.

You will want to create two new Vector2f objects called acceleration and attractor. Acceleration will modify the velocity vector of the particles, and attractor will contain x and y coordinates for point to which the particles will be accelerated. They will be placed at the end of your class.

```
    std::vector<Particle> m_particles;
    sf::VertexArray m_vertices;
    sf::Time m_lifetime;
    sf::Vector2f m_emitter;
    sf::Vector2f attractor;
    sf::Vector2f acceleration;


};
```

The end of the class should now look like the above image.

Next you will add a new function allowing you to change the position of the attractor with your mouse. This should be near the top of the class below the public declarations.

```
void setAttractor(sf::Vector2f position)
{
    attractor = position;
}
```

To be able to change the position of where the particles are being emitted and where the attractor is you will want to include this code in the while (window.isOpen()) loop in your int main().

```
sf::Vector2i mouse = sf::Mouse::getPosition(window);
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
{
    particles.setEmitter(window.mapPixelToCoords(mouse));

}
if (sf::Mouse::isButtonPressed(sf::Mouse::Right))
{
    particles.setAttractor(window.mapPixelToCoords(mouse));
}
```

The top line will set a vector with the positions of the mouse on screen. The next two if statements will allow you to set the position of the emitter and attractor to the mouse by pressing the left and right mouse buttons respectively.

Now you will go into void update(sf::Time Elapsed) and add this code to the end. The if statement will determine if the right mouse button is being clicked, and if it is it will calculate acceleration towards the attractor. The program is designed such that acceleration will only occur while the right button is being pressed down.

The angle function will take the inverse tangent of the attractor's y and x position minus the particle's y and x positions to find the angle of the acceleration. Once this is calculated, the acceleration will be 5 times the cos and sin of the angle. The acceleration will be added to particle's velocity to give the new velocity vector. The acceleration is plus equal because we want to retain the previous velocity, but just add a new acceleration on to it.

```
 if (sf::Mouse::isButtonPressed(sf::Mouse::Right))
{
angle = atan2(attractor.y-m_vertices[i].position.y,attractor.x-m_vertices[i].position.x);
acceleration = sf::Vector2f(5*std::cos(angle), 5*std::sin(angle));
p.velocity += acceleration;

}
```

The final thing will be to change the update() function so that the particles will not disappear unless the left button is pressed to set a new emitter. The particle class gives particles a finite lifetime, but the program will look much nicer if the particles do not disappear until the left button is pressed.

You will replace the line that says if (p.lifetime <= sf::Time::Zero) with this code here

```
// if the particle is dead, respawn it
    if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
{
    resetParticle(i);

}
```

This will only run the resetParticle(i) function if the left mouse button is pressed.

Now you should be able to run your program. Pressing the left mouse button will allow you to set the position of the emitter, and the right button will set the position of the attractor. Your particle system should now have acceleration towards a point.