Lazy Foo' Productions

Motion

Last Updated 11/15/07

Now it's moment you've been waiting for, it's time to move a dot around on the screen. It can be more difficult than it sounds. It requires a good understanding of key events. Here you're going to get walked through the basic concepts of moving an object around on the screen.

```
//The dot that will move around on the screen
class Dot
{
    private:
    //The X and Y offsets of the dot
    int x, y;

    //The velocity of the dot
    int xVel, yVel;

    public:
    //Initializes the variables
    Dot();

    //Takes key presses and adjusts the dot's velocity
    void handle_input();

    //Moves the dot
    void move();

    //Shows the dot on the screen
    void show();
};
```

Here we have the rundown of our dot class.

"x" and "y" are the offsets of the dot. "xVel" and "yVel" are the dot's velocity or rate of movement.

The contructor initializes the dot's variables, handle_input() handles events, move() moves the dot, and show() shows it on the screen.

```
Dot::Dot()
{
    //Initialize the offsets
    x = 0;
    y = 0;

    //Initialize the velocity
    xVel = 0;
    yVel = 0;
}
```

As you see all the constructor does is put the dot to the upper left corner and makes sure the dot is still when a Dot object is made.

```
void Dot::handle_input()
{
    //If a key was pressed
    if( event.type == SDL_KEYDOWN )
    {
        //Adjust the velocity
        switch( event.key.keysym.sym )
        {
            case SDLK_UP: yVel -= DOT_HEIGHT / 2; break;
            case SDLK_DOWN: yVel += DOT_HEIGHT / 2; break;
            case SDLK_LEFT: xVel -= DOT_WIDTH / 2; break;
            case SDLK_RIGHT: xVel += DOT_WIDTH / 2; break;
        }
    }
```

Here's our event handler for the dot.

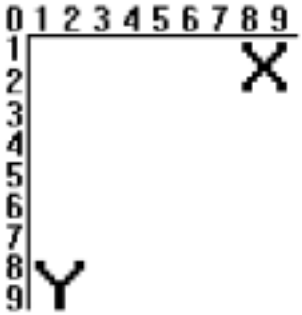
Now you may be thinking that there's no need for all this. That all you need to do is x++/-- or y++/-- when there's a keydown event.

The problem with that is the dot will only move when there's a keydown event. That means you'll have to press the key, release the key, then press it again so it will continue moving.

So what we do instead is set the dot's velocity. The dot has two velocities, the rate it moves along the X axis and the rate it moves along the Y axis. When the right arrow is pressed, we increase the X velocity by half the dot's width (which is 10) so it increases its X offset by 10 every frame. When the left arrow is pressed, we decrease the X velocity by 10 so it decreases its X offset by 10 every frame. The same principle is applied to the Y offset.

Remember that the Y axis doesn't work like this:

It works like this:



So increasing the Y offset causes the dot to go down, and decreasing it causes it to go up.

```
    //If a key was released
    else if( event.type == SDL_KEYUP )
    {
        //Adjust the velocity
        switch( event.key.keysym.sym )
        {
            case SDLK_UP: yVel += DOT_HEIGHT / 2; break;
            case SDLK_DOWN: yVel -= DOT_HEIGHT / 2; break;
            case SDLK_LEFT: xVel += DOT_WIDTH / 2; break;
            case SDLK_RIGHT: xVel -= DOT_WIDTH / 2; break;
        }
    }
}
```

Now we also have to handle when a key is released. When a key is released, a SDL_KEYUP event will happen.

Now when you release a key, you basically undo the change in velocity when you pressed it. When you pressed right, you increased the x velocity by 10 so when you release right you decrease it by 10.

```
void Dot::move()
{
    //Move the dot left or right
    x += xVel;

    //If the dot went too far to the left or right
    if( ( x < 0 ) || ( x + DOT_WIDTH > SCREEN_WIDTH ) )
    {
        //move back
        x -= xVel;
    }

    //Move the dot up or down
    y += yVel;

    //If the dot went too far up or down
    if( ( y < 0 ) || ( y + DOT_HEIGHT > SCREEN_HEIGHT ) )
    {
        //move back
        y -= yVel;
    }
}
```

Now it's time to move the dot.

First we move the dot by adding its velocity to its offset, then we check if the dot went off the screen. If it did we undo the movement by subtracting its velocity from its offset. We do this so the dot doesn't go off the screen.

I'll admit this is a pretty sloppy way to do this because you might get something like this:

where the dot seems to get stuck if you set the velocity to something that's not divisible by the screen's dimensions. That's because the dot isn't moving to the wall, it's only moving to it's position before it went off the screen.

A better way is to set the dot's offset to the screen's dimension minus the dot's dimension when you go off the screen, but I can't spoon feed you everything. That and I'm too lazy to go back and change the code.

```
void Dot::show()
{
    //Show the dot
    apply_surface( x, y, dot, screen );
}
```

Here in the show() function we blit the dot surface on the screen.

```
    //While the user hasn't quit
    while( quit == false )
    {
        //Start the frame timer
        fps.start();

        //While there's events to handle
        while( SDL_PollEvent( &event ) )
        {
            //Handle events for the dot
            myDot.handle_input();

            //If the user has Xed out the window
            if( event.type == SDL_QUIT )
            {
                //Quit the program
                quit = true;
            }
        }

        //Move the dot
        myDot.move();

        //Fill the screen white
        SDL_FillRect( screen, &screen->clip_rect, SDL_MapRGB( screen->format, 0xFF, 0xFF, 0xFF )

        //Show the dot on the screen
        myDot.show();

        //Update the screen
        if( SDL_Flip( screen ) == -1 )
        {
            return 1;
        }
```

```
        //Cap the frame rate
        if( fps.get_ticks() < 1000 / FRAMES_PER_SECOND )
        {
            SDL_Delay( ( 1000 / FRAMES_PER_SECOND ) - fps.get_ticks() );
        }
    }
```

Now we have our main loop.

First we check for events then handle the key events for the dot then check for a user quit. Then we move the dot, clear the screen by filling it white and then we blit the dot on the screen. After that we update the screen, then the frame rate is capped.

Now that you're doing things that are real time, you're going to need to know how to properly structure your code flow. In article 4 I cover how to make a proper game loop. I recommend you take a look at it.

Download the media and source code for this tutorial here.