

La Clase Vector2D

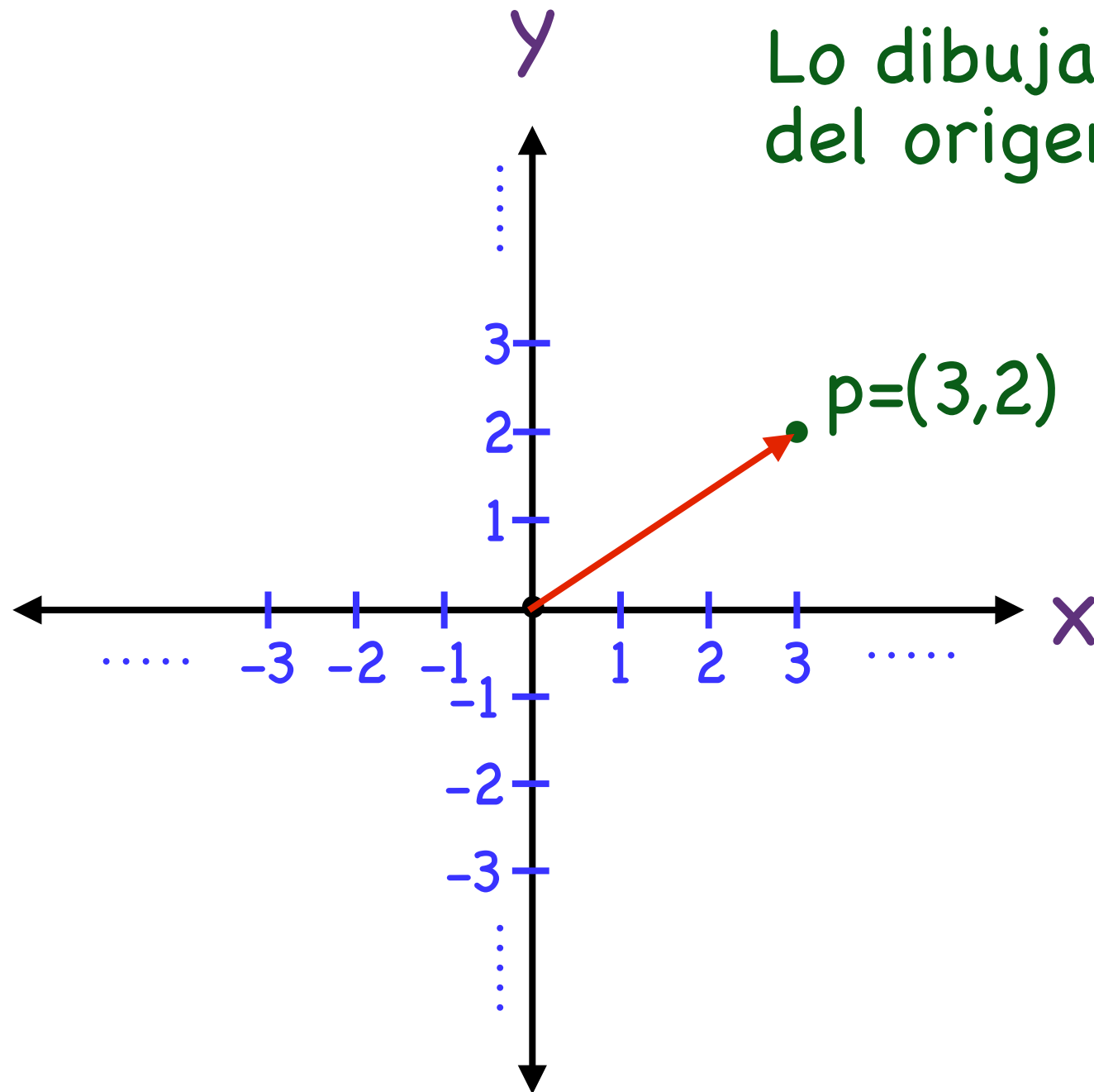
TPV 2

Samir Genaim

Vector2D

- ♦ Una clase que representa un vector en un plano bidimensional (las coordenadas de tipo **float**).
- ♦ Implementa operaciones sobre vectores (suma, resta, multiplicación escalar, rotación, magnitud, ...)
- ♦ **IMPORTANTE:** todas las operaciones (aparte de **operator=** y los setters) no modifican el vector, sino devuelven el resultado como otro vector
- ♦ Se usa para las características físicas de los objetos de juego: posición, vector de velocidad, etc.

Vector2D es un punto en el plano



Lo dibujamos como una linea que sale del origen (0,0) hasta el punto

```
Vector2D p(3.0f,2.0f);
```

```
float x = p.getX();
```

```
float y = p.getY();
```

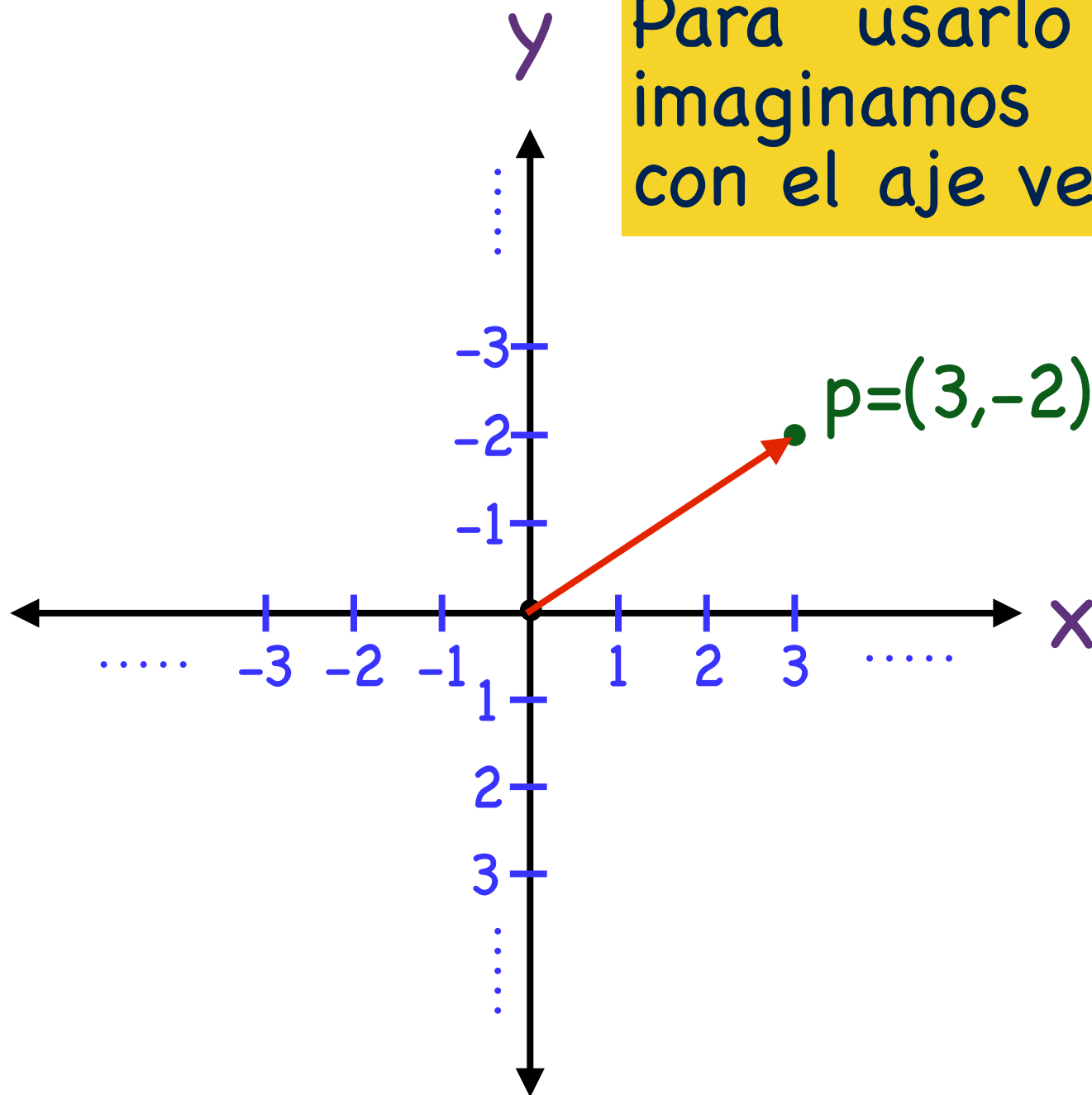
```
p.setY(4.0f);
```

```
p.setX(5.0f);
```

```
p.set(4.4f,5.2f);
```

Invertir el eje vertical

Para usarlo más fácilmente con SDL, imaginamos un sistema de coordenadas con el eje vertical invertido.



```
Vector2D p(3.0f,-2.0f);
```

```
float x = p.getX();
```

```
float y = p.getY();
```

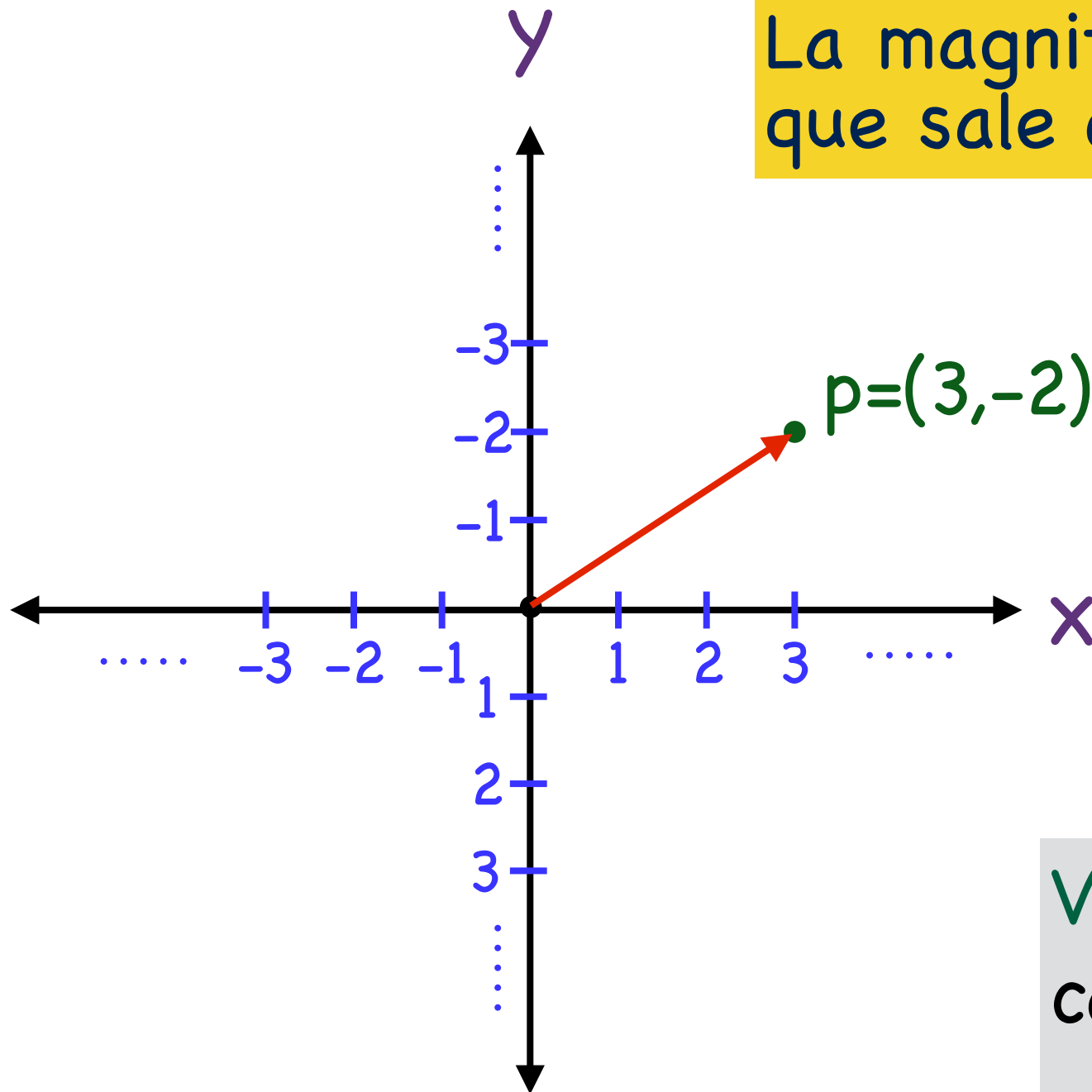
```
p.setY(4.0f);
```

```
p.setX(5.0f);
```

```
p.set(4.4f,5.2f);
```

Magnitud (Longitud)

La magnitud es la longitud de la línea que sale del origen hasta el punto

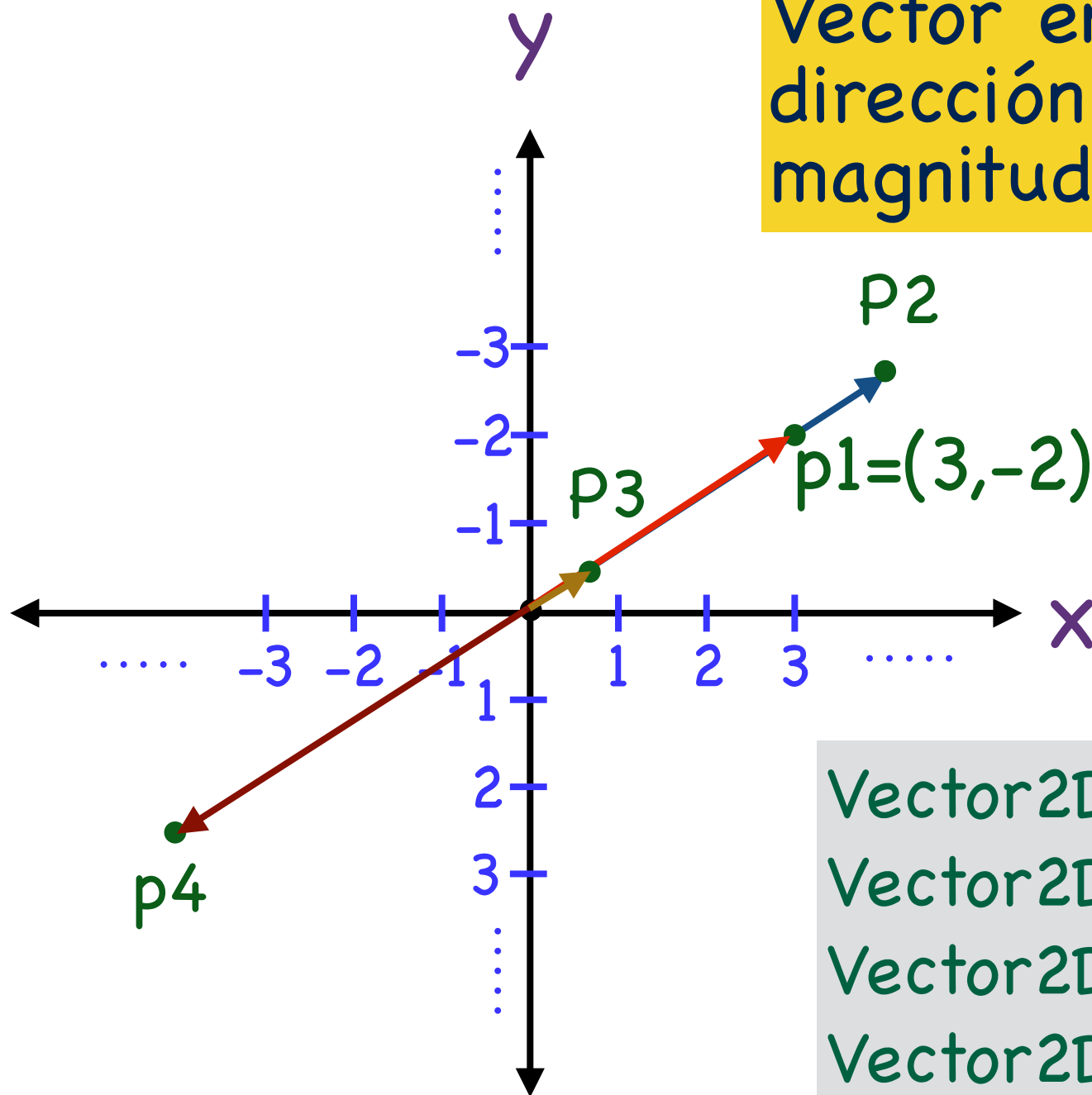


```
Vector2D p(3.0f, -2.0f);  
cout << p.magnitude() << endl;
```

3.60555

Mult./Division por Constante

Vector en la misma dirección (o en la dirección contraria) pero escalando la magnitud.



```
Vector2D p1(3.0f,-2.0f);
```

```
Vector2D p2 = p*1.4f;
```

```
Vector2D p3 = p/5.0f;
```

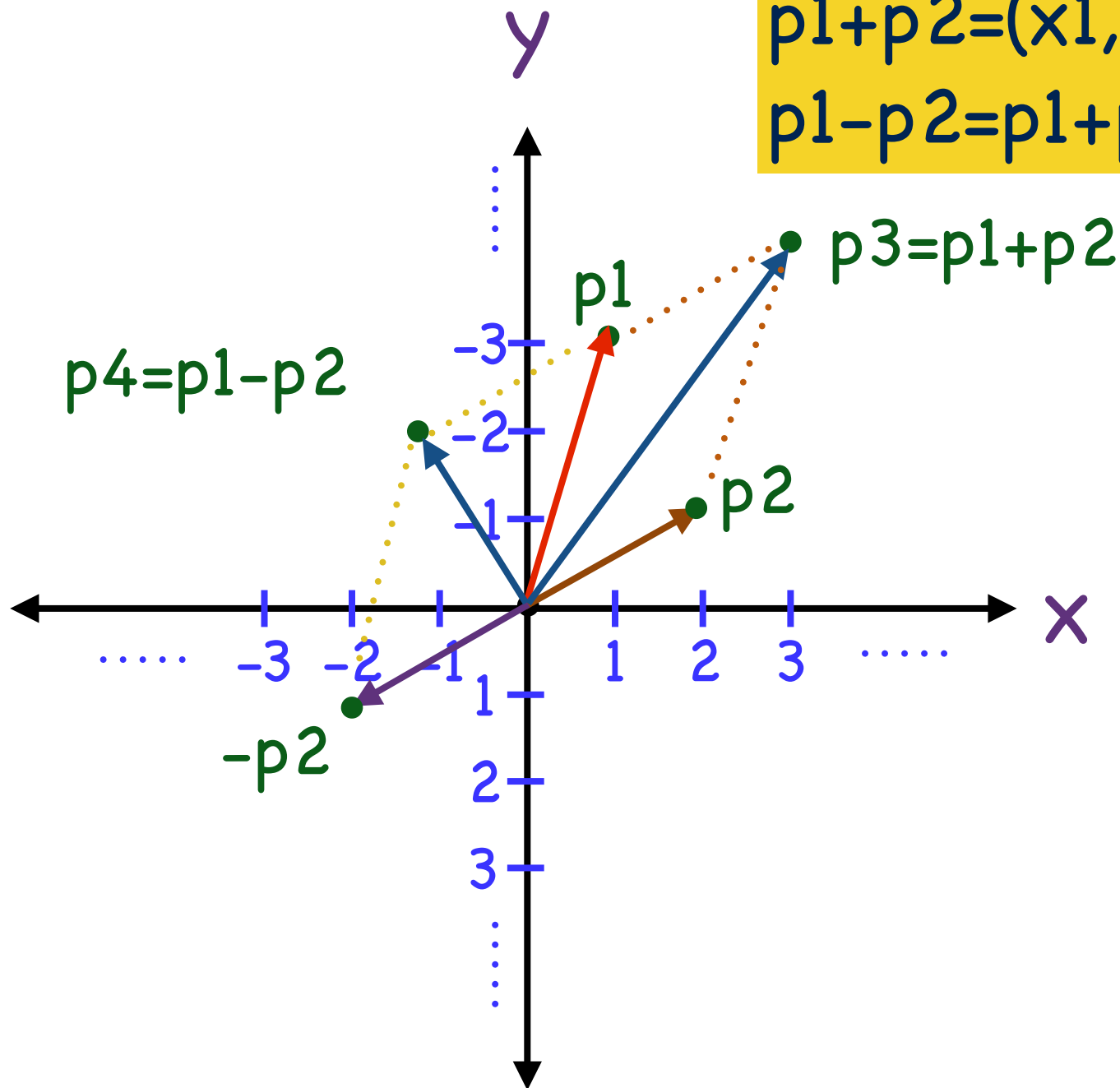
```
Vector2D p4 = p*(-1.4f);
```

```
cout << p2 << " " << p3 << " " << p4 << endl;
```

```
(4.2,-2.8) (0.6,-0.4) (-4.2,2.8)
```

Suma y Resta

$$p1+p2=(x1,y1)+(x2,y2)=(x1+x2,y2+y2)$$
$$p1-p2=p1+p2*(-1)$$



```
Vector2D p1(1.0f,-3.0f);
```

```
Vector2D p2(2.0f,-1.0f);
```

```
Vector2D p3 = p1+p2;
```

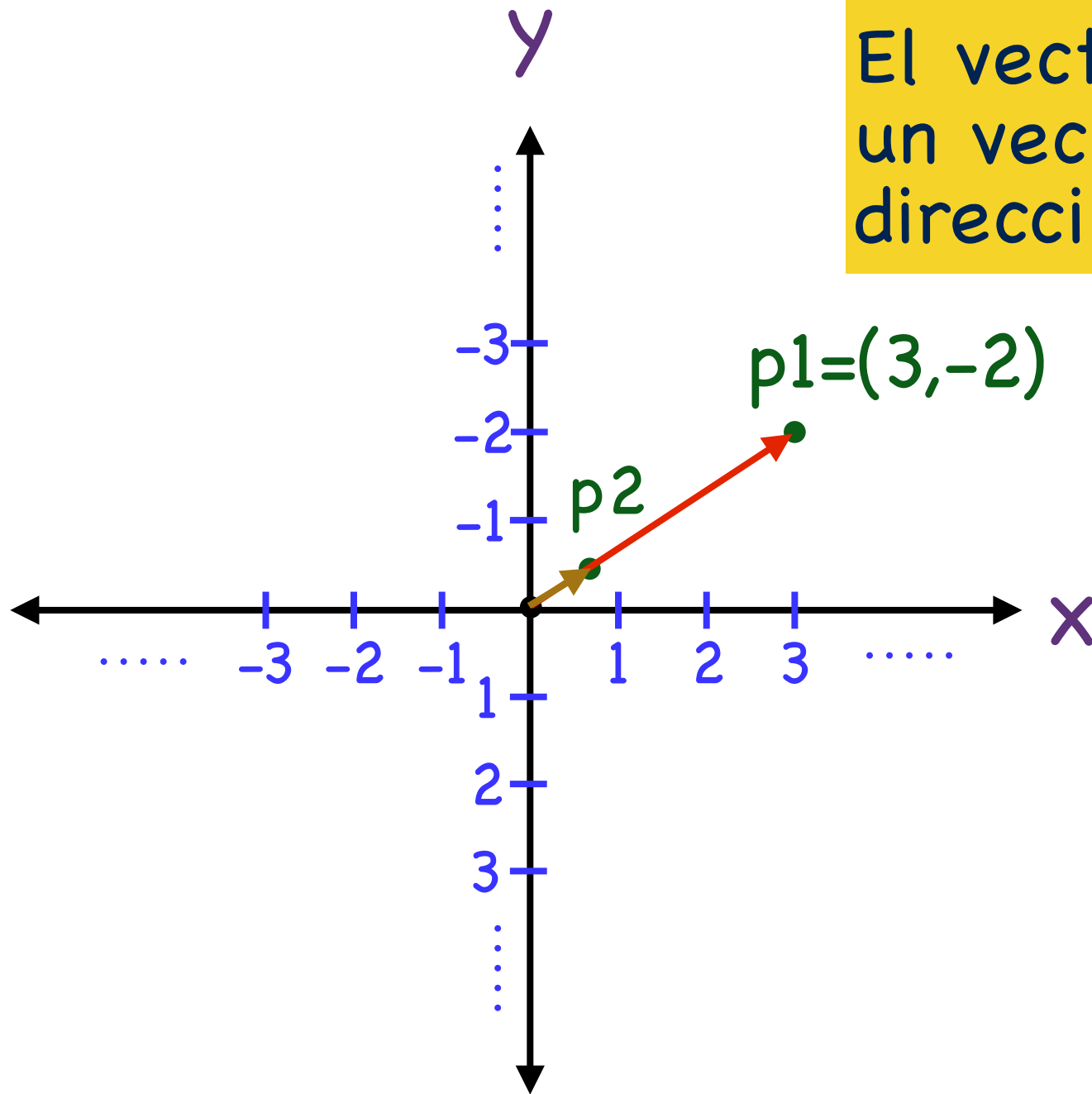
```
Vector2D p4 = p1-p2;
```

```
cout << p3 << " " << p4 << endl;
```

```
(3,-4) (-1,-2)
```

Vector Normal (dirección)

El vector normal (o la dirección) de un vector es un vector en la misma dirección pero tiene longitud 1.



```
Vector2D p1(3.0f,-2.0f);  
Vector2D p2 = p1.normalize();  
cout << p2 << endl;  
cout << p2.magnitude() << endl;
```

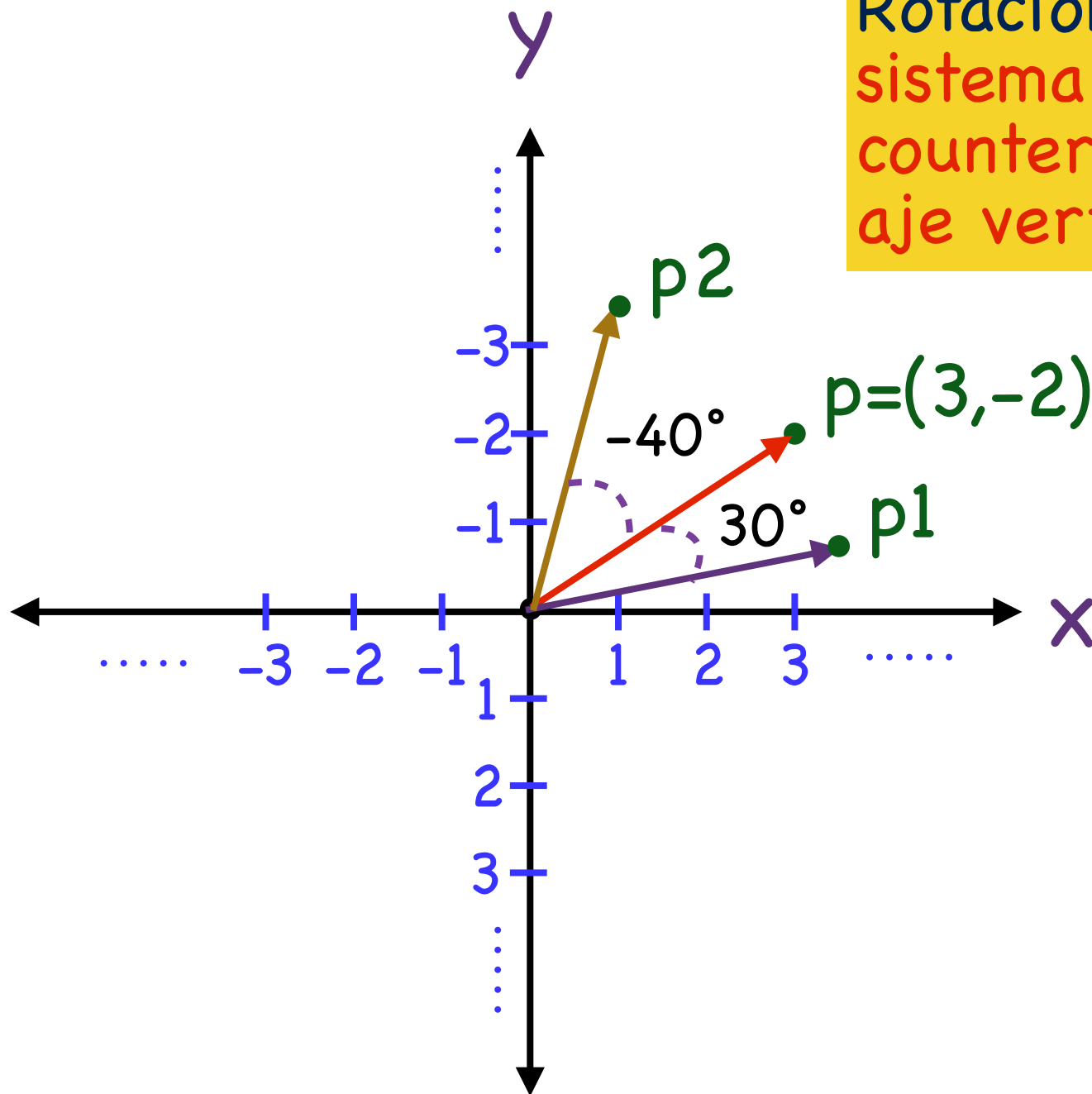
(0.83205,-0.5547)

1

Es muy útil para obtener un vector en la misma dirección de otro pero con diferente longitud.

Rotación

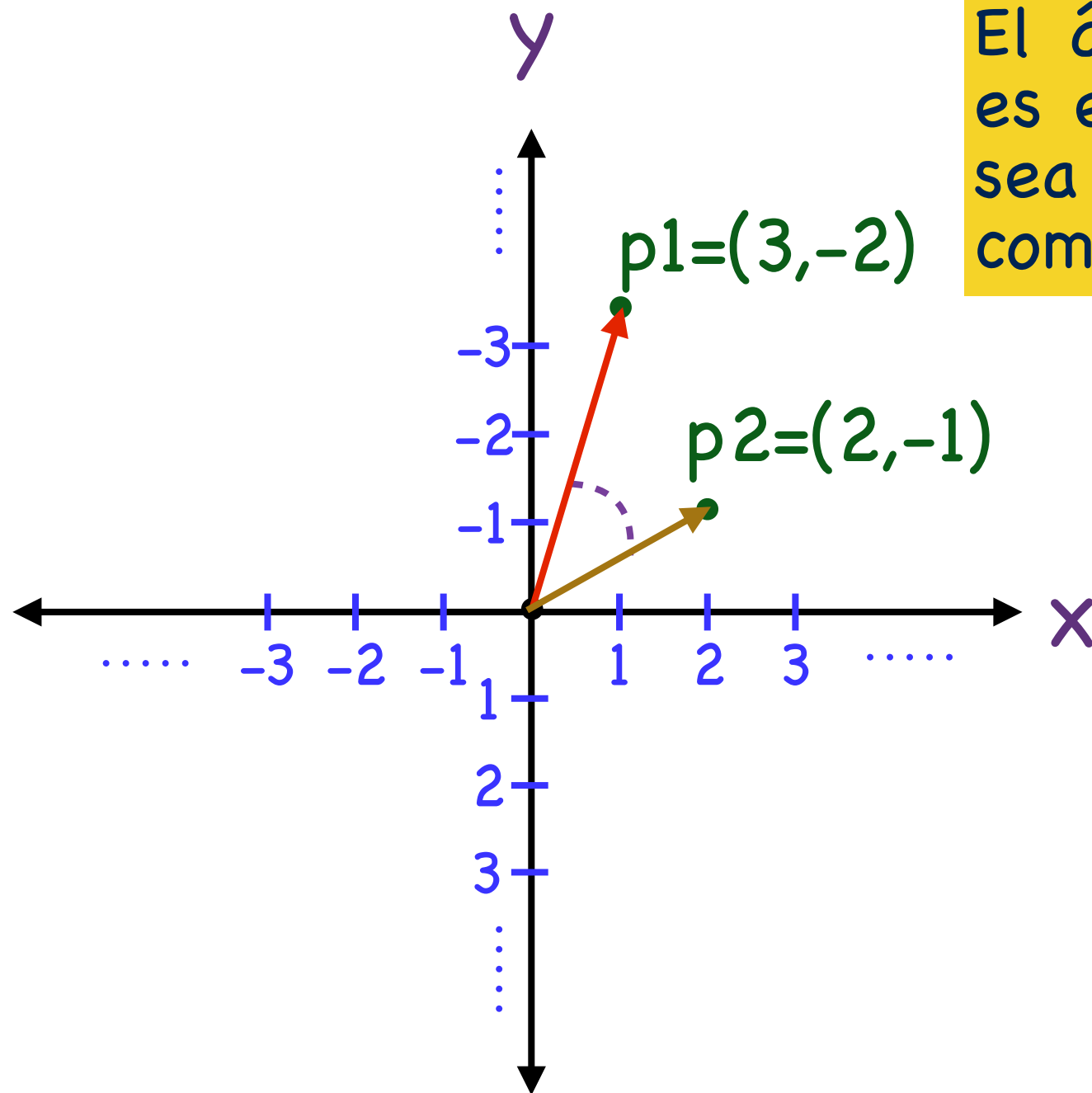
Rotación en α grados clockwise (en el sistema de coordenadas normal sería counterclockwise, pero como tenemos el eje vertical invertido es clockwise)



```
Vector2D p(3.0f,-2.0f);  
Vector2D p1 = p.rotate(30.0f);  
Vector2D p2 = p.rotate(-40.0f);  
cout << p1 << endl;  
cout << p2 << endl;
```

```
(3.59808,-0.232051)  
(1.01256,-3.46045)
```

Ángulo entre Vectores



El ángulo entre 2 vectores $p1$ y $p2$ es el ángulo α para que $p1.rotate(\alpha)$ sea un vector en la misma dirección como $p2$.

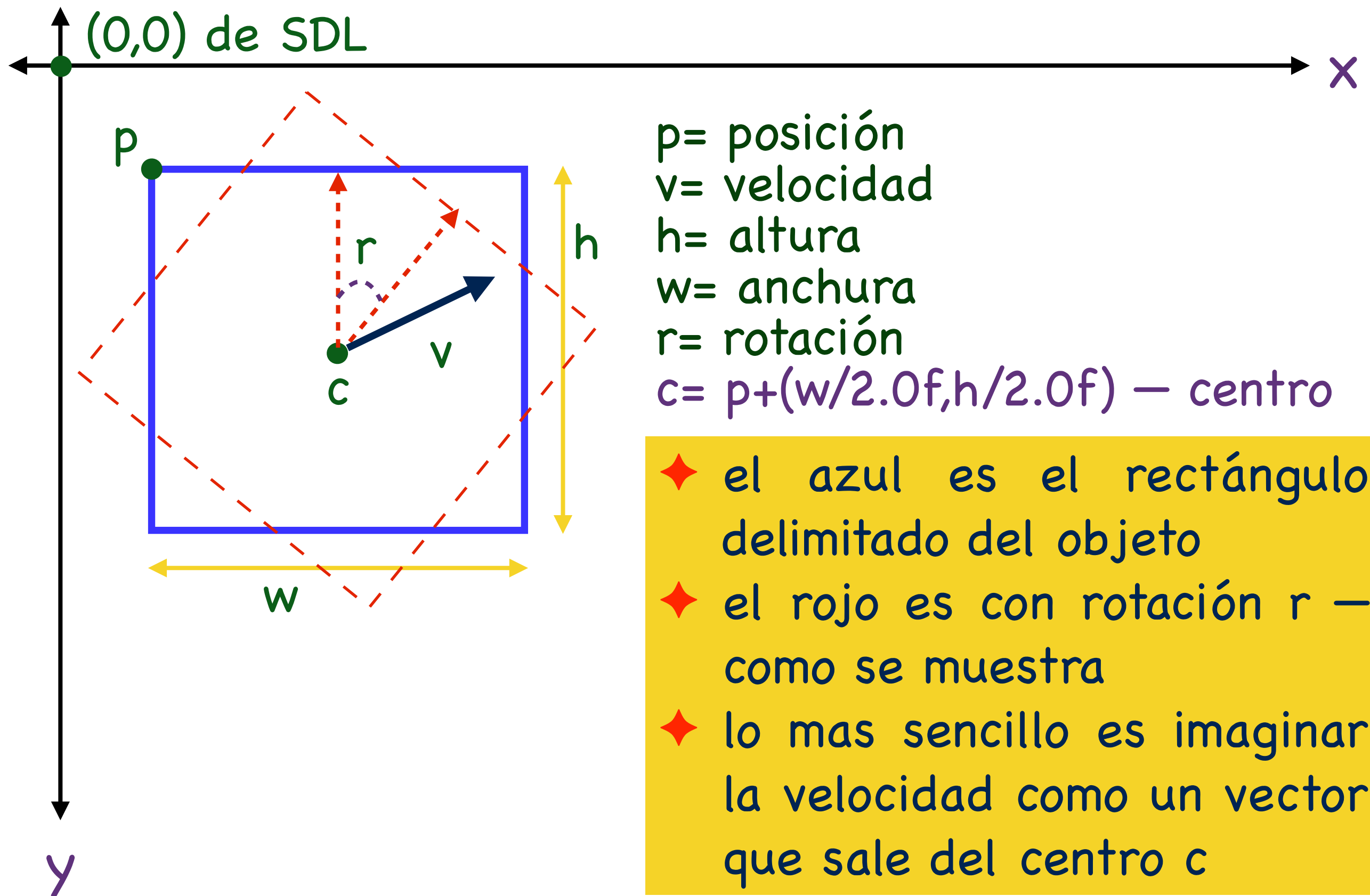
```
Vector2D p1(1.0f,-3.0f);  
Vector2D p2(2.0f,-1.0f);
```

```
cout << p1.angle(p2) << endl;  
cout << p2.angle(p1) << endl;
```

45

-45

Game Objects



p = posición

v = velocidad

h = altura

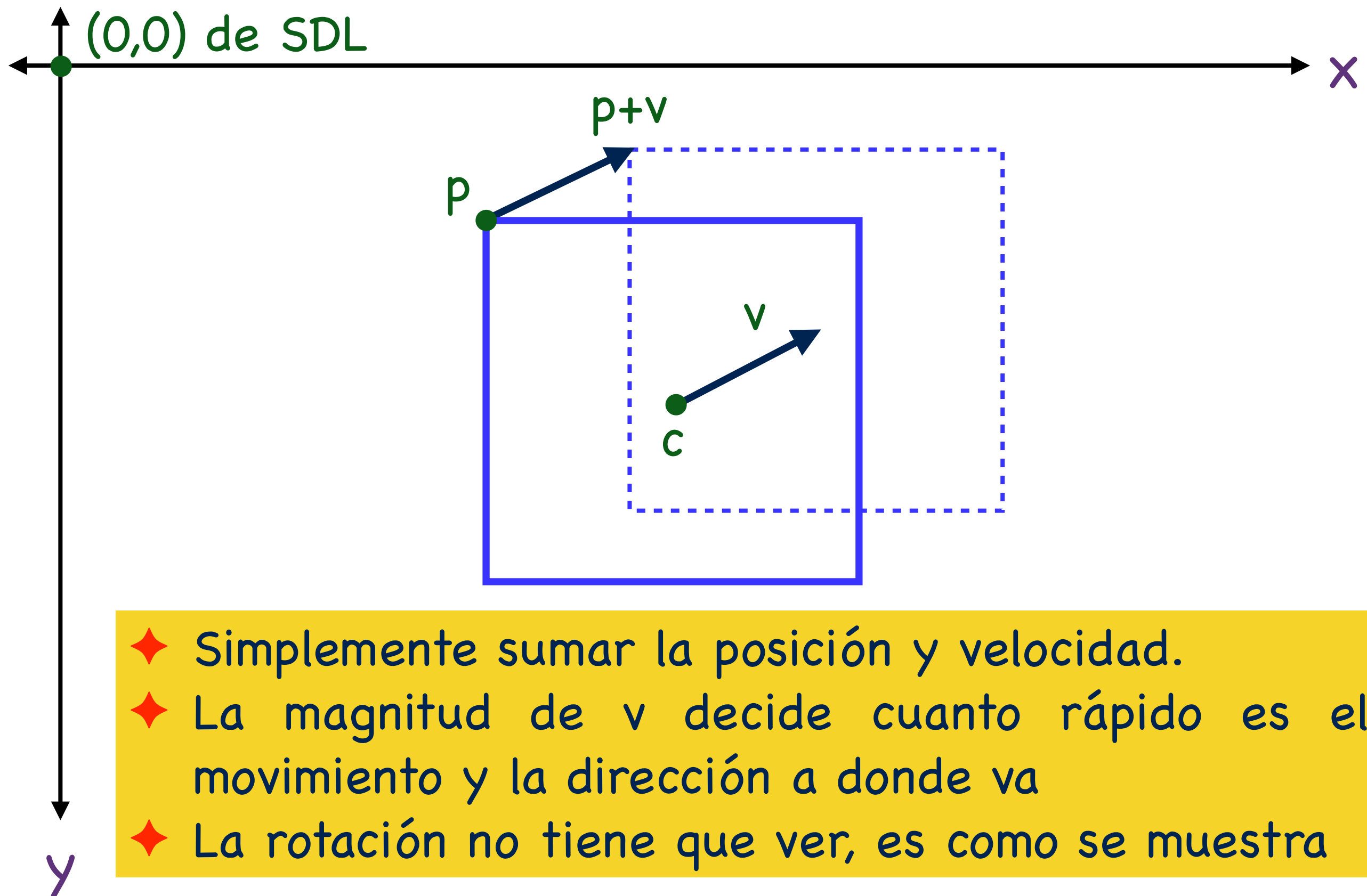
w = anchura

r = rotación

$c = p + (w/2.0f, h/2.0f)$ — centro

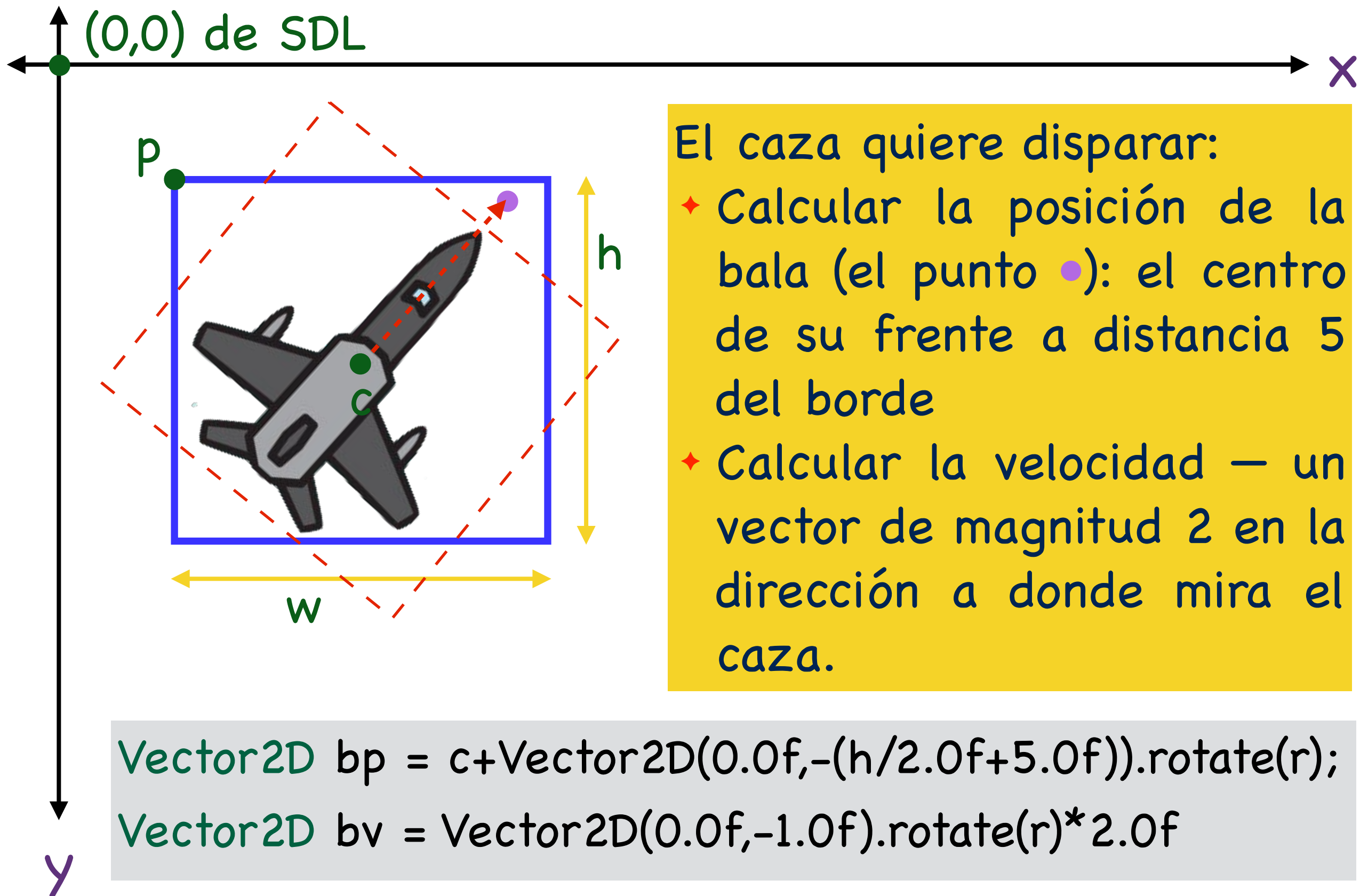
- ♦ el azul es el rectángulo delimitado del objeto
- ♦ el rojo es con rotación r — como se muestra
- ♦ lo mas sencillo es imaginar la velocidad como un vector que sale del centro c

Mover Game Objects



- ◆ Simplemente sumar la posición y velocidad.
- ◆ La magnitud de v decide cuanto rápido es el movimiento y la dirección a donde va
- ◆ La rotación no tiene que ver, es como se muestra

Ejercicio 1



Ejercicio 2

