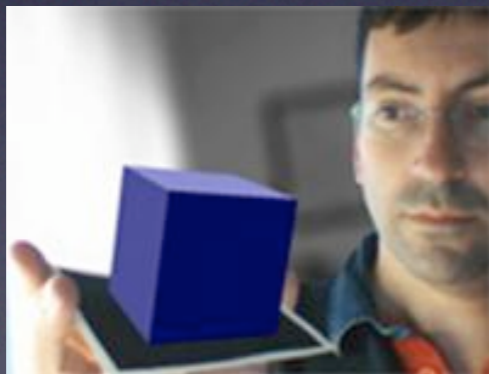


Gráficos por Computador

Interacción



Jordi Linares i Pellicer

Escola Politècnica Superior d'Alcoi

Dep. de Sistemes Informàtics i Computació

jlinares@dsic.upv.es

<http://www.dsic.upv.es/~jlinares>

Interacción

- *processing* facilita permite dos mecanismos para el control de la interacción con el usuario:
 - Mediante la consulta de variables del sistema
 - Mediante la definición de funciones de respuesta a eventos
- El uso de un mecanismo u otro depende de las necesidades de la aplicación y puede incluso combinarse
 - La consulta de las variables del sistema se realiza generalmente en el interior de la función `draw()`. Aunque la función se ejecuta de forma continua, no se garantiza la detección de todos los eventos de esta forma.
 - El uso de funciones de respuesta a eventos, gracias al uso de una cola para los mismos, garantiza la ejecución de las acciones asociadas a su invocación.

Interacción

`mouseX, mouseY`

- Almacenan siempre la posición actual del ratón:
 - `mouseX` => la coordenada x
 - `mouseY` => la coordenada y

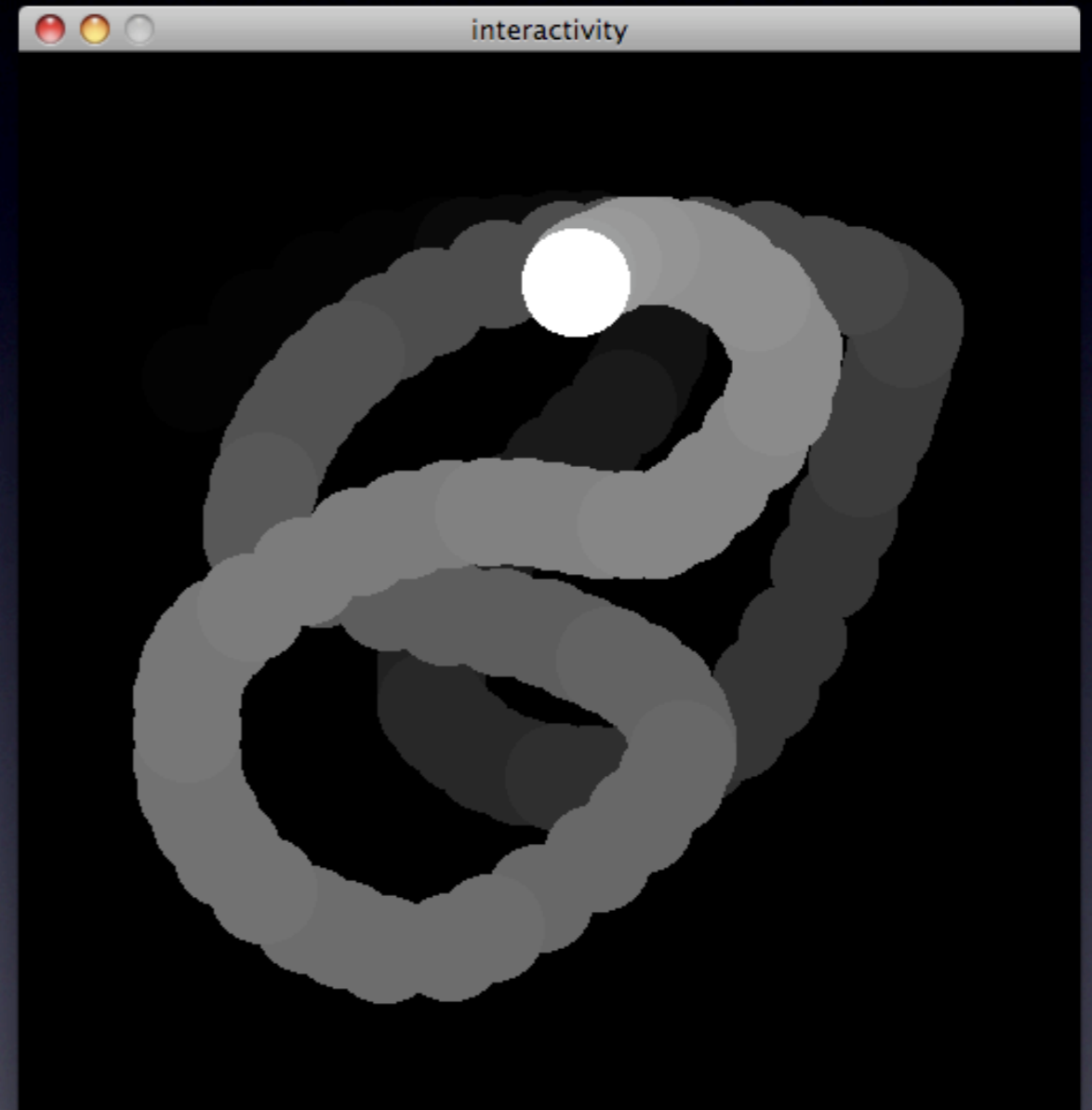
Interacción

```
// Una bola tras el cursor
void setup()
{
  size(500, 500);
  noStroke();

  // Fondo
  background(0);
}

void draw()
{
  // Efecto fundido
  fill(0, 1);
  rect(0, 0, width, height);

  // Dibujamos la bola en
  // las coordenadas actuales
  // del ratón
  fill(255);
  ellipse(mouseX, mouseY, 50, 50);
}
```



Interacción

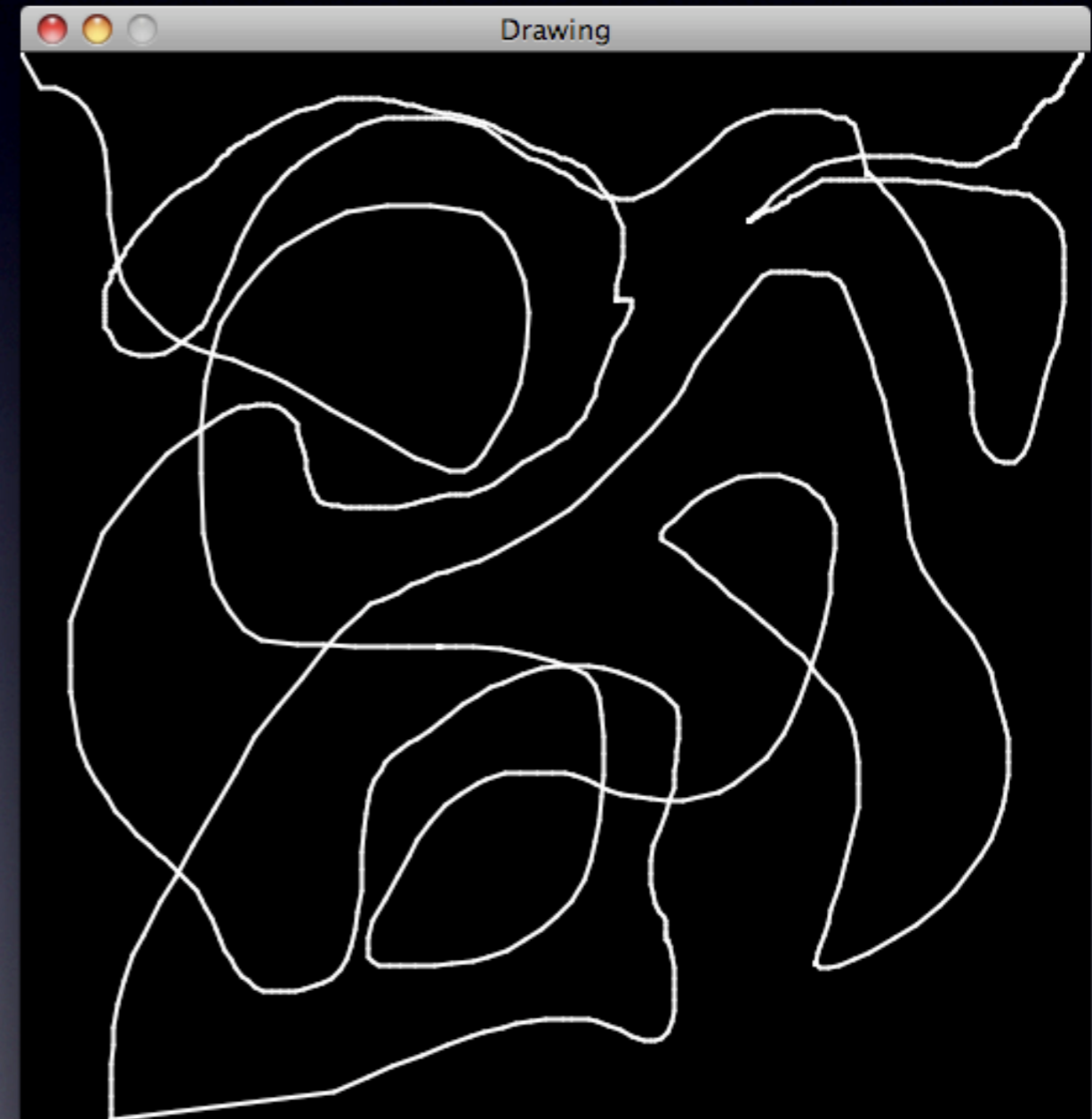
`pmouseX`, `pmouseY`

- Almacenan la posición del ratón en el frame anterior:
 - `pmouseX` => la coordenada x
 - `pmouseY` => la coordenada y

Interacción

```
// Dibujo libre
void setup()
{
  size(500, 500);
  background(0);
  stroke(255);
  strokeWeight(2);
  smooth(); // Aplica técnicas antialiasing
}

void draw()
{
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```



Interacción

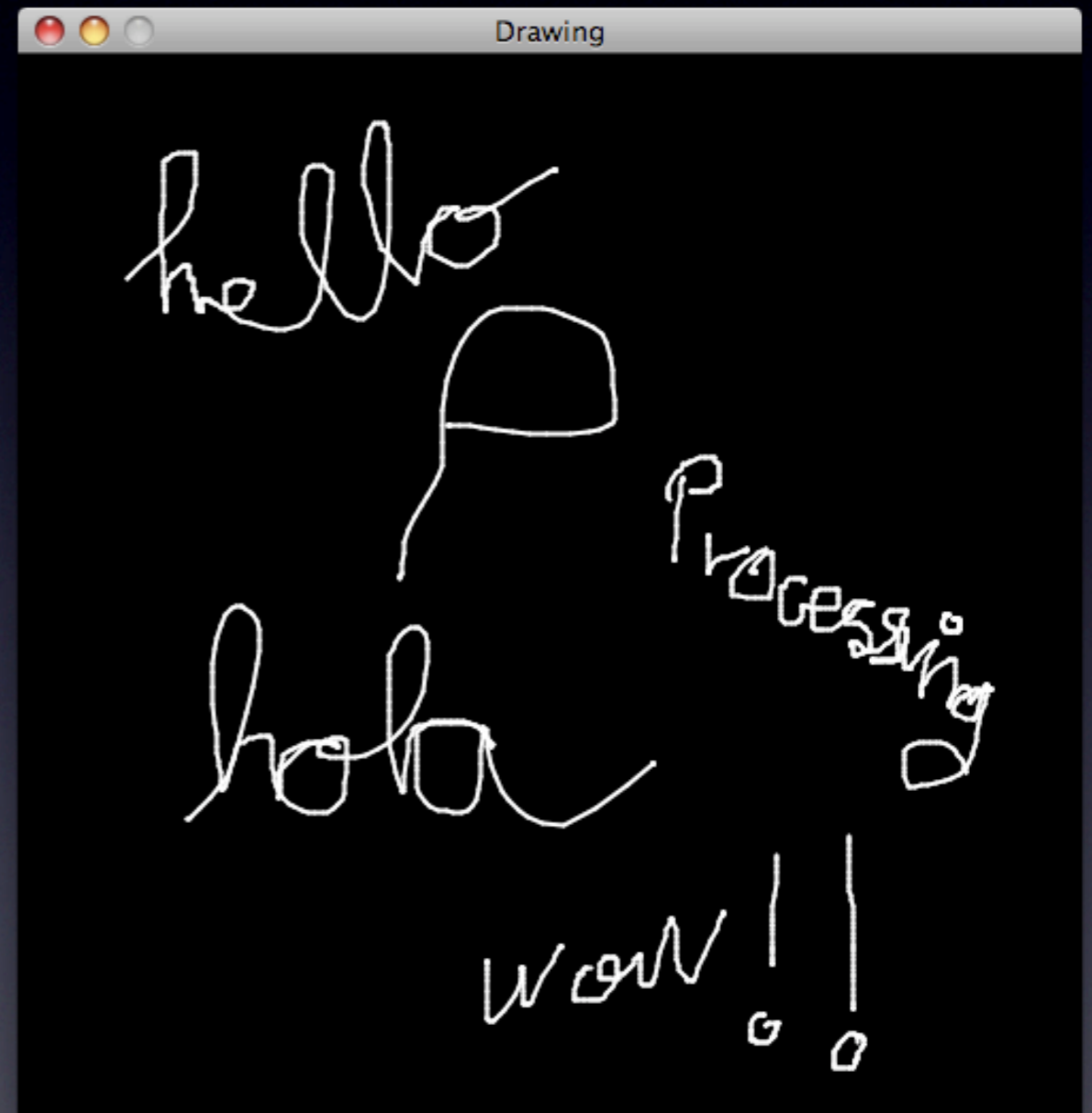
`mousePressed`

- Será `TRUE` si algún botón del ratón se encuentra pulsado y `FALSE` en caso contrario

Interacción

```
// Dibujo libre
void setup()
{
  size(500, 500);
  background(0);
  stroke(255);
  strokeWeight(2);
  smooth(); // Aplica técnicas antialiasing
}

void draw()
{
  if (mousePressed)
    line(mouseX, mouseY, mouseX, mouseY);
}
```



Interacción

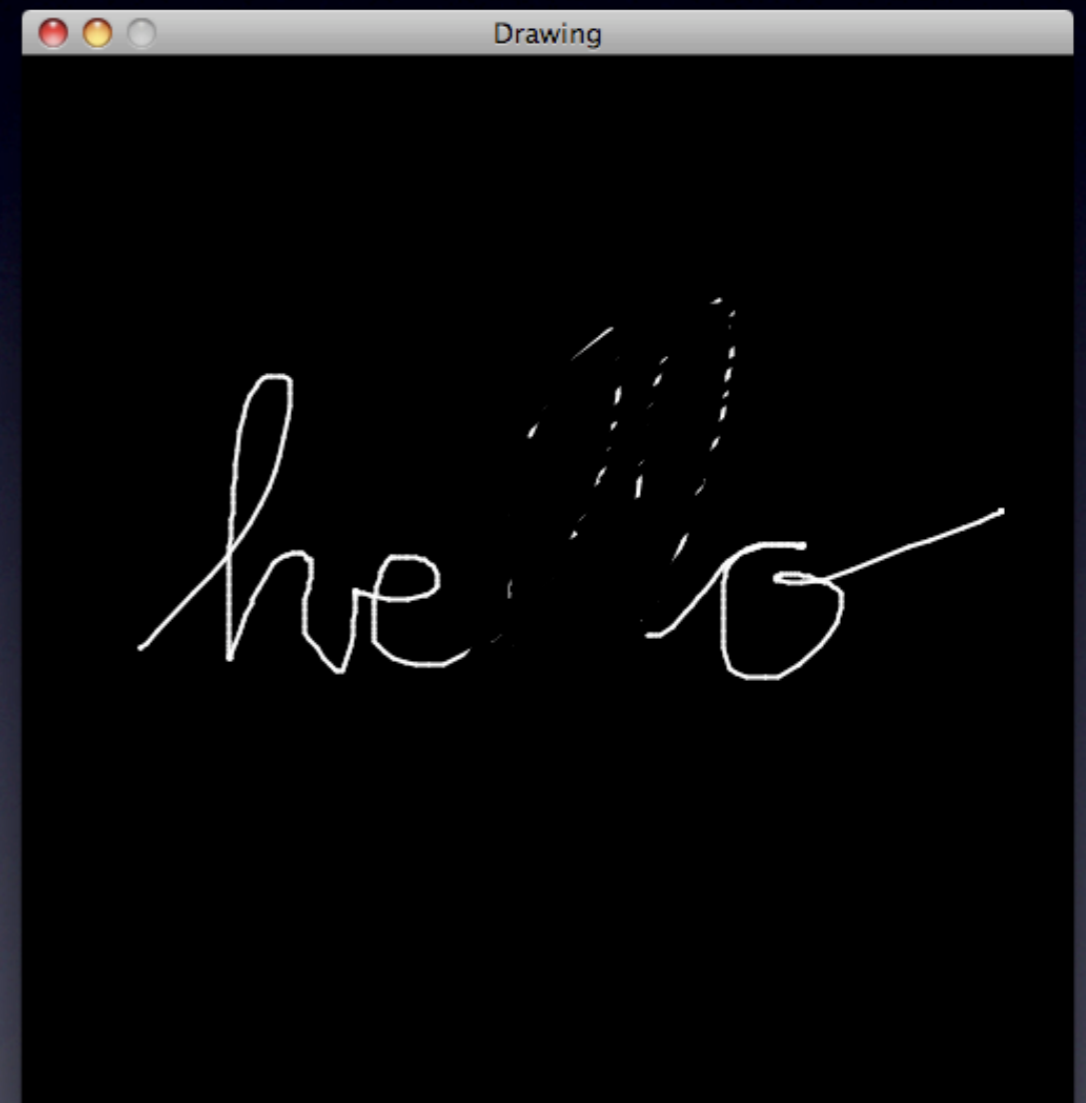
`mouseButton`

- Será `LEFT`, `RIGHT` o `CENTER` en función del botón pulsado
- Su uso debe combinarse con `mousePressed`

Interacción

```
// Dibujo libre
void setup()
{
  size(500, 500);
  background(0);
  strokeWeight(2);
  smooth(); // Aplica técnicas antialiasing
}

void draw()
{
  if (mousePressed) {
    if (mouseButton == LEFT) { // Dibujamos
      stroke(255);
      strokeWeight(2);
    }
    else { // Borrarnos
      stroke(0);
      strokeWeight(4);
    }
    line(mouseX, mouseY, pmouseX, pmouseY);
  }
}
```



Interacción

`keyPressed`

- Será `TRUE` si una tecla ha sido pulsada

`key`

- Será el carácter de la tecla pulsada.
- Puede contener también los valores: `BACKSPACE`, `TAB`, `ENTER`, `RETURN`, `ESC`, y `DELETE`
- Si su valor coincide con la constante `CODED`, la variable del sistema `keyCode` nos indicará la pulsación de alguna tecla especial: `UP`, `DOWN`, `LEFT`, `RIGHT`, `ALT`, `CONTROL`, `SHIFT`.
- Su uso debe combinarse con `keyPressed`

Interacción

Definición de funciones de respuesta a eventos

- La definición de funciones de respuesta a eventos consiste simplemente en añadir al programa una nueva función que será ejecutada siempre que el evento asociado se produzca
- Estas funciones tienen nombres específicos, sin parámetros y retornan `void`
- En su interior, puede convenir la consulta de las variables del sistema detalladas anteriormente

Interacción

```
void mousePressed()
```

- Se llama siempre después de la pulsación de un botón del ratón
- Con `mouseButton` podríamos determinar el botón concreto
- Detecta concretamente la acción de pulsación (primer click al presionar)

```
void mouseReleased()
```

- Se llama siempre después de la liberación de un botón del ratón, es decir, el segundo click tras una acción de pulsación previa
- Con `mouseButton` podríamos determinar el botón concreto

Interacción

```
void mouseClicked()
```

- Se llama siempre después de realizar una pulsación completa, es decir, presión y liberación
- Con `mouseButton` podríamos determinar el botón concreto
- Con anterioridad se habrán generado igualmente los eventos `mousePressed()` y `mouseReleased()`

Interacción

```
void mouseMoved()
```

- Se llama siempre después de un movimiento de la posición del ratón si que ningún botón se encuentre pulsado
- Es habitual utilizar `mouseX` y `mouseY` en su interior para consultar la posición del ratón

```
void mouseDragged()
```

- Se llama siempre después de un movimiento de la posición del ratón cuando un botón se encuentre pulsado (`mouseButton` nos permitirá conocer cual)
- Es habitual utilizar `mouseX` y `mouseY` en su interior para consultar la posición del ratón

Interacción

```
void keyPressed()
```

- Se llama siempre después de la pulsación de una tecla (acción de presión sobre ella)
- Es habitual utilizar `key` en su interior para consultar la tecla en cuestión (`keyCode` también puede utilizarse)

```
void keyReleased()
```

- Se llama siempre después de la liberación de una tecla (soltarla tras ser pulsada)
- Es habitual utilizar `key` en su interior para consultar la tecla en cuestión (`keyCode` también puede utilizarse)

Interacción

```
void keyTyped()
```

- Se llama siempre después de la pulsación completa de una tecla (acción de presión y liberación)
- Los eventos `keyPressed()` y `keyReleased()` serán invocados con anterioridad
- La pulsación continua de teclas lanzará repetidos eventos `keyTyped()` según esté establecido en el sistema operativo
- Las teclas Control, Alt y Shift son ignoradas en este proceso

Interacción

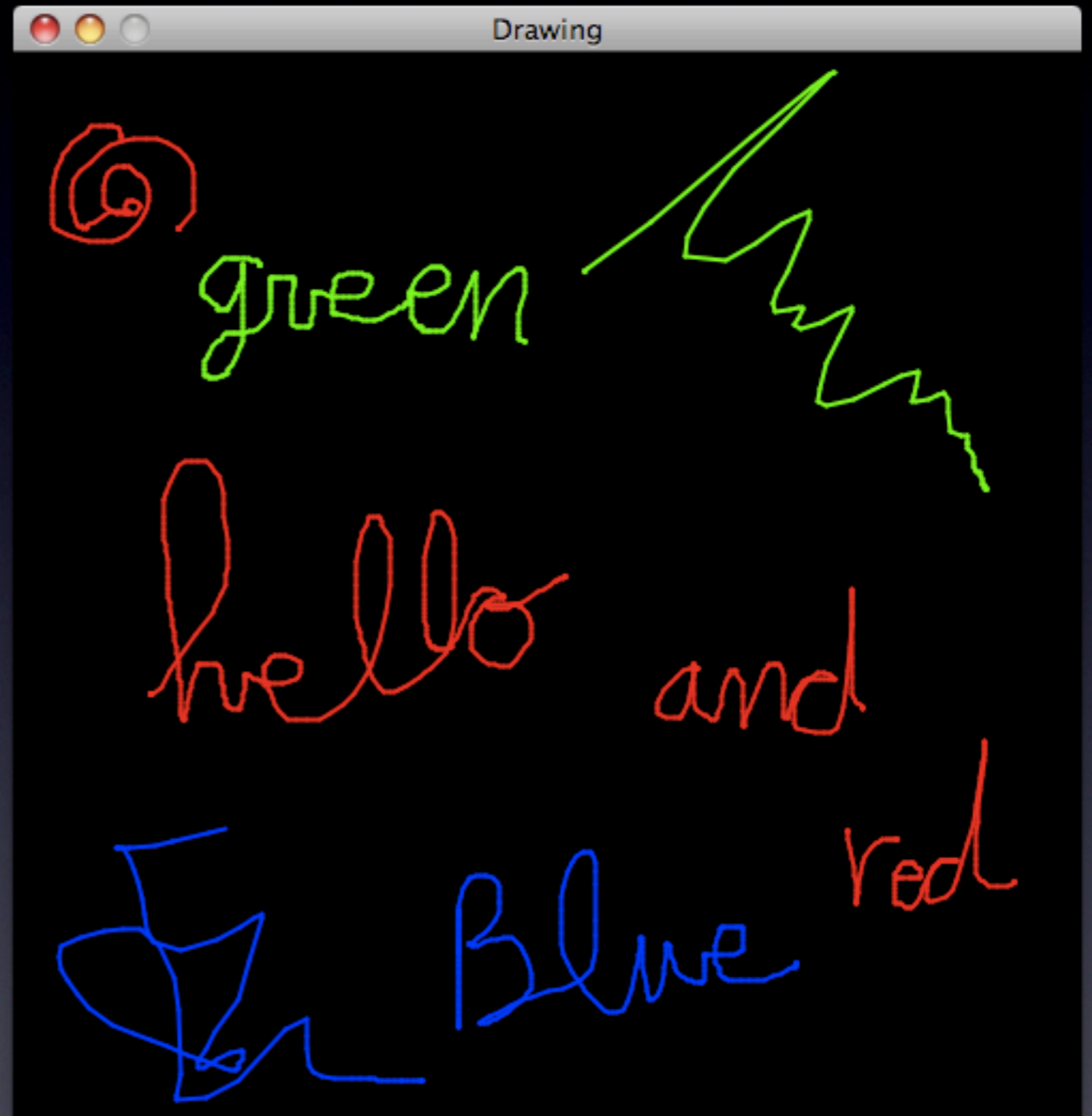
```
// Dibujo libre
color colorStroke = color(255, 0, 0);

void setup()
{
  size(500, 500);
  background(0);
  strokeWeight(2);
  smooth(); // Aplica técnicas antialiasing
}

void draw()
{
  if (mousePressed) {
    if (mouseButton == LEFT) { // Dibujamos
      stroke(colorStroke);
      strokeWeight(2);
    }
    else {
      stroke(0);
      strokeWeight(4);
    }

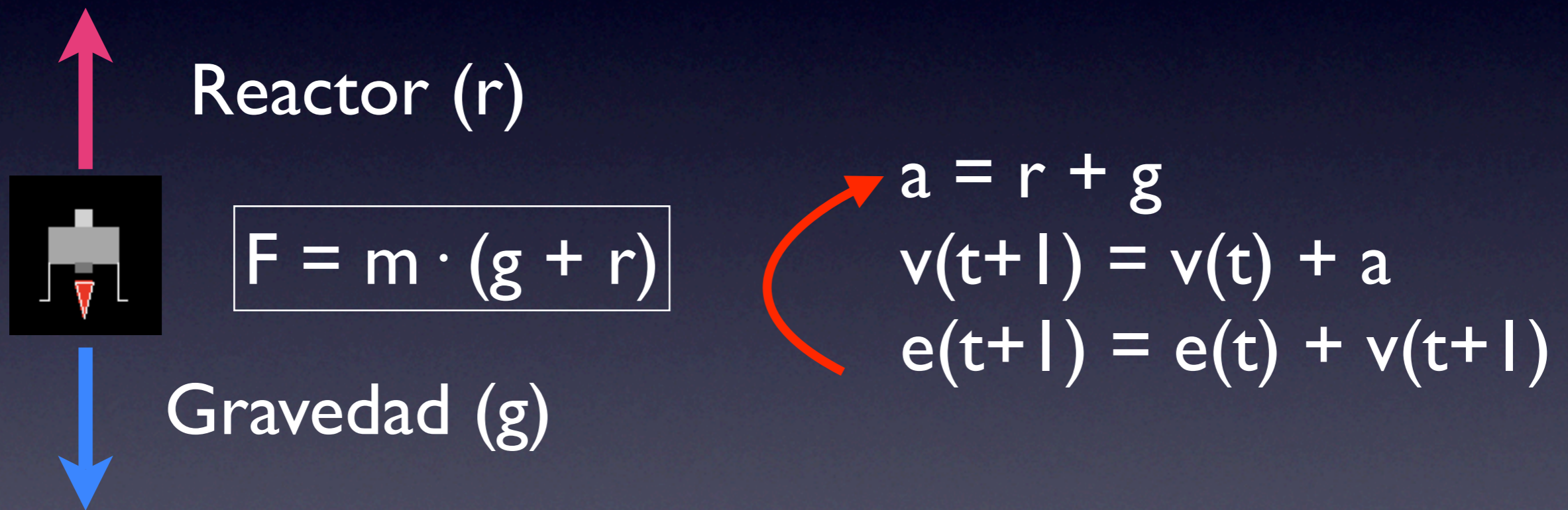
    line(mouseX, mouseY, pmouseX, pmouseY);
  }
}

void keyPressed()
{
  switch (key) {
    case 'r':
    case 'R':
      colorStroke = color(255,0,0);
      break;
    case 'g':
    case 'G':
      colorStroke = color(0,255,0);
      break;
    case 'b':
    case 'B':
      colorStroke = color(0,0,255);
      break;
  }
}
```



Práctica 6-1

- Desarrollar el juego 'Lunar Landing'



Práctica 6-1

- La nave es ubicada inicialmente en la parte superior en una posición horizontal aleatoria
- En la superficie, dibujar la superficie de la luna como un simple rectángulo, en donde se ubica, en el centro, la base de aterrizaje
- La nave se verá afectada por la fuerza de la gravedad (de igual forma que en el ejemplo de gravedad visto en el tema anterior)
- Cuando el usuario pulsa la tecla del cursor UP, y sólo en ese caso, la fuerza final aplicada a la nave tendrá también en cuenta el reactor (encendido o apagado, sin niveles intermedios)
- La pulsación de las teclas LEFT y RIGHT deben permitir desplazar la nave a la izquierda y la derecha (desplazar sin más consideración adicional)

Práctica 6-1

- El aterrizaje será correcto únicamente si:
 - La nave está dentro de la plataforma de aterrizaje
 - La velocidad de la nave en ese momento es inferior a un determinado umbral. Si se supera, se considerará que la nave se estrella aunque o haga sobre la plataforma.
- Tras un aterrizaje correcto o incorrecto, se reanudará el juego, ubicando la nave en la posición inicial
- Dibujar libremente la nave (se aconseja el uso de una traslación para su ubicación en pantalla)
- Si se pulsa el reactor, se puede dibujar el fuego de la propulsión
- También se puede dibujar la colisión de la nave
- Ajustar los valores asociados a la gravedad y la fuerza del reactor, así como los desplazamientos laterales para dar emoción al juego
- La interacción puede ser resuelta en su totalidad en el `draw()` mediante la lectura de las variables del sistema

Práctica 6-1

