Design Programming

DECO1012
Rob Saunders
Rob Saunders

web: http://web.arch.usyd.edu.au/~rob

e-mail: rob@arch.usyd.edu.au

office: Room 274, Wilkinson Building
Interaction
Interaction Design & Human Computer Interaction

- Human Computer Interaction (HCI) and Interaction Design are terms used for the research, development and study of the ways that humans and computers interact.
- It includes the research, development and study of hardware and software interfaces that try to make the process of interacting with a computer either simpler or more enjoyable.
Human Computer Interfaces

- The most common hardware devices used to interact with desktop computers is the mouse and keyboard.
- These are used to manipulate the desktop interfaces, mostly comprised of windows, icons, buttons, etc., that have become the dominant way to work with computers.
Mouse
Mouse

- The Processing variables `mouseX` and `mouseY` store the x-coordinate and y-coordinate of the cursor relative to the origin in the upper-left corner of the display window.

- If `mouseX` and `mouseY` are used in programs without a `draw()` or if `noLoop()` is run in `setup()`, the values will always be 0.
Mouse Example 1

// Circle follows the cursor
void setup() {
    size(100, 100);
    smooth();
    noStroke();
}

void draw() {
    background(126);
    ellipse(mouseX, mouseY, 33, 33);
}
// Invert cursor position to create a secondary response
void setup() {
    size(100, 100);
    noStroke();
    smooth();
}
void draw() {
    float x = mouseX;
    float y = mouseY;
    float ix = width - mouseX; // Inverse X
    float iy = mouseY - height; // Inverse Y
    background(126);
    fill(255, 150);
    ellipse(x, height/2, y, y);
    fill(0, 159);
    ellipse(ix, height/2, iy, iy);
}
Tracking the Mouse Over Frames

- The Processing variables `pmouseX` and `pmouseY` store the mouse values from the previous frame.
  - If the mouse does not move, the values will be the same, but if the mouse is moving quickly there can be large differences between the values.
// Draw a line between the current and previous positions
void setup() {
    size(100, 100);
    strokeWeight(8);
    smooth();
}

void draw() {
    background(204);
    line(mouseX, mouseY, pmouseX, pmouseY);
}
Mouse Buttons

- Computer mice and other similar input devices typically have between one and three buttons, and Processing can detect when these buttons are pressed.
- The `mousePressed` variable is `true` if any mouse button is pressed and `false` if no mouse button is pressed.
- The variable `mouseButton` is `LEFT`, `CENTER`, or `RIGHT` depending on the mouse button most recently pressed.
Mouse Example 4

// Set the square to black when the left mouse button
// is pressed and white when the right button is pressed
void setup() {
  size(100, 100);
}

void draw() {
  if (mouseButton == LEFT) {
    fill(0); // Black
  } else if (mouseButton == RIGHT) {
    fill(255); // White
  } else {
    fill(126); // Gray
  }
  rect(25, 25, 50, 50);
}
Cursor Icon

› The cursor icon used to show the position of the mouse can be hidden with the noCursor() function and can be set with the cursor() function to appear as different icons.

› The options for the cursor() function’s single parameter are ARROW, CROSS, HAND, MOVE, TEXT, and WAIT.

› A custom cursor can be drawn and controlled with the mouseX and mouseY variables.
Keyboard
Keyboard

- Processing registers the most recently pressed key and whether a key is currently pressed.
  - The boolean variable keyPressed is true if a key is pressed and is false if not.
  - The keyPressed variable remains true while the key is held down and becomes false only when the key is released.
  - Including this variable in an if structure allows lines of code to run only if a key is pressed.
Keyboard Example 1

// Draw a rectangle when any key is pressed
void setup() {
  size(100, 100);
  smooth();
  strokeWeight(4);
}

void draw() {
  background(204);
  if (keyPressed == true) { // If the key is pressed,
    line(20, 20, 80, 80); // draw a line
  } else { // Otherwise,
    rect(40, 40, 20, 20); // draw a rectangle
  }
}
Getting the Key Pressed

- The key variable is of the char data type and stores the most recently pressed key.
- The key variable can store only one value at a time. The most recent key pressed will be the only value stored in the variable.
PFont font;

void setup() {
    size(100, 100);
    font = loadFont("ThesisMonoLight-72.vlw");
    textFont(font);
}

void draw() {
    background(0);
    text(key, 28, 75);
}
Using **keyPressed** and **key** in Conditional Statements

- The **key** variable may be used to determine whether a specific key is pressed.
- The following example uses the expression `key=='A'` to test if the A key is pressed. The single quotes signify A as the data type `char`.
- The logical AND symbol, the `&&` operator, is used to connect the expression with the `keyPressed` variable to ascertain that the key pressed is the uppercase A.
void setup() {
    size(100, 100);
    smooth();
    strokeWeight(4);
}

void draw() {
    background(204);
    // If the 'A' key is pressed draw a line
    if ((keyPressed == true) && (key == 'A')) {
        line(50, 25, 50, 75);
    } else { // Otherwise, draw an ellipse
        ellipse(50, 50, 50, 50);
    }
}
Checking for ‘a’ or ‘A’

- If you want to check for both uppercase and lowercase letters, you have to extend the relational expression with a logical OR, the || relational operator, for example:

```java
if ((keyPressed == true) && ((key == 'a') || (key == 'A'))) {
```
Coded Keys

- Processing can also read the values from other keys including the arrow keys and the Alt, Control, Shift, Backspace, Tab, Enter, Return, Escape, and Delete keys.
- The variable `keyCode` stores the ALT, CONTROL, SHIFT, UP, DOWN, LEFT, and RIGHT keys as constants.
- The expression `key==CODED` is true if the key is coded and false otherwise.
Coded Keys

- Even though not alphanumerical, some keys included in the ASCII specification (TAB, BACKSPACE, ENTER, RETURN, ESC, and DELETE) will not be identified as coded.

- For cross-platform projects, note that the Enter key is commonly used on PCs and UNIX and the Return key is used on Macintosh. Check for both Enter and Return to make sure your program will work for all platforms.
color y = 35;

void setup() {
    size(100, 100);
}

void draw() {
    background(204);
    line(10, 50, 90, 50);
    if (key == CODED) {
        if (keyCode == UP) y = 20;
        if (keyCode == DOWN) y = 50;
    } else {
        y = 35;
    }
    rect(25, y, 50, 30);
}
Events
Events

- Functions called event handlers alter the normal flow of a program when an action such as a key press or mouse movement takes place.
- The code inside an event function is run once each time the corresponding event occurs.
Mouse Events

- The mouse event functions are:
  - `mousePressed()`
    - A mouse button has been pressed
  - `mouseReleased()`
    - A mouse button has been released
  - `mouseMoved()`
    - The mouse has been moved
  - `mouseDragged()`
    - The mouse has been “dragged”
void setup() {
    size(100, 100);
    fill(0, 102);
}

void draw() { } // Empty draw() keeps the program running

void mousePressed() {
    rect(mouseX, mouseY, 33, 33);
}
Mouse Events Example 2

```java
int dragX, dragY, moveX, moveY;
void setup() {
    size(100, 100);
    smooth();
    noStroke();
}
void draw() {
    background(204);
    fill(0);
    ellipse(dragX, dragY, 33, 33); // Black circle
    fill(153);
    ellipse(moveX, moveY, 33, 33); // Gray circle
}
// Move gray circle
void mouseMoved() { moveX = mouseX; moveY = mouseY; }
// Move black circle
void mouseDragged() { dragX = mouseX; dragY = mouseY; }
```
Keyboard Events

- Each key press is registered through the keyboard events:
  - keyPressed()
    - A key has been pressed
  - keyReleased()
    - A key has been released
void setup() {
    size(100, 100);
    noStroke();
    fill(255, 51);
}

void draw() {} // Empty draw() keeps the program running

void keyPressed() {
    int y = key - 32;
    rect(0, y, 100, 4);
}
String letters = "";

void setup() {
    textFont(loadFont("Eureka-24.vlw"));
    stroke(255);
    fill(0);
}

void draw() {
    background(204);
    line(textWidth(letters), 0, textWidth(letters), 100);
    text(letters, 0, 50);
}

void keyPressed() {
    if (key == BACKSPACE) {
        // Backspace
        if (letters.length() > 0) {
            letters = letters.substring(0, letters.length()-1);
        }
    } else if (textWidth(letters+key) < width) {
        letters = letters+key;
    }
}
int back = 102;

// setup() same as before...

void draw() {
    background(back);
    text(letters, 50, 50);
}

void keyPressed() {
    if ((key == ENTER) || (key == RETURN)) {
        letters = letters.toLowerCase();
        println(letters); // Print to console to see input
        if (letters.equals("black")) back = 0;
        if (letters.equals("gray")) back = 204;
        letters = ""; // Clear the variable
    }
    else if ((key > 31) && (key != CODED)) {
        // If the key is alphanumeric, add it to the String
        letters = letters + key;
    }
}
Controlling the Flow

- Programs written with `draw()` display frames to the screen as quickly as possible.
- The `frameRate()` function is used to set a limit on the number of frames that will display each second, and the `noLoop()` function can be used to stop `draw()` from looping.
- The additional functions `loop()` and `redraw()` provide more options when used in combination with the mouse and keyboard event functions.
Controlling the Flow

- Running `loop()` resumes a paused sketch.
- The event functions continue to run when a program is paused with `noLoop()`, the `loop()` function can be used within these events to continue running the code in `draw()`.
- The `redraw()` function runs the code in `draw()` one time and then halts the execution. It’s helpful when the display needn’t be updated continuously.
void setup() {
    size(100, 100);
    noLoop();
}

void draw() {
    background(204);
    line(mouseX, 0, mouseX, 100);
}

void mousePressed() {
    redraw(); // Run the code in draw one time
}
Lab Exercises
Lab Exercises

‣ Control the position of a shape with the mouse.
  ‣ Strive to create a more interesting relation than directly mimicking the position of the cursor.

‣ Invent three unique shapes that behave differently in relation to the mouse.
  ‣ Each shape’s behavior should change when the mouse is pressed. Relate the form of each shape to its behavior.
Lab Exercises

› Use the number keys on the keyboard to modify the movement of a line.

› Use the arrow keys to change the position of a shape within the display window.

› Animate a shape to react when the mouse is pressed and when it is released.