processing, typographie

Atelier Processing : typographie

Ressources de fonts à télécharger

- https://fonts.google.com
- https://fontawesome.com
- http://www.fontsaddict.com

Sous linux, c'est grace à la font “Noto Color Emoji”, installée par défaut, que l'on peut afficher des émojis dans diverses applications (dont le Terminal). Comme il n'est pas possible d'utiliser une font de couleur dans Processing on pourra utiliser “Noto Emoji” à la place https://fonts.google.com/noto/specimen/Noto+Emoji.

Voir le contenu d'une font

fontViewer.pde

```
/**************************************************

  Nécessite l'installation de la librairie "Drop"

**************************************************/


import java.awt.Font;
import drop.*;

SDrop drop;


PFont defaultFont;
PFont font;
Font nativeFont;
int size = 32;
int margin = 2;
int old_width, old_height;

ArrayList<MyGlyph> glyphs = new ArrayList();
int iFirst, iLast;
float yOffset = 0;
float yVel = 0;
float yMax = 0;
int selected = -1;
//ArrayList


void setup() {
  size(800, 600);
  //surface.setResizable(true);
  //fullScreen();
  old_width = width;
  old_height = height;

  printArray(PFont.list());

  // Drag & drop
  drop = new SDrop(this);

  text(' ', 0, 0);  // dirty hack to get processing's default font
  defaultFont = g.textFont;

  //loadFontFile("NotoEmoji-VariableFont_wght.ttf");
}


void loadFontFile(String fontFile) {
  font = createFont(fontFile, size);
  nativeFont = (Font) font.getNative();
  textFont(font);
  glyphs.clear();
  selected = -1;

  //int x = margin;
  //int y = font.getSize() + margin;
```

```
  int numChars = 0;
  for (int i = 0; i < 0x10ffff; i++) {
    if (nativeFont.canDisplay(i)) {
      numChars++;
      MyGlyph glyph = new MyGlyph();
      glyph.codepoint = i;
      glyph.s = new String(Character.toChars(i));
      glyph.w = textWidth(glyph.s);
      glyph.h = font.getSize();
      /*if (x + glyph.w + margin > width) {
        x = margin;
        y += font.getSize() + margin;
      }
      glyph.x = x;
      glyph.y = y;
      x += glyph.w + margin;*/
      glyphs.add(glyph);
    }
  }
  updateFont();

  if (!glyphs.isEmpty())
    yMax = max(0, glyphs.get(glyphs.size()-1).y + 4 * margin - height);

  println(numChars, "glyphs in font");
}

void updateFont() {
  int x = margin;
  int y = font.getSize() + margin;
  for (MyGlyph glyph : glyphs) {
    if (x + glyph.w + margin > width) {
        x = margin;
        y += font.getSize() + margin;
      }
      glyph.x = x;
      glyph.y = y;
      x += glyph.w + margin;
  }
}


void draw() {
  background(255);

  if (font == null) {
    fill(0);
    textSize(32);
    String mess = "Glissez-déposez une font dans la fenêtre";
    float w = textWidth(mess);
    text(mess, (width - w) * 0.5f, height/2);
    return;
  }

  yOffset = constrain(yOffset + yVel, 0, yMax);
  yVel *= 0.9;

  pushMatrix();
  translate(0, -yOffset);

  textFont(font);
  fill(0);
  iFirst = glyphs.size();
  iLast = 0;
  for (int i = 0; i < glyphs.size(); i++) {
    MyGlyph glyph = glyphs.get(i);
    if (glyph.y > yOffset && glyph.y - glyph.h < height + yOffset) {
      if (i == selected)
        fill(255, 0, 0);
      else
        fill(0);
      text(glyph.s, glyph.x, glyph.y);
      if (i < iFirst)
        iFirst = i;
      if (i > iLast)
        iLast = i;
    }
  }

  if (selected >= 0) {
    MyGlyph glyph = glyphs.get(selected);
    textFont(defaultFont);
    textSize(18);
    String text = String.format("%d | 0x%h", glyph.codepoint, glyph.codepoint);
    float tw = textWidth(text);
    float th = defaultFont.getSize();
    fill(255, 220);
    noStroke();
    rect(glyph.x + 0.5*(glyph.w - tw), glyph.y + 4, tw + 16, th + 8, 8);
    fill(0, 0, 255);
    text(text, glyph.x + 0.5*(glyph.w - tw) + 8, glyph.y + 4 + th + 4);
  }

  popMatrix();
```

```
}


void mousePressed() {
  if (font == null)
    return;

  selected = -1;

  if (keyPressed && keyCode == 16)
    println("maj");

  for (int i = iFirst; i <= iLast; i++) {
    MyGlyph glyph = glyphs.get(i);
    if (mouseX > glyph.x && mouseX < glyph.x + glyph.w &&
        mouseY + yOffset > glyph.y - glyph.h && mouseY + yOffset < glyph.y) {
      selected = i;
      println(glyph.s);
      break;
    }
  }
}

void mouseReleased() {
  if (width != old_width || height != old_height) {
    updateFont();
    println("update");
    old_width = width;
    old_height = height;
  }
}

void mouseWheel(MouseEvent event) {
  yVel += 2 * event.getCount();
}


class MyGlyph {
  public String s;
  public int codepoint;
  public int x, y;
  public float w, h;
}


void dropEvent(DropEvent theDropEvent) {
  if (theDropEvent.isFile() && (theDropEvent.toString().endsWith(".ttf") || theDropEvent.toString().endsWith(".otf"))) {
    loadFontFile(theDropEvent.toString());
  }
}
```

Effets de fonts avec Processing

textEffects.pde

```
abstract class TextEffect<T extends TextEffect> {
  protected String text;
  protected PVector position = new PVector();
  protected int delay = 0;
  protected PFont font;
  protected color col;
  protected int init = -1;
  protected boolean removable = false;

  protected TextEffect() {
    font = g.textFont;
  }

  public T delay(float seconds) {
    delay = round(seconds * 1000);
    return (T)this;
  }

  public T setText(String text) {
    this.text = text;
    reset();
    return (T)this;
  }

  public T setPosition(float x, float y) {
    this.position.set(x, y);
    reset();
    return (T)this;
  }

  public T setFont(String font, int size) {
    this.font = createFont(font, size);
    return (T)this;
  }
```

```
  public T setFont(PFont font) {
    this.font = font;
    return (T)this;
  }

  public T setColor(int r, int g, int b) {
    col = color(r, g, b);
    return (T)this;
  }

  public T setColor(color c) {
    col = c;
    return (T)this;
  }

  public abstract void update();

  public boolean isRemovable() {
    return removable;
  }

  public void reset() {
    init = -1;
    removable = false;
  }
}



class TextFade extends TextEffect<TextFade> {
  private int fadein = 2000;
  private int sustain;
  private int fadeout = 2000;

  public TextFade(String text, float x, float y) {
    super();
    this.text = text;
    this.position.set(x, y);
    sustain = text.length() * 60;
  }

  public TextFade setDuration(float fadein, float sustain, float fadeout) {
    this.fadein = round(fadein * 1000);
    this.sustain = round(sustain * 1000);
    this.fadeout = round(fadeout * 1000);
    return this;
  }

  public void update() {
    int time = millis();

    if (init < 0) {
      init = time;
    }

    if (time < init + delay) {
      return;
    } else if (time < init + delay + fadein) {
      float alpha = (time - init - delay) / float(fadein);
      alpha *= g.colorModeA;
      fill(col, alpha);
      if (font != null) textFont(font);
      text(text, position.x, position.y);
    } else if (time < init + delay + fadein + sustain) {
      fill(col);
      if (font != null) textFont(font);
      text(text, position.x, position.y);
    } else if (time < init + delay + fadein + sustain + fadeout) {
      float alpha = (time - init - delay - fadein - sustain) / float(fadeout);
      alpha = (1.0 - alpha) * g.colorModeA;
      fill(col, alpha);
      if (font != null) textFont(font);
      text(text, position.x, position.y);
    } else {
      removable = true;
    }
  }
}



class TextHScroll extends TextEffect<TextHScroll> {
  private float speed;
  private float textWidth;

  public TextHScroll(String text, float y, float speed) {
    super();
    this.text = text;
    position.y = y;
    this.speed = speed;
  }

  public void update() {
    int time = millis();
```

```
    if (init < 0) {
      init = time;
      if (speed > 0)
        position.x = -textWidth;
      else
        position.x = width;
    }

    if (time < init + delay) {
      return;
    } else if (speed > 0 && position.x < width || speed < 0 && position.x + textWidth > 0) {
      position.x += speed;
      fill(col);
      if (font != null) textFont(font);
      text(text, position.x, position.y);
    } else {
      removable = true;
    }
  }
}
```

Utilisation

```
TextEffect te;
te = new TextFade("Hello", 100, 180).setFont("FreeMono Bold", 50);

void draw() {
  te.update();
}

TextEffect te = TextHScroll(String "Bonjour", 400, 2)
```

Types d'effets:

- TextFade(String text, int x, int y);
- TextHScroll(String text, int y, float speed);

Methodes de la classe TextEffect:

- setFont(String font, int size)
- setFont(PFont font)
- delay(float seconds)
- setText(String text)
- setPosition(float x, float y)
- setColor(int r, int g, int b)
- setColor(color c)
- reset()

Article extrait de : https://lesporteslogiques.net/wiki/ - **WIKI Les Portes Logiques**
Adresse : **https://lesporteslogiques.net/wiki/ressource/code/processing/atelier_typo**
Article mis à jour: **2022/12/05 17:05**