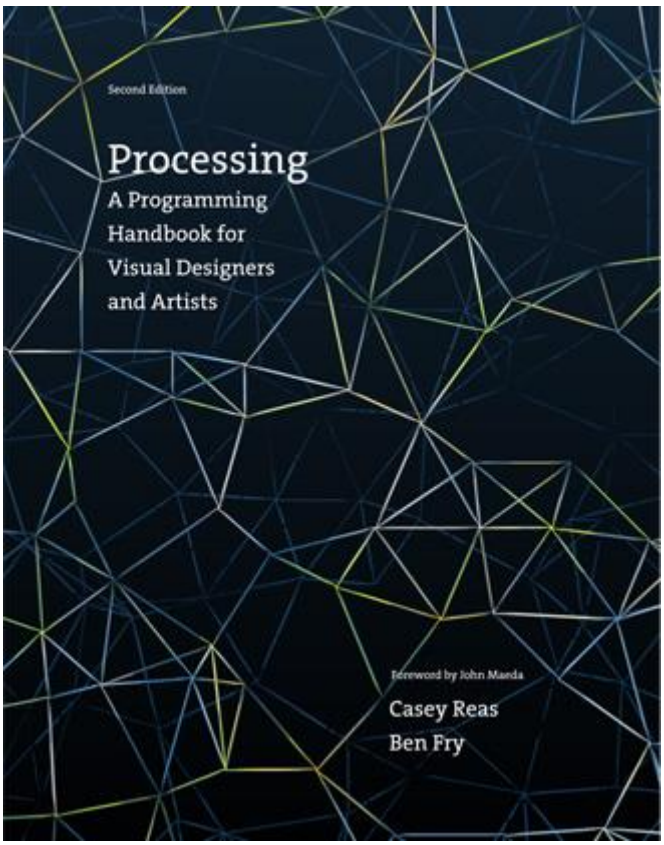
TECNOLOGICO NACIONAL DE MEXICO
INSTITUTO TECNOLOGICO DE IGUALA

JUEGO DEL ASTEROIDE

(PROCESSING)



INTEGRANTES:

ANAYELI SALVADOR CABALLERO
ROCIO ROA CRISTOBAL

PROFESOR DE LA ASIGNATURA:

CARLOS ARTURO RODRIGUEZ ROMAN

Mayo 2018 Iguala, gro

INTRODUCCION

PROCESSING

Processing es un lenguaje de programación orientado a diseñadores que no tienen necesariamente que saber programar para usarlo creado por Ben Fry y Casey Reas. Pensado especialmente para proyectos multimedia de diseñadores audiovisuales y como herramienta alternativa al software propietario, ya que se distribuye con licencia GNU GPL.

Es sorprendente la cantidad de tutoriales y vídeos que demuestran la potencia visual de este lenguaje (gráficos 2D o 3D, texturas, formas geométricas, etc...) y la facilidad de exportar los trabajos tanto como aplicaciones locales, integrada en vídeos (como los créditos de una película) o para la web. Processing está basado en java por lo que toda la sintaxis es exportada a applets Java, también se puede usar JavaScript o incluso Android para ejecutar las creaciones en aplicaciones móviles.

CARACTERÍSTICAS

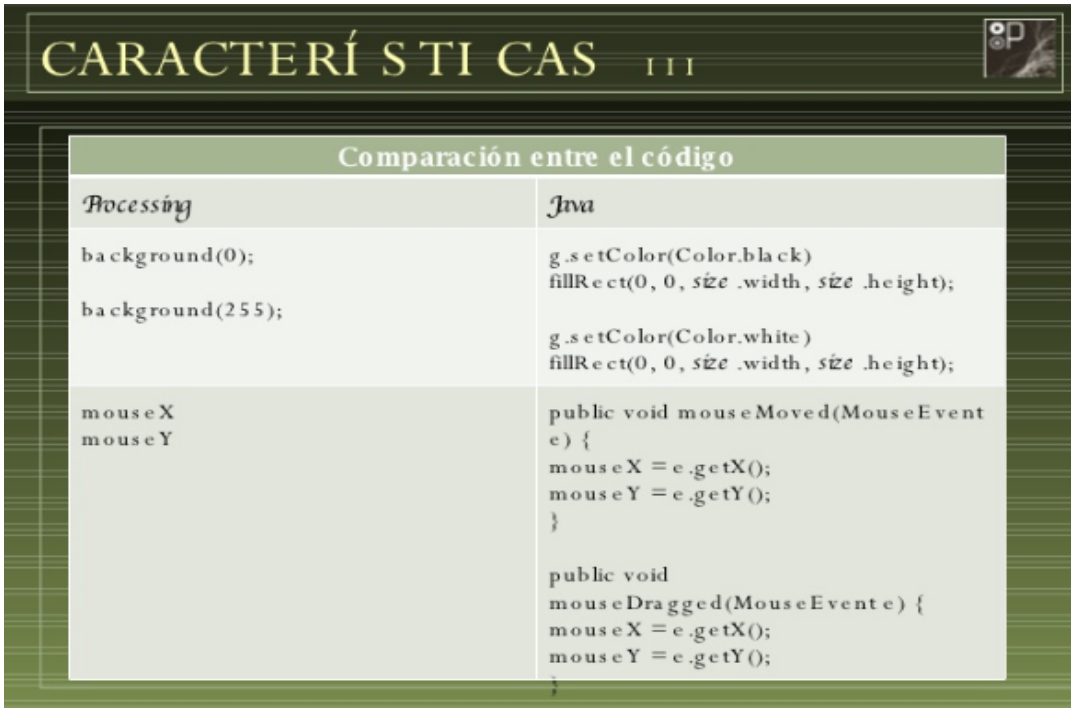
Los programas generados son traducidos a JAVA y generados como JAVA applets.

Processing se encuentra en código abierto y sus proyectos desarrollados se pueden distribuir usando la licencia “Creative Commons”.

Su ambiente de trabajo fue desarrollado en JAVA.

Utiliza una excelente librería grafica.

Los programas generados en JAVA y Processing se ejecutan mas rápido que los generados en ActionScri Es un lenguaje de programación simplificado que no requiere de un conocimiento básico de programación orientada a objetos pt o Lingo.



DESARROLLO

JUEGO DEL ASTEROIDE CODIGO

Asteroides es un videojuego de arcade que lanzó Atari en 1979 cuyo objetivo era disparar a los asteroides que aparecían en la pantalla y evitar chocar contra ellos y contra los fragmentos que desprendían al explotar.

```
//Juego asteroides


ArrayList<shot> shots = new ArrayList<shot>();

ArrayList<astroid> astroids = new ArrayList<astroid>();


// Settings - how many seconds between each new astroid (3 seconds = 3 * 60)
```

```
int astroid_rate = 2 * 30;
int astroid_count = 0;
// Size in pixel of nominal astroid
float ast_size = 10;
int ast_id = 1;
int score = 0;
float hitRate = 0;
int numShots = 0;
int ships = 3;

int pause = 0;

// Run once
  void setup () {
  frameRate(60);
  size(600, 500);
  stroke(0);
  fill(250);
}

// Called 60 times per second
void draw()
{
  int i;
//Find the angle from x=250, y=250 to the mouse
  float angle = atan2(mouseY - 250, mouseX - 250);

  if (pause==0) {
```

```
// 1 new astroid every 5 seconds (60 fps * 4 sec)
  if (astroid_count--===0) {
  astroids.add(new astroid(random(0, TWO_PI), random(0.1, 2.5), random(0.5, 4), random(-0.1, 0.1),
  random(-150, 150), random(-150, 150), ast_id++));

// Increase rate just a little
  astroid_count = astroid_rate--;
}

// Clear screen, black

  background(0);

// Go through all astroids (if any) and update their position
  for (i = 0; i<astroids.size(); i++) {
  astroid a = astroids.get(i);
  if (a.update()) {

// Remove bullet, if outside screen
  astroids.remove(i);
}

// Detect collisions with Astroids by approximating ship with 4 circles
// fill(255, 0, 100);
// ellipse(250, 230, 11, 11);
// ellipse(13*cos(angle-PI)+250, 13*sin(angle-PI)+250, 17, 17);
```

```
// ellipse(10*cos(angle)+250, 10*sin(angle)+250, 7, 7);
// ellipse(18*cos(angle)+250, 18*sin(angle)+250, 2, 2);
      if (a.coll(250, 250, 6, -1) ||
      a.coll(13*cos(angle-PI)+250, 13*sin(angle-PI)+250, 9, -1) ||
      a.coll(10*cos(angle)+250, 10*sin(angle)+250, 4, -1) ||
      a.coll(18*cos(angle)+250, 18*sin(angle)+250, 1, -1)) {
      ships--;
      pause=3*60;
      }
    }

// "pushMatrix" saves current viewpoint
    pushMatrix();
// Set 255,255 as the new 0,0
    translate(250, 250);
// Rotate screen "angle"
    rotate(angle);
    fill(250);
// Draw a triangle (the ship)
    triangle(30, 0, -20, -10, -20, 10);
// Bring back normal perspektive
    popMatrix();
  }
  else {
// Pause is larger than 0
// Clear screen, black
    background(0, 10);
```

```
  // Go through all astroids (if any) and update their position
  for (i = 0; i<astroids.size(); i++) {
   astroid a = astroids.get(i);
   a.incSpeed();
   if (a.update()) {
    // Remove bullet, if outside screen
    astroids.remove(i);
   }
  }
  if (ships == 0) {
   // Clear screen, black
   textAlign(DOWN);
   text("juego terminado", width/2, height/2);
   // 1 new astroid every 0.5 seconds (60 fps * 0.5 sec)
   // To make something happen while waiting
   if (astroid_count--==0) {
    astroids.add(new astroid(random(0, TWO_PI), random(0.1, 2.0), random(0.5, 4),
random(-0.1, 0.1),
      random(-150, 150), random(-150, 150), ast_id++));
    // Increase rate just a little
    astroid_count = 30;
   }
   if (keyPressed == true) {
    score = 0;
    numShots = 0;
    ships = 3;
    astroid_rate = 3 * 60;
    astroid_count = 0;
```

```
      ast_id = 1;
      astroids = new ArrayList<astroid>();
    }
   } else {
    // Wait until astroids are gone
    if (astroids.size()==0) {
     pause=0;
    }
   }
  }
  // Go through all shots (if any) and update their position
  for (i = 0; i<shots.size(); i++) {
   shot s = shots.get(i);
   if (s.update()) {
    // Remove bullet, if outside screen or if hits astroid
    shots.remove(i);
   }
  }
  textAlign(LEFT);
  text("Score   : " + score, 15, 15);
  text("Ships   : " + ships, 15, 30);
  text("Hit rate: " + int(100*score/float(numShots)) + "%", 15, 45);
}

// When left mouse button is pressed, create a new shot
void mousePressed() {
  if (pause==0) {
   // Only add shots when in action
```

```
    if (mouseButton == LEFT) {
      float angle = atan2(mouseY - 250, mouseX - 250);
      shots.add(new shot(angle, 4));
      numShots++;
    }
    if (mouseButton == RIGHT) {
      astroids.add(new astroid(random(0, TWO_PI), random(0.1, 2.0), random(0.5, 4),
random(-0.1, 0.1),
        random(-80, 80), random(-80, 80), ast_id++));
    }
  }
}

// Class definition for the shot
class shot {
  // A shot has x,y, and speed in x,y. All float for smooth movement
  float angle, speed;
  float x, y, x_speed, y_speed;

  // Constructor
  shot(float _angle, float _speed) {
    angle = _angle;
    speed = _speed;
    x_speed = speed*cos(angle);
    y_speed = speed*sin(angle);
    x = width/2+20*cos(angle);
    y = height/2+20*sin(angle);
  }
```

```
// Update position, return true when out of screen
boolean update() {
  int i;
  x = x + x_speed;
  y = y + y_speed;

  // Draw bullet
  ellipse (x, y, 3, 3);

  // Check for collisions
  // Go through all astroids (if any)
  for (i = 0; i<astroids.size(); i++) {
    astroid a = astroids.get(i);
    if (a.coll(x, y, 3, -1)) {
      score++;
      ast_id++;
      astroids.remove(i);
      //Remove bullet
      return true;
    }
  }
  // End, check if outside screen
  if (x<0 || x>width || y<0 || y>height) {
    return true;
  } else {
    return false;
  }
```

```
  }
}



// Class definition for the shot
class astroid {
  // An astroid angle, speed, size, rotation
  float angle, speed, size, rotSpeed;
  float position;
  float rotation;
  float xoff, yoff;
  float x, y;
  PShape s;  // The PShape object - Keeps the astroid shape
  float i;
  int id;



  // Constructor
  astroid(float _angle, float _speed, float _size, float _rotSpeed, float _xoff, float _yoff, int
_id) {
    angle = _angle;
    speed = _speed;
    size = _size;
    rotSpeed = _rotSpeed;
    xoff = _xoff;
    yoff = _yoff;
    id = _id;
```

```
  if (xoff<1000) {
    x = 250+500*cos(angle)+xoff;
    y = 250+500*sin(angle)+yoff;
  } else {
    x = _xoff-2000;
    y = _yoff-2000;
  }
  rotation = 0;
  // Generate the shape of the astroid - Some variations for all
  s = createShape();
  s.beginShape();
  s.fill(255, 255, 100);
  s.noStroke();
  for (i=0; i<TWO_PI; i=i+PI/(random(4, 11))) {
    s.vertex(random(ast_size*0.8, ast_size*1.2)*cos(i), random(ast_size*0.8,
ast_size*1.2)*sin(i));
  }
  s.endShape(CLOSE);
 }

 // Increases the speed. Used in the end of the game to clear screen of astroids
 void incSpeed() {
  speed = speed * 1.02;
 }

 // Update position, return true when out of screen
 boolean update() {
  int i;
```

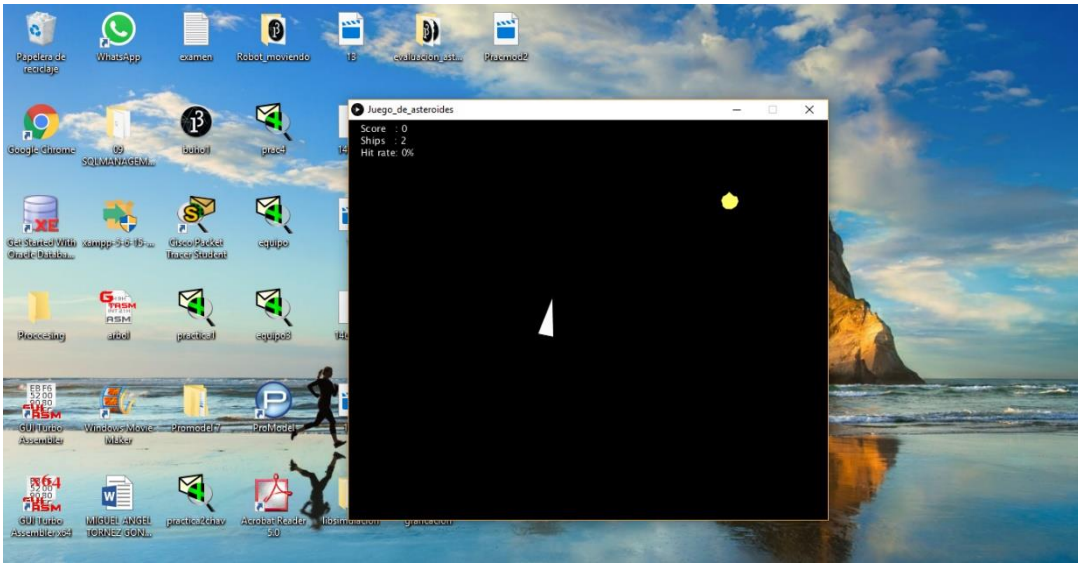
```
  x = x - cos(angle)*speed;
  y = y - sin(angle)*speed;
  rotation = rotation + rotSpeed;

  // Check for astroid vs astroid collision
  for (i = 0; i<astroids.size(); i++) {
   astroid a = astroids.get(i);
   if ((a != this) && (a.coll(x, y, ast_size*size, id))) {
    if (size > 1) {
     astroids.add(new astroid(angle-random(PI/5, PI/7), speed+random(0, speed/2), size/2,
rotSpeed, 2000+x, 2000+y, id));
     astroids.add(new astroid(angle+random(PI/5, PI/7), speed+random(0, speed/2), size/2,
rotSpeed, 2000+x, 2000+y, id));
      ast_id++;
    }
    astroids.remove(i);
   }
  }

  pushMatrix();
  // Set position as the new 0,0
  translate(x, y);
  // Rotate screen "angle"
  rotate(rotation);
  // Draw astroid
  scale(size);
  shape(s, 0, 0);
  // Bring back normal perspektive
```

```
  popMatrix();


  if (x<-300 || x>800 || y<-300 || y>800) {
    return true;
  } else {
    return false;
  }
 }


 //
 boolean coll(float _x, float _y, float _size, int _id) {
  float dist;


  dist = sqrt ((x-_x)*(x-_x) + (y-_y)*(y-_y));


  // Check if distance is shorter than astroid size and other objects size
  if ((dist<(_size+ast_size*size)) && (id!=_id)) {
    // Collision,
    if (_id>0) id = _id;
    if (size > 1) {
      // If the astroid was "large" generate two new fragments
      astroids.add(new astroid(angle-random(PI/5, PI/7), speed+random(0, speed/2), size/2,
rotSpeed, 2000+x, 2000+y, id));
      astroids.add(new astroid(angle+random(PI/5, PI/7), speed+random(0, speed/2), size/2,
rotSpeed, 2000+x, 2000+y, id));
    }
    return true;
  } else {
```

```
      return false;
    }
  }
}
```

PROGRAMA PROCESSING

EJECUCION DEL PROGRAMA



GLOSARIO

Background //función del parámetro color

col=color(random(255), random(255), random(255)); fill(col);//indica la función del color

draw(): El código dentro de esta función se repetirá indefinidamente hasta que se cierre el programa, es muy util cuando queremos dibujar constantemente algún control o gráfico. Para que trabaje la funcion draw() es necesario haber definido la funcion setup() previamente.

Float // declara el tipo de variable

Int // declaración de variables de tipo entero

if() sirve para hacer que el código escrito entre llaves ‘{}’ se ejecute sólo si se da la condición expresada entre paréntesis. Lo veremos con detalle más adelante

pushMatrix // Establecer posición como el nuevo 0,0

random // devuelve valores inesperados dentro del rango

rotate // rotación de las coordenadas

size(600, 500) //función que diseña el tamaño del lienzo

Es una herramienta que sirve como inicialización del aprendizaje de la progra mación como un medio de desarrollo de proyectos multimedia.

CONCLUSIÓN

Es una herramienta que sirve como inicialización del aprendizaje de la progra mación como un medio de desarrollo de proyectos multimedia.
Existe una gran comunidad de desarrollo gracias a que la herramienta es de c ódigo abierto y a la simplicidad de su uso. a la imaginación ya que no estable ce limites y restricciones que existen al utilizar herramientas comerciales o len guajes complicados de programación.