Materia: Graficación

Carrera: Ingeniería en Sistemas Computacionales

Alumnos:

Rafael Alejandro Santos Trujillo No. Control: 15670021

Ángel de Jesús Romero Rosas No. Control: 15670057

Documentación del proyecto “Juego Asteroides realizado en processing”

Docente: Arturo Carlos Rodríguez Román

Fecha: 29/05/2018

Contenido

Introducción:

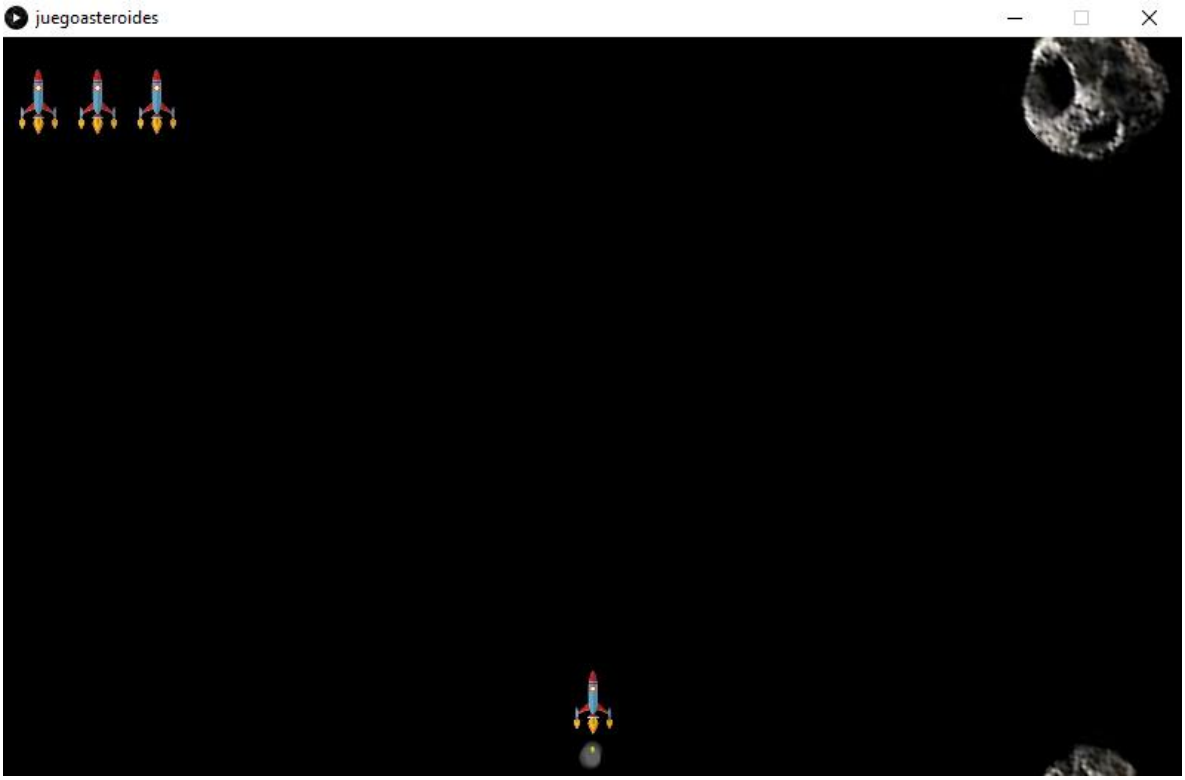
Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Fue iniciado por Ben Fry y Casey Reas, ambos miembros de Aesthetics and Computation Group del MIT Media Lab dirigido por John Maeda.

Uno de los objetivos declarados de Processing es el de actuar como herramienta para que artistas, diseñadores visuales y miembros de otras comunidades ajenos al lenguaje de la programación, aprendieran las bases de la misma a través de una muestra gráfica instantánea y visual de la información. El lenguaje de Processing se basa en Java, aunque hace uso de una sintaxis simplificada y de un modelo de programación de gráficos.

Durante el transcurso del semestre trabajamos con Processing para el desarrollo de diferentes actividades, que fueron desde pintar un pixel hasta la elaboración del juego que a continuación vamos a presentar.

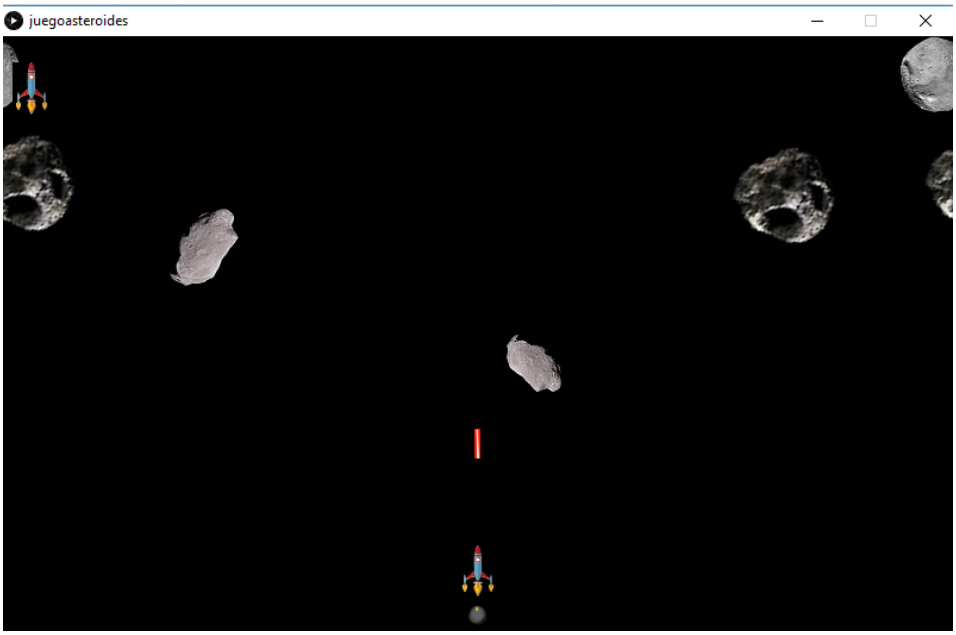
Capturas de pantalla del juego en ejecución:

Se da click en cualquier parte de la pantalla para iniciar el juego.



Comenzará el juego, en la parte superior izquierda de la pantalla se encuentran las “vidas” que tenemos, en este caso aparecen 3 naves, lo que representa que tenemos 3 “vidas”.

Para disparar el láser de la nave utilizamos la barra espaciadora.



En caso de ser impactado 3 veces por un asteroide se perderá el juego y aparecerá la siguiente pantalla:

Para movernos sobre el lienzo utilizamos las teclas UP (flecha hacia arriba), DOWN (flecha hacia abajo), LEFT (flecha hacía la izquierda) y RIGHT (flecha hacia la derecha). Estos mismos movimientos se pueden realizar con las teclas “W”, “S”, “A” y “D” respectivamente.

Código del juego:

```
Ship ship;
boolean upPressed = false;
boolean downPressed = false;
boolean aPressed = false;
boolean dPressed = false;

float shipSpeed = 2;
float rotationAngle = .2;
float bulletSpeed = 10;
int numAsteroids = 1;
```

```
int startingRadius = 50;
PImage[] asteroidPics = new PImage[3];

float bgColor = 0;

PImage naves;


ArrayList<Exhaust> exhaust;
ArrayList<Exhaust> fire;
ArrayList<Bullet> bullets;
ArrayList<Asteroid> asteroids;

PFont font;


int darkCounter;
int darkCounterLimit = 24*2;
int MAX_LIVES = 3;
int lives;
int stage = -1;
int diffCurve = 2;

void setup(){
 background(bgColor);
 size(800,500);
 font = createFont("Cambria", 32);
 asteroidPics[0] = loadImage("Asteroide-1.jpg");
 asteroidPics[1] = loadImage("Asteroide-2.jpg");
 asteroidPics[2] = loadImage("Asteroide-3.jpg");
```

```
 naves = loadImage("nave.jpg");
 frameRate(24);
 lives = 3;
 asteroids = new ArrayList<Asteroid>(0);
}


void draw(){
 if( lives >= 0 && asteroids.size()>0){
   float theta = heading2D(ship.rotation)+PI/2;
   background(0);

   ship.update(exhaust, fire);
   ship.edges();
   ship.render();
   if(ship.checkCollision(asteroids)){
    lives--;
    ship = new Ship();
   }

   if(aPressed){
     rotate2D(ship.rotation,-rotationAngle);
   }
   if(dPressed){
     rotate2D(ship.rotation, rotationAngle);
   }
   if(upPressed){
     ship.acceleration = new PVector(0,shipSpeed);
     rotate2D(ship.acceleration, theta);
   }
```

```
for(Exhaust e: exhaust){
 e.update();
 e.render();
}

for(Exhaust e: fire){
 e.update();
 e.render();
}

for(int i = 0; i < bullets.size(); i++){
 bullets.get(i).edges();
 if(bullets.get(i).update()){
   bullets.remove(i);
   i--;
 }
 if(i < 0){
  break;
 }
 bullets.get(i).render();
 if(bullets.get(i).checkCollision(asteroids)){
   bullets.remove(i);
   i--;
 }
}

 while(exhaust.size() > 20){
 exhaust.remove(0);
}
```

```
  while(fire.size()>3){
   fire.remove(0);
  }

   while(bullets.size() > 30){
   bullets.remove(0);
  }

  for(Asteroid a : asteroids){
   a.update();
   a.edges();
   a.render();
  }


  for(int i = 0; i < lives; i++){
    image(naves,40*i + 10,ship.r*1.5,2*ship.r,3*ship.r);
  }
} else if(lives < 0){
  if(darkCounter < darkCounterLimit){
   background(0);
   darkCounter++;
   for(Asteroid a : asteroids){
     a.update();
     a.edges();
     a.render();
    }
   fill(0, 255-(darkCounterLimit-darkCounter)*3);
   rect(0,0,width,height);
```

```
  } else {
    background(0);
    for(Asteroid a : asteroids){
     a.update();
     a.edges();
     a.render();
    }
    image(naves,width/2 - 5 * ship.r,height/2-7.5*ship.r,10*ship.r,15*ship.r);
    textFont(font, 33);
    fill(0, 200);
    text("HAS PERDIDO XD", width/2-80-2, height*.75-1);
    textFont(font, 32);
    fill(255);
    text("HAS PERDIDO XD", width/2-80, height*.75);

    textFont(font, 16);
    fill(0, 200);
    text("Da click en el mouse para jugar de nuevo", width/2-80-2, height*.9-1);
    textFont(font, 15);
    fill(255);
    text("Da click en el mouse para jugar de nuevo", width/2-80, height*.9);
  }
} else {
    background(0);
    ship = new Ship();
    ship.render();

    textFont(font, 32);
    fill(255);
    if(stage > -1){
```

```
        text("Escenario " + (stage + 1) + " Completo", width/2-120, height/2);
    } else {
        text("Asteroides", width/2-80, height/2);
    }

    textFont(font, 15);
    fill(255);
    text("Da click en el mouse para jugar " + (stage + 2), width/2-100, height*.75);


 }
}

void mousePressed(){
  if(lives < 0){
   stage = -1;
   lives = 3;
   asteroids = new ArrayList<Asteroid>(0);
  } else  if (asteroids.size()==0){
   stage++;
   reset();
  }
}


void reset(){
 ship  = new Ship();
 exhaust = new ArrayList<Exhaust>();
 fire = new ArrayList<Exhaust>();
 bullets = new ArrayList<Bullet>();
```

```
 asteroids = new ArrayList<Asteroid>();
 for(int i = 0; i <numAsteroids + diffCurve*stage; i++){
  PVector position = new PVector((int)(Math.random()*width),
(int)(Math.random()*height-100));
  asteroids.add(new Asteroid(position, startingRadius, asteroidPics, stage));
 }
 darkCounter = 0;
}

void fireBullet(){
  PVector pos = new PVector(0, ship.r*2);
  rotate2D(pos,heading2D(ship.rotation) + PI/2);
  pos.add(ship.position);
  PVector vel  = new PVector(0, bulletSpeed);
  rotate2D(vel, heading2D(ship.rotation) + PI/2);
  bullets.add(new Bullet(pos, vel));
}

void keyPressed(){
  if(key==CODED){
   if(keyCode==UP){
      upPressed=true;
   } else if(keyCode==DOWN){
      downPressed=true;
   } else if(keyCode == LEFT){
    aPressed = true;
   }else  if(keyCode==RIGHT){
    dPressed = true;
   }
  }
```

```
  if(key == 'a'){
    aPressed = true;
  }
  if(key=='d'){
   dPressed = true;
  }
  if(key=='w'){
    upPressed=true;
  }
  if(key=='s'){
   downPressed=true;
  }
}

void keyReleased(){
  if(key==CODED){
   if(keyCode==UP){
     upPressed=false;
    ship.acceleration = new PVector(0,0);
   } else if(keyCode==DOWN){
     downPressed=false;
     ship.acceleration = new PVector(0,0);
   } else if(keyCode==LEFT){
     aPressed = false;
   } else  if(keyCode==RIGHT){
    dPressed = false;
   }
  }
  if(key=='a'){
   aPressed = false;
```

```
  }
  if(key=='d'){
   dPressed = false;
  }
  if(key=='w'){
    upPressed=false;
    ship.acceleration = new PVector(0,0);
  }
  if(key=='s'){
   downPressed=false;
   ship.acceleration = new PVector(0,0);
  }
  if(key == ' '){
   fireBullet();
  }
}


float heading2D(PVector pvect){
   return (float)(Math.atan2(pvect.y, pvect.x));
}

void rotate2D(PVector v, float theta) {
  float xTemp = v.x;
  v.x = v.x*cos(theta) - v.y*sin(theta);
  v.y = xTemp*sin(theta) + v.y*cos(theta);
}


class Asteroid{
```

```
float radius;
float omegaLimit = .05;
PVector position;
PVector velocity;
PVector rotation;
float spin;
int col = 100;
PImage pics[];
PImage pic;
int stage;
float dampening = 1;


public Asteroid(PVector pos, float radius_, PImage[] pics_, int stage_){
  radius  = radius_;
  stage = stage_;
  position = pos;
  float angle = random(2 * PI);
  velocity = new PVector(cos(angle), sin(angle));
  velocity.mult((50*50)/(radius*radius));
  velocity.mult(sqrt(stage  + 2));
  velocity.mult(dampening);
  angle = random(2 * PI);
  rotation = new PVector(cos(angle), sin(angle));
  spin = (float)(Math.random()*omegaLimit-omegaLimit/2);
  int rnd = (int)(Math.random()*3);
  pics = pics_;
  pic = pics[rnd];
}
```

```
void breakUp(ArrayList<Asteroid> asteroids){
 if(radius <= 30){
  asteroids.remove(this);
 } else if (radius < 33){
    for(int i = 0; i < 2; i++){
     float angle = random(2*PI);
     PVector rand = new PVector(radius*sin(angle), radius*cos(angle));
     rand.add(position);
     asteroids.add(new Asteroid(rand, radius*.8, pics, stage));
   }
   asteroids.remove(this);
 } else {
   for(int i = 0; i < 3; i++){
     float angle = random(2*PI);
     PVector rand = new PVector(radius*sin(angle), radius*cos(angle));
     rand.add(position);
     asteroids.add(new Asteroid(rand, radius*.8, pics, stage));
   }
   asteroids.remove(this);
 }
}

void update(){
  position.add(velocity);
  rotate2D(rotation, spin);
}
void render(){
  fill(col);
  circ(position.x, position.y);
  if (position.x < radius){
```

```
    circ(position.x + width, position.y);
  } else if (position.x > width-radius) {
    circ( position.x-width, position.y);
  }
  if (position.y < radius) {
    circ(position.x, position.y + height);
  } else if (position.y > height-radius){
    circ(position.x, position.y-height);
  }
}

void edges(){
 if (position.x < 0){
    position.x = width;
  }
  if (position.y < 0) {
    position.y = height;
  }
  if (position.x > width) {
    position.x = 0;
  }
  if (position.y > height){
    position.y = 0;
  }
}


void circ(float x, float y){
 pushMatrix();
 translate(x,y);
```

```
  rotate(heading2D(rotation)+PI/2);
  image(pic, -radius,-radius,radius*2, radius*2);
  popMatrix();
 }


float heading2D(PVector pvect){
   return (float)(Math.atan2(pvect.y, pvect.x));
}

void rotate2D(PVector v, float theta) {
  float xTemp = v.x;
  v.x = v.x*cos(theta) - v.y*sin(theta);
  v.y = xTemp*sin(theta) + v.y*cos(theta);
}

}

class Bullet{
 PVector position;
 PVector velocity;
 int radius = 5;
 int counter = 0;
 int timeOut = 24 * 2;
 float alpha;
 PImage img = loadImage("laser.png");

 public Bullet(PVector pos, PVector vel){
  position = pos;
  velocity = vel;
```

```
 alpha = 255;
}

void edges(){
 if (position.x < 0){
     position.x = width;
   }
   if (position.y < 0) {
     position.y = height;
   }
   if (position.x > width) {
     position.x = 0;
   }
   if (position.y > height){
     position.y = 0;
   }
}

boolean checkCollision(ArrayList<Asteroid> asteroids){
  for(Asteroid a : asteroids){
    PVector dist = PVector.sub(position, a.position);
    if(dist.mag() < a.radius){
     a.breakUp(asteroids);
     return true;
    }
  }
  return false;
}

boolean update(){
```

```
   alpha *= .9;
  counter++;
  if(counter>=timeOut){
    return true;
  }
  position.add(velocity);
  return false;
 }

 void render(){
  fill(255);
  pushMatrix();
  translate(position.x, position.y);
  rotate(heading2D(velocity)+PI/2);
  //ellipse(0,0, radius, radius*5);
  image(img, -radius/2, -2*radius, radius, radius*5);
  popMatrix();
 }

float heading2D(PVector pvect){
   return (float)(Math.atan2(pvect.y, pvect.x));
}

}
class Exhaust{
  PVector position;
  PVector velocity;
  float diameter;
  color hugh;
```

```
  public Exhaust(PVector pos, PVector vel, color col, int rad){
   position = pos;
   velocity = vel;
   diameter = (float)(Math.random()*rad);
   hugh = col;
  }

  void render(){
   noStroke();
   fill(hugh);
   ellipse(position.x, position.y, diameter, diameter);

  }

  void update(){
   position.add(velocity);
   velocity.mult(.9);
  }
}

class Ship{
 PVector position;
 PVector velocity;
 PVector acceleration;
 PVector rotation;
 float drag = .9;
 float r = 15;
 PImage img = loadImage("nave.jpg");

 public Ship(){
```

```
 position = new PVector(width/2, height-50);
 acceleration = new PVector(0,0);
 velocity = new PVector(0,0);
 rotation = new PVector(0,1);
}

void update(ArrayList<Exhaust> exhaust, ArrayList<Exhaust> fire){
  PVector below = new PVector(0, -2*r);
  rotate2D(below, heading2D(rotation)+PI/2);
  below.add(position);
  color grey = color(100, 75);

  int  exhaustVolume = (int)(velocity.mag())+1;
  for(int i = 0; i  <exhaustVolume; i++){
    float angle = (float)(Math.random()*.5-.25);
    angle += heading2D(rotation);
    PVector outDir = new PVector(cos(angle), sin(angle));
    exhaust.add(new Exhaust(below, outDir, grey, 15));
  }
  for(int i = 0; i  <1; i++){
    float angle = (float)(Math.random()*.5-.25);
    angle += heading2D(rotation);
    PVector outDir = new PVector(cos(angle), sin(angle));
    outDir.y = 0;
    below.add(outDir);
    below.y-=.5;
    color red = color((int)(200 + Math.random()*55),(int)( 150+Math.random()*105),
50, 250);
    fire.add(new Exhaust(below,outDir, red, 5));
  }
```

```
  velocity.add(acceleration);
  velocity.mult(drag);
  velocity.limit(5);
  position.add(velocity);

}

void edges(){
 if (position.x < r){
     position.x = width-r;
   }
   if (position.y < r) {
     position.y = height-r;
   }
   if (position.x > width-r) {
     position.x = r;
   }
   if (position.y > height-r){
     position.y = r;
   }
}
boolean checkCollision(ArrayList<Asteroid> asteroids){
  for(Asteroid a : asteroids){
   PVector dist = PVector.sub(a.position, position);
   if(dist.mag() < a.radius + r/2){
    a.breakUp(asteroids);
    return true;
   }
  }
  return false;
```

```
 }
 void render(){
   float theta = heading2D(rotation)  + PI/2;
   theta += PI;
   pushMatrix();
   translate(position.x, position.y);
   rotate(theta);
   fill(0);

   image(img,-r,-r*1.5,2*r,3*r);
   popMatrix();
 }

float heading2D(PVector pvect){
   return (float)(Math.atan2(pvect.y, pvect.x));
}
 void rotate2D(PVector v, float theta) {
  float xTemp = v.x;
  v.x = v.x*cos(theta) - v.y*sin(theta);
  v.y = xTemp*sin(theta) + v.y*cos(theta);
}
}
```

Explicación de las funciones utilizadas en el código:

Nombre: PImage
Ejemplo:

```
PImage naves;
void setup(){
naves = loadImage("nave.jpg");
}
void draw(){
for(int i = 0; i < lives; i++){
    image(naves,40*i + 10,ship.r*1.5,2*ship.r,3*ship.r);
}
```

Descripción: Tipo de datos para almacenar imágenes. El procesamiento puede mostrar imágenes .gif, .jpg, .tga y .png. Las imágenes pueden mostrarse en espacios 2D y 3D. Antes de usar una imagen, debe cargarse con la función loadImage (). La clase PImage contiene campos para el ancho y alto de la imagen, así como una matriz llamada pixels [] que contiene los valores para cada píxel en la imagen.

Para que la imagen sea cargada de manera correcta, la tenemos que agregar al sketch, esto lo hacemos dando click en la opción Sketch de la barra de tarea de processing, después damos click en “Añadir archivo” y seleccionamos la imagen que vamos a cargar.

Nombre: PFont
Ejemplo:

```
PFont font;
void setup(){
font = createFont("Cambria", 32);
}
```

```
Void draw(){
textFont(font, 16);
    fill(0, 200);
    text("Da click en el mouse para jugar de nuevo", width/2-80-2, height*.9-1);
    textFont(font, 15);
    fill(255);
    text("Da click en el mouse para jugar de nuevo", width/2-80, height*.9);
  }
```

Descripción: PFont es la clase de fuente para Processing. Para crear una fuente para usar con Procesamiento. Procesamiento muestra fuentes utilizando el formato de fuente .vlw, que utiliza imágenes para cada letra, en lugar de definirlas a través de datos vectoriales. La función loadFont () construye una nueva fuente y textFont () hace que una fuente esté activa. Con la función createFont () se puede convertir dinámicamente las fuentes en un formato para usar con Processing.

Nombre: Background()
Ejemplo:

```
void setup(){
 background(0);
}
```

Descripción: La función background () establece el color utilizado para el fondo de la ventana de procesamiento.

Nombre: size()
Ejemplo:

```
void setup(){
size(800,500);
}
```

Descripción: Define la dimensión del ancho y alto de la ventana de visualización en unidades de píxeles.

Nombre: frameRate()
Ejemplo:

```
void setup(){
frameRate(24);
}
```

Descripción: Especifica la cantidad de cuadros que se mostrarán cada segundo. Por ejemplo, la función llamada frameRate (24) intentará actualizar 24 veces por segundo. Si el procesador no es lo suficientemente rápido para mantener la velocidad especificada, la velocidad de cuadros no se logrará.

Nombre: void draw()
Ejemplo:

Aunque está vacío aquí, draw () es necesario para el boceto puede procesar eventos de entrada del usuario (el mouse presiona en este caso).

```
 void draw () {}
 void mousePressed () {
   línea (mouseX, 10, mouseX, 90);
 }
```

Descripción: La función draw () ejecuta continuamente las líneas de código contenidas dentro de su bloque hasta que se detiene el programa o se invoca noLoop (). draw () se llama automáticamente y nunca se debe llamar explícitamente.

Nombre: fill()
Ejemplo:

```
fill(153);
```

Descripción: Establece el color utilizado para rellenar formas. Por ejemplo, si ejecuta relleno (204, 102, 0), todas las formas siguientes se llenarán de naranja.

Nombre: if()
Ejemplo:

```
if( lives >= 0 && asteroids.size()>0){
```

```
  float theta = heading2D(ship.rotation)+PI/2;
  background(0);
}
```

Descripción:

Permite que el programa tome una decisión sobre qué código ejecutar. Si la prueba se evalúa como verdadera, las instrucciones incluidas en el bloque se ejecutan y si la prueba se evalúa como falsa, las declaraciones no se ejecutan.

Nombre: for()
Ejemplo: for (int i = 0; i <40; i = i + 1) {línea (30, i, 80, i); }

Descripción: Controla una secuencia de repeticiones. Una estructura básica tiene tres partes: init , test y update . Cada parte debe estar separada por un punto y coma (;). El ciclo continúa hasta que la prueba se evalúa como falsa.

Nombre: while()
Ejemplo: int i = 0; while (i <80) {línea (30, i, 80, i); i = i + 5; }

Descripción: Controla una secuencia de repeticiones. La estructura while ejecuta una serie de instrucciones continuamente mientras que la expresión es verdadera. La expresión debe actualizarse durante las repeticiones o el programa nunca "saldrá" de while.

Nombre: mousePressed()
Ejemplo:

```
void mousePressed(){
  if(lives < 0){
    stage = -1;
    lives = 3;
    asteroids = new ArrayList<Asteroid>(0);
  } else  if (asteroids.size()==0){
    stage++;
    reset();
  }
```

}

Descripción: La función mousePressed () se invoca una vez cada vez que se presiona un botón del mouse. La variable mouseButton (ver la entrada de referencia relacionada) se puede usar para determinar qué botón se ha presionado.

Los eventos de mouse y teclado solo funcionan cuando un programa tiene draw (). Sin draw (), el código solo se ejecuta una vez y luego deja de escuchar eventos.

Nombre: PVector
Ejemplo:

```
void fireBullet(){
  PVector pos = new PVector(0, ship.r*2);
  rotate2D(pos,heading2D(ship.rotation) + PI/2);
  pos.add(ship.position);
  PVector vel  = new PVector(0, bulletSpeed);
  rotate2D(vel, heading2D(ship.rotation) + PI/2);
  bullets.add(new Bullet(pos, vel));
}
```

Descripción: Una clase para describir un vector de dos o tres dimensiones, específicamente un vector euclidiano (también conocido como geométrico). Un vector es una entidad que tiene tanto magnitud como dirección. Sin embargo, el tipo de datos almacena los componentes del vector (x, y para 2D y x, y, z para 3D).

Nombre: keyPressed()
Ejemplo:

```
void keyPressed(){
  if(key==CODED){
   if(keyCode==UP){
      upPressed=true;
   } else if(keyCode==DOWN){
      downPressed=true;
   } else if(keyCode == LEFT){
```

```
    aPressed = true;
   }else  if(keyCode==RIGHT){
    dPressed = true;
   }
  }
  if(key == 'a'){
    aPressed = true;
  }
  if(key=='d'){
   dPressed = true;
  }
  if(key=='w'){
    upPressed=true;
  }
  if(key=='s'){
   downPressed=true;
  }
}
```

Descripción: El código puesto como ejemplo es el utilizado en el juego para mandar la señal de cuando se presiona una de las teclas que hará que la nave se mueva.

La función keyPressed () se invoca una vez cada vez que se presiona una tecla.

Nombre: keyReleased()
Ejemplo:

```
void keyReleased(){
  if(key==CODED){
   if(keyCode==UP){
     upPressed=false;
    ship.acceleration = new PVector(0,0);
   } else if(keyCode==DOWN){
```

```
    downPressed=false;
    ship.acceleration = new PVector(0,0);
  } else if(keyCode==LEFT){
    aPressed = false;
  } else  if(keyCode==RIGHT){
   dPressed = false;
  }
 }
 if(key=='a'){
  aPressed = false;
 }
 if(key=='d'){
  dPressed = false;
 }
 if(key=='w'){
   upPressed=false;
   ship.acceleration = new PVector(0,0);
 }
 if(key=='s'){
  downPressed=false;
  ship.acceleration = new PVector(0,0);
 }
 if(key == ' '){
  fireBullet();
 }
}
```

Descripción: La función keyReleased () se invoca una vez cada vez que se suelta una tecla.

Nombre: pushMatrix
Ejemplo:

```
void circ(float x, float y){
  pushMatrix();
  translate(x,y);
  rotate(heading2D(rotation)+PI/2);
  image(pic, -radius,-radius,radius*2, radius*2);
  popMatrix();
 }
```

Descripción: La función pushMatrix () guarda el sistema de coordenadas actual en la pila y popMatrix () restaura el sistema de coordenadas anterior. pushMatrix () y popMatrix () se usan en conjunción con las otras funciones de transformación y se pueden incorporar para controlar el alcance de las transformaciones.

Nombre: popMatrix()
Ejemplo:

```
void circ(float x, float y){
  pushMatrix();
  translate(x,y);
  rotate(heading2D(rotation)+PI/2);
  image(pic, -radius,-radius,radius*2, radius*2);
  popMatrix();
 }
```

Descripción: La función pushMatrix () guarda el sistema de coordenadas actual en la pila y popMatrix () restaura el sistema de coordenadas anterior. pushMatrix () y popMatrix () se usan en conjunción con las otras funciones de transformación y se pueden incorporar para controlar el alcance de las transformaciones.

Nombre: translate()
Ejemplo:

```
void circ(float x, float y){
```

```
 pushMatrix();
 translate(x,y);
 rotate(heading2D(rotation)+PI/2);
 image(pic, -radius,-radius,radius*2, radius*2);
 popMatrix();
}
```

Descripción: Especifica una cantidad para desplazar objetos dentro de la ventana de visualización. El parámetro “x” especifica la traducción izquierda / derecha, el parámetro “y” especifica la conversión ascendente / descendente, y el parámetro z especifica las traducciones hacia / desde la pantalla.

Nombre: rotate()
Ejemplo:

```
void circ(float x, float y){
 pushMatrix();
 translate(x,y);
 rotate(heading2D(rotation)+PI/2);
 image(pic, -radius,-radius,radius*2, radius*2);
 popMatrix();
}
```

Descripción: Gira la cantidad especificada por el parámetro de ángulo . Los ángulos se deben especificar en radianes (valores de 0 a TWO_PI), o se pueden convertir de grados a radianes con la función radianes () .

Conclusiones:

Conclusión de Rafael Alejandro Santos Trujillo:

A lo largo del semestre trabajamos con processing y vimos varios métodos y funciones, las cuales nos permitían crear trazos como círculos, rectángulos, líneas y otros tipos de formas, también vimos la manera de implementar clases en nuestros proyectos lo cual nos hacía tener un mejor código. Con la elaboración de este proyecto que consistió en replicar el juego de asteroides utilizando processing, pudimos dar un gran repaso a la mayoría de las funciones que ya conocíamos, tales como keyPressed, mousePressed, rotate, translate, entre otras, además pudimos aprender por medio de la replicación de código como es que funcionan otro tipo de métodos o funciones que desconocíamos. Sin lugar a duda realizar el juego de Asteroides me permitió entender un poco más a fondo lo increíble que puede llegar a ser la programación y como con tan poco se pueden hacer grandes cosas.

Conclusión de Ángel de Jesús Romero Rosas:

En este proyecto que realizamos fue la elaboración del juego de asteroides con las herramientas y funciones que habíamos visto en la clase de graficación tales como background que se utiliza para el fondo del lienzo que genera processing, también se utilizó la función keyPressed que la utilizamos para el movimiento de la nave, la función mousePressed que la utilizamos para los asteroides, para cargar las imágenes utilizamos PImage que es la que permite subir las imágenes al archivo del programa del juego. También utilizamos la función rotate() que es para hacer rotar las imágenes de los asteroides, utilizamos el ciclo for() para generar algunas funciones del juego. Otra de las cosas que utilizamos y que ya habíamos visto fue como crear clases dentro del programa, en si este trabajo fue difícil para su elaboración del juego, aunque la mayoría de las funciones ya las habíamos visto anteriormente.