****

**TECNOLOGICO NACIONAL DE MEXICO.**

**INSTITUTO TECNOLOGICO DE IGUALA.**

**DOCENTE: CARLOS ARTURO RODRIGUEZ ROMAN.**

**CARRERA: INGENIERIA EN SISTEMAS COMPUTACIONALES.**

**MATERIA: GRAFICACIÓN**

**-DOCUMENTACIÓN-**

**INTEGRANTES:**

**\*JOSE ADRIAN CERON LOPEZ. 14670104**

**\*KAREN LORELY REYNA VARGAS. 14670103**

**GRUPO: “B” AULA: E-3**

**HORARIO: 6:00 PM A 8:00PM**

**LUNES Y MARTES.**

INTRODUCCION:

Los gráficos por computadora se han convertido en una potente herramienta para la producción rápida y económica de imágenes. Los gráficos por computadora son una herramienta que representa una ventaja y que puede ser aplicada en diversos campos tales como diseño industrial, diseño ambiental, diseño y fotografía, audio y video, diseño web, diseño de videojuegos etc. La graficación es muy compleja y muy entretenida. Es un tipo de representación de datos en donde se utilizan las líneas, puntos, vectores y símbolos, además utilizas mucho tu creatividad y tus habilidades para programar.

En este documento explicaremos paso a paso de cómo realizar un programa en donde con códigos e imágenes nos de cómo resultado un juego de una nave espacial con asteroides, explicaremos también los códigos aprendidos en clase y el resultado.

El programa utilizado para su elaboración es Processing el cual 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 iterativos de diseño digital.

**CODIGO DEL JUEGO DE NAVE.**

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("asteroide1.jpg");

asteroidPics[1] = loadImage("asteroide2.jpg");

asteroidPics[2] = loadImage("asteroide3.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("GAME OVER", width/2-80-2, height\*.75-1);

textFont(font, 32);

fill(255);

text("GAME OVER", width/2-80, height\*.75);

textFont(font, 16);

fill(0, 200);

text(" CLICK PARA EMPEZAR UN NUEVO JUEGO", width/2-80-2, height\*.9-1);

textFont(font, 15);

fill(255);

text("CLICK PARA UN EMPEZAR NUEVO JUEGO", width/2-80, height\*.9);

}

} else {

background(0);

ship = new Ship();

ship.render();

textFont(font, 32);

fill(255);

if(stage > -1){

text("Mision " + (stage + 1) + " Finalizada", width/2-120, height/2);

} else {

text("PILOTO DE COMBATE", width/2-80, height/2);

}

textFont(font, 15);

fill(255);

text("PRECIONA CLICK PARA EMPEZAR " + (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("lacer.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);

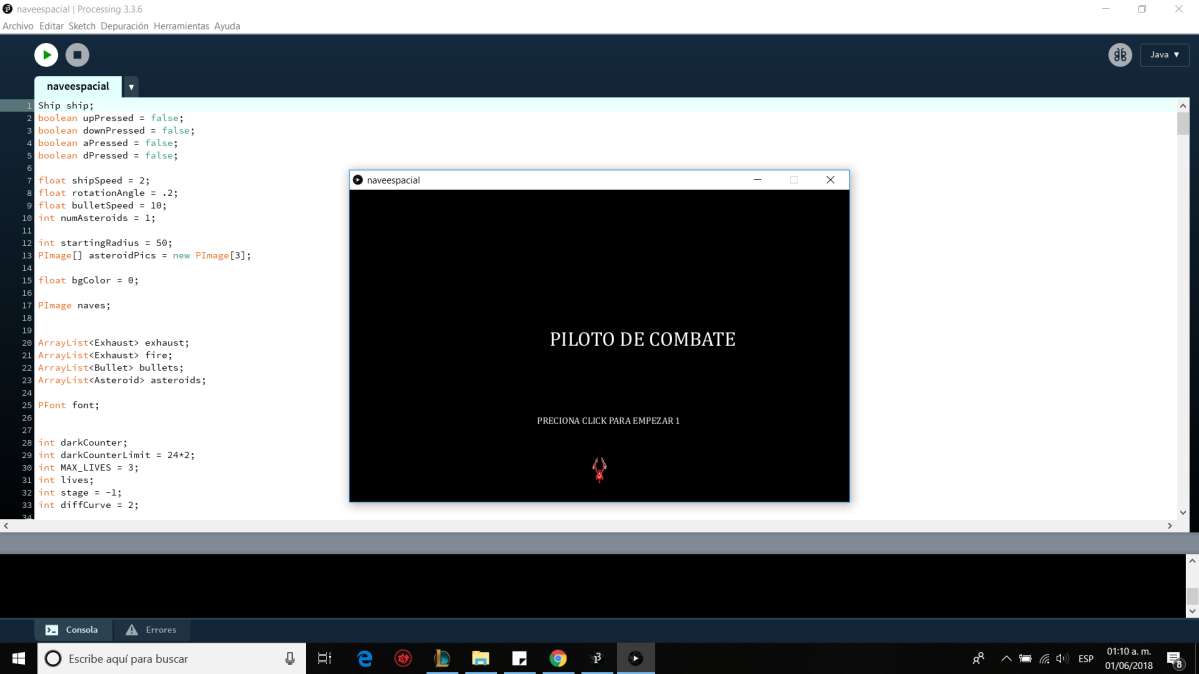
v.y = xTemp\*sin(theta) + v.y\*cos(theta);

}

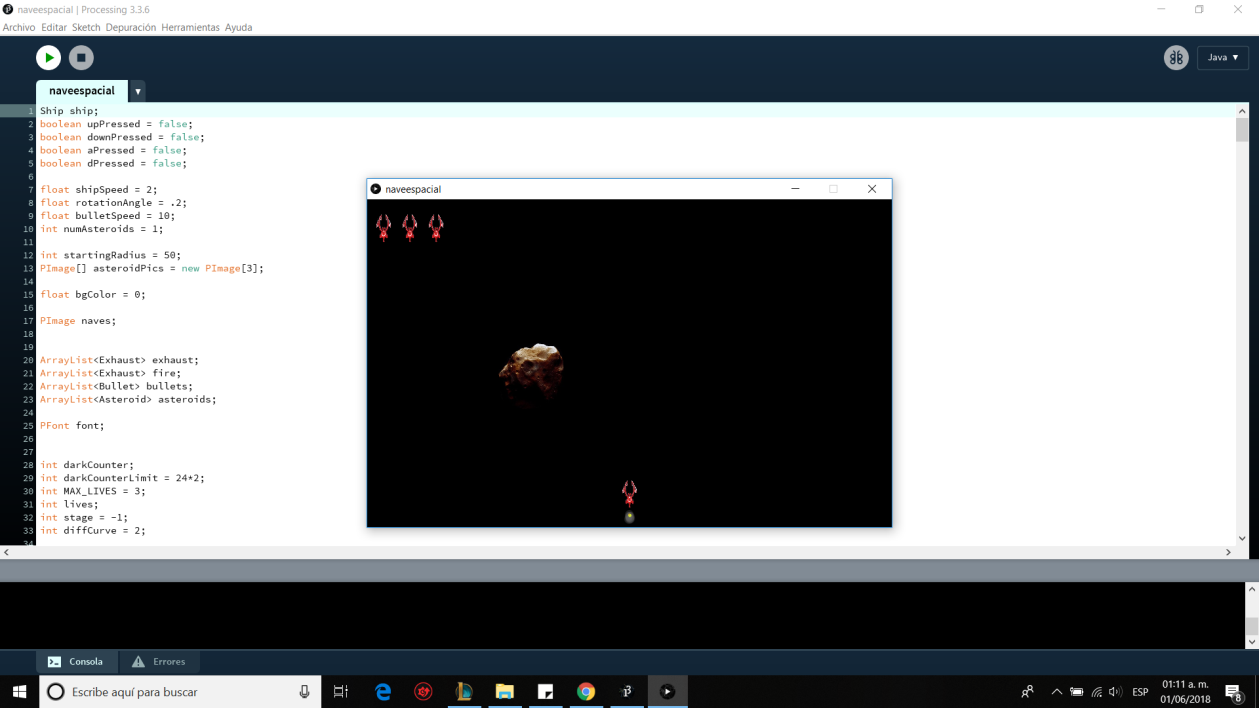
}

**RESULTADO:**

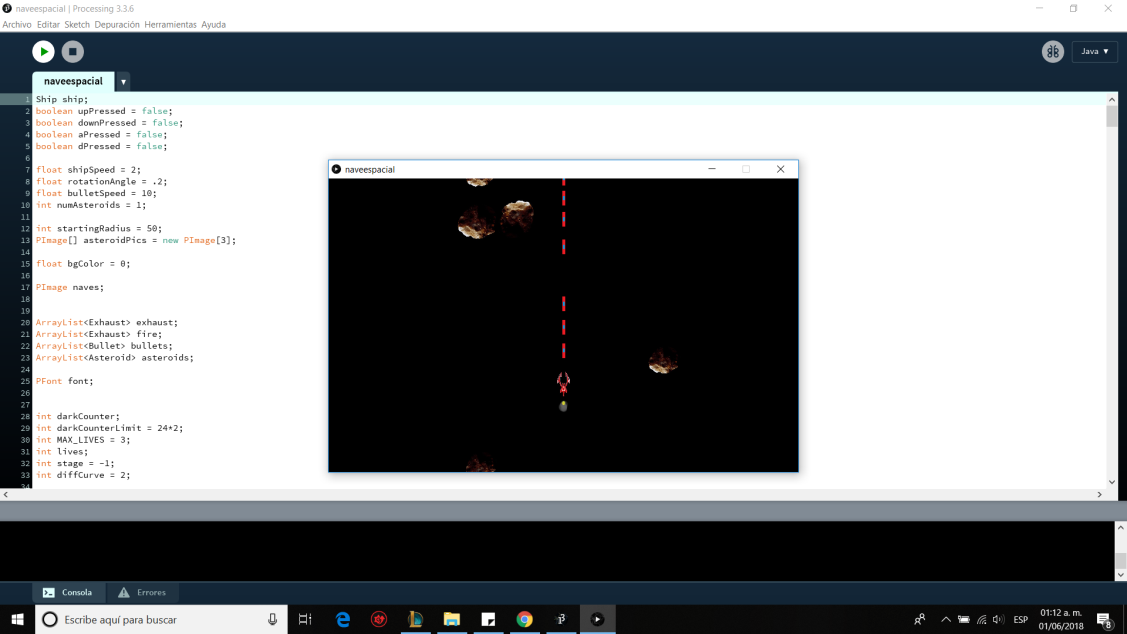
Al ejecutar el programa nos abre esta ventana en la cual podemos observar que el juego empezara al darle click a la pantalla.



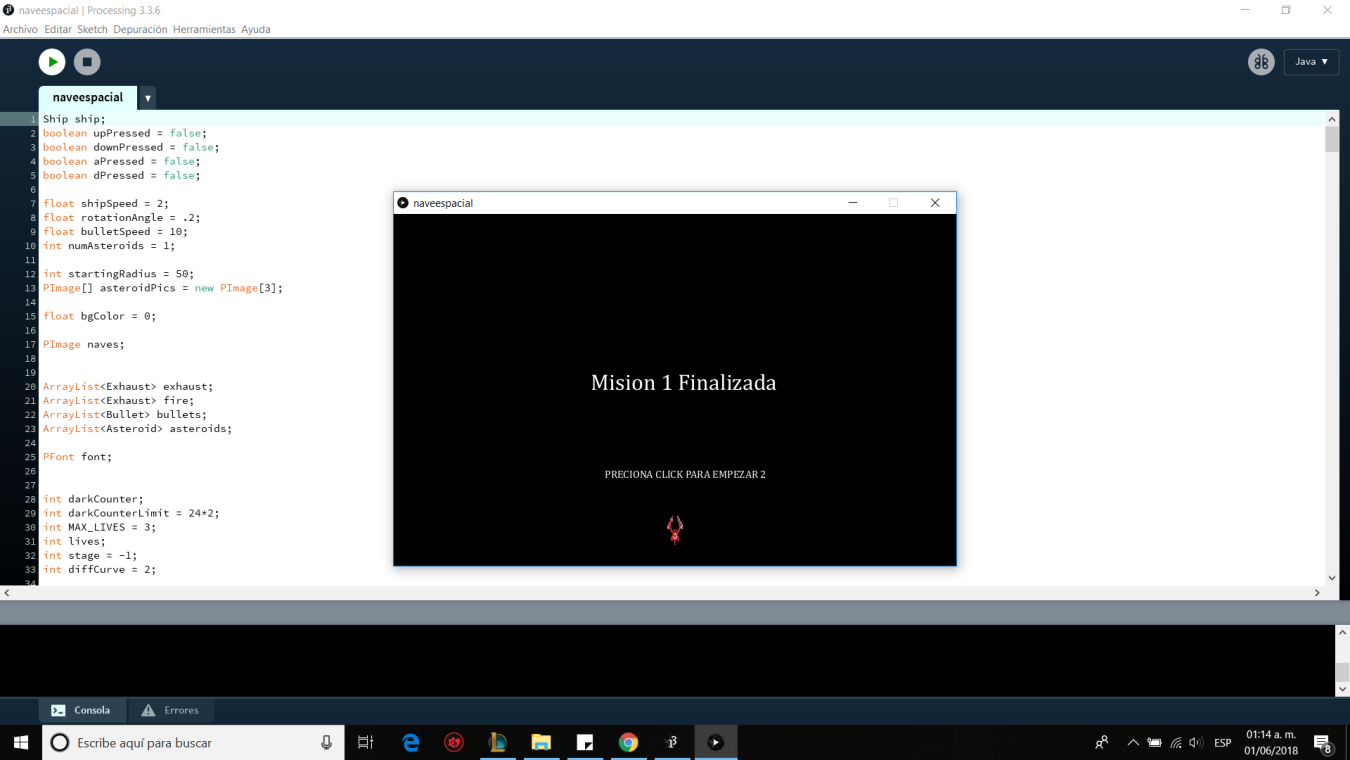
Después de dar click el juego ah comenzado, podemos observar en la parte izquierda las vidas que tenemos en este caso son 3, también se observa la nave y los asteroides.



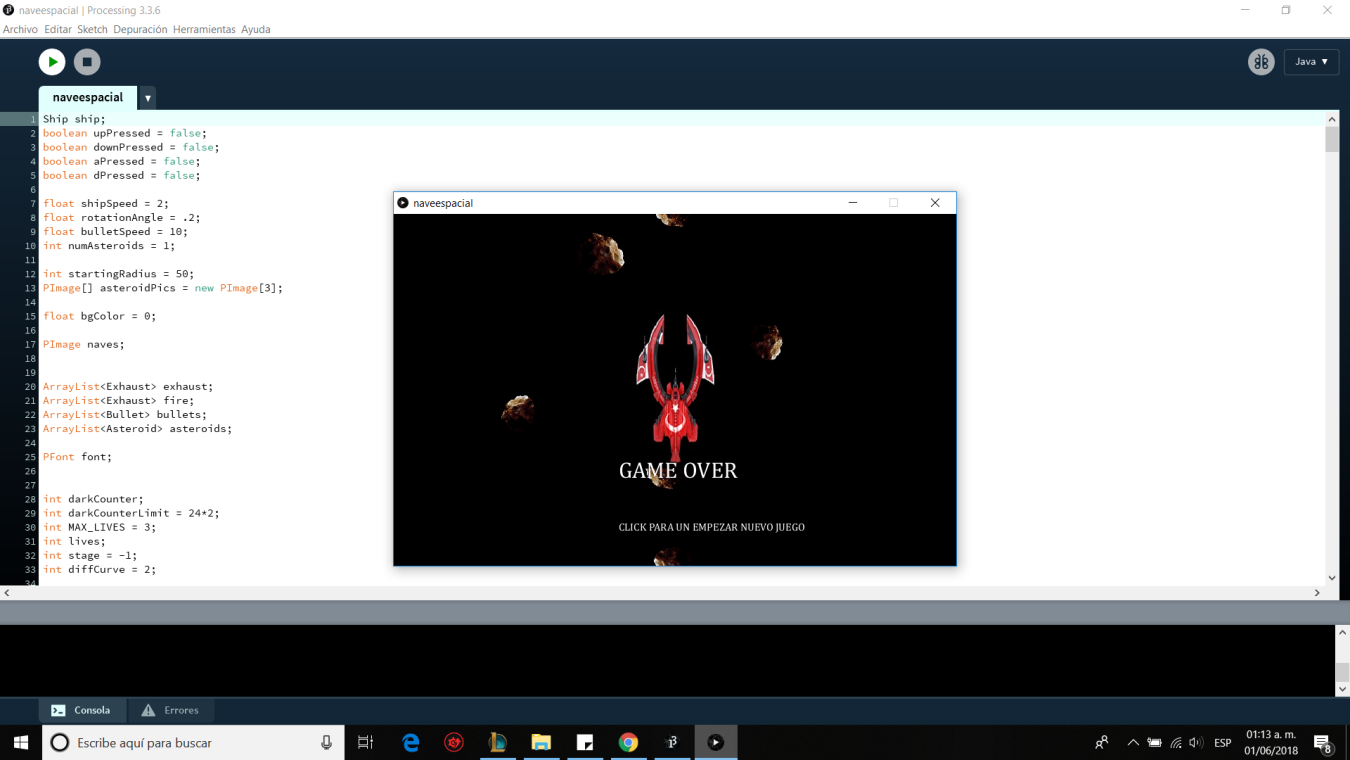
Básicamente el juego trata de no dejarte golpear por los asteroides y al mismo tiempo dispararles para poder romperlos y así evitar que te golpeen, como recordamos tenemos 3 vidas, así que no perderás si te llega a golpear 1 o 2 veces.



Cuando pasas el nivel aparecerá en la pantalla la siguiente ventana en donde dice que has completado la 1 misión.



En caso de perder todas tus vidas y no haber completado la misión aparecerá una ventana con en donde te dirá que el juego se ah acabado.



**Códigos aprendidos en clase que utilizamos en el código del juego de la nave.**

* Boolean: Puede almacenar únicamente dos valores: verdadero o falso. Constantes: true: representa el valor verdadero. false: representa el valor falso.
* Float: Tipo de datos para números de punto flotante, por ejemplo, números que tienen un punto decimal.
* Int: Permiten representar valores numéricos de tipo entero.
* PImages: Tipos de datos para almacenar imágenes, puede mostrar imágenes .gif, .jpg, .tga y .png, así como también se pueden mostrar en espacios 2D y 3D.
* PFond: 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.
* Background (): La función background () establece el color utilizado para el fondo de la ventana de procesamiento.
* for (): 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.
* while (): 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.
* Fill: Establece el color utilizado para rellenar formas. Por ejemplo, si ejecuta relleno (204, 102, 0), todas las formas siguientes se llenarán de naranja.
* If: 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.
* mousePressed (): 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.
* size (): Define la dimensión del ancho y alto de la ventana de visualización en unidades de píxeles. frameRate () Especifica la cantidad de cuadros que se mostrarán cada segundo. Por ejemplo, la función llamada frameRate (24) intentará actualizar 24 veces por segundo.
* keyPressed (): 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.
* keyReleased (): La función keyReleased () se invoca una vez cada vez que se suelta una tecla.
* pushMatrix : 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.
* popMatrix (): 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.
* rotate (): 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 ().
* translate (): 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.

**Conclusión:**

Karen Lorely Reyna Vargas:

Como conclusión para mi esta materia se me hizo muy interesante y muy entretenida, yo que no soy muy buena con la programación pero si soy creativa, y es lo que más me gusto de esta materia, ya que puedes hacer muchísimas cosas interesantes en el programa Pocessing que es el programa en el cual trabajamos, puedes utilizar colores, líneas, puntos, signos, imágenes y hasta audio. A lo largo de este semestre hicimos muchos programas desde un punto hasta un búho con movimiento y sonido. En este proyecto final hicimos un juego de una nave la cual la realizamos con algunas funciones vistas en clase y así poder poner en práctica lo aprendido, cabe destacar que los conocimientos y la forma de enseñar de mi profesor Arturo influyo mucho para ponerle interés a esta la materia y así aprender.

José Adrian Cerón López:

En este proyecto final pusimos en práctica muchos códigos vistos en la clase de graficación, en mi punto de vista el programa processing es una herramienta muy fácil para enseñar y aprender, este programa lo utilizamos a lo largo del semestre y con los conocimientos aprendidos pudimos realizar este videojuego de una nave espacial y asteroides.