SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE IGUALA

GRAFICACION

JUEGO DE ASTEROIDES

PROFESOR: ING. ARTURO CARLOS RODRIGUEZ ROMAN

EQUIPO:

MARIA BIBIANA CARRO AVILA

RAFAEL QUINTO ZAGAL

INGENIERÍA EN SISTEMAS COMPUTACIONALES

SEMESTRE: 6º GRUPO: "B" AULA: E-3

Iguala de la independencia Gro. a 30 de mayo del 2018.

Contenido

Introducción

Processing en un lenguaje de programación que nos permite hacer proyectos e interactivos de diseño digital, por ejemplo, simular un juego, dibujar líneas, una casa, etc.

Es una herramienta para personas que les gusta mucho el diseño y que sepan manejar el lenguaje de programación adecuadamente, ya que esta basado en lenguaje Java. Esta documentación describe el proyecto final presentado en la clase de graficacion, es un juego de asteroides donde simulas estar jugando.

Se explicará cuáles fueron las sentencias utilizadas y para qué sirven, los métodos utilizados, colores, el diseño, todo lo que estuvimos aprendiendo durante el semestre, pero ya presentado en un solo proyecto.

Código del juego

```
Ship ship;
boolean upPressed = false;//CHANGE LEFT AND RIGHT TO UP AND DOWN (IN SHIP TOO)
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); // DECLARAMOS EL COLOR DEL FONDO
 size (800,500); // TAMAÑO
 font = createFont ("Cambria", 32);
 asteroidPics [0] = loadImage("asteroide.jpg"); // ingresamos la ruta de la imagen
 asteroidPics [1] = loadImage("aster.jpg");
 asteroidPics [2] = loadImage("asteroide3.jpg");
 naves = loadImage("nave.jpg");
 frameRate (24);
 lives = 5; // DECLARAMOS LAS VIDAS QUE TENDREMOS EN EL JUEGO
 asteroids = new ArrayList<Asteroid> (0);
}
void draw () {
  // Empieza el codigo
 if (lives >= 0 && asteroids.size ()>0) {
```

```
  float theta = heading2D(ship.rotation)+PI/2; // Sirve para darle la rotacion a la
nave
  background (0);

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

  if(aPressed){
    rotate2D(ship.rotation,-rotationAngle);
    //ellipse(100,100,100,100);
  }
  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("Fin del Juego", width/2-80-2, height*.75-1);
    textFont(font, 32);
    fill(255);
    text("Fin del Juego", width/2-80, height*.75);

    textFont(font, 16);
    fill(0, 200);
    text("Click Para Jugar de Nuevo", width/2-80-2, height*.9-1);
    textFont(font, 15);
    fill(255);
    text("Click 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("Nivel " + (stage + 1) + " Juego Completado", width/2-120, height/2);
    } else {
       text("Bienvenido al Juego de Asteroides", width/2-80, height/2);
    }

    textFont(font, 15);
    fill(255);
    text("Click en la pantalla para jugar Nivel=" + (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);
  // ellipse(0,0,2.1*radius, 1.9*radius);
   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);
}

}
/* @pjs preload="laser.png";
 */
```

```
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);
  }
}
/* @pjs preload="Rocket.png";
 */

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);
}

}
```

Funciones utilizadas

PImage

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 pixeles [] que contiene los valores para cada píxel en la imagen.

PFont

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.

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.

void draw ()

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.

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.

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.

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.

PVector
Una clase para describir un vector de dos o tres dimensiones, específicamente un vector euclidiano (también conocido como geométrico).

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.

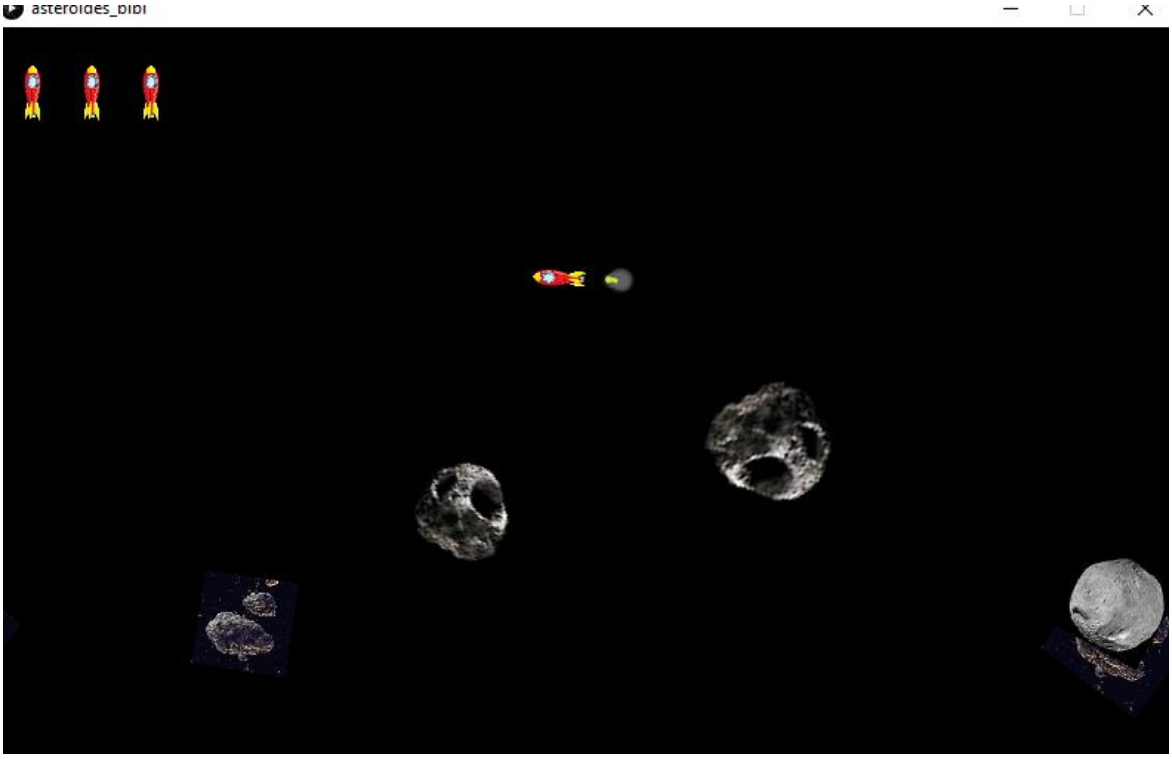
pushMatrix
La función pushMatrix () guarda el sistema de coordenadas actual en la pila y popMatrix () restaura el sistema de coordenadas anterior. pushMatrix () y popMatrix () se usan en conjunción con las otras funciones de transformación y se pueden incorporar para controlar el alcance de las transformaciones.

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.

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.

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 ().

Capturas de pantalla del juego


asteroides_bibi


asteroides_bibi
Fin del Juego
Click Para Jugar de Nuevo

Conclusiones

María Bibiana Carro Ávila

Para concluir este proyecto recordé cada uno de los códigos que nos enseñó a utilizar el profe desde dibujar un punto, una línea, una casita, hasta ponerle movimiento, hacer un dibujo, hacer rotar una figura, etc.

El programa Processing es un programa muy fácil tanto como en el diseño y en la programación. Incluso la sencillez que tiene Processing para crear y diseñar juegos, en este proyecto todos los conocimientos que adquirimos durante el semestre fueron de gran ayuda para sacar adelante este proyecto.

Rafael Quinto Zagal

Con este proyecto de este videojuego que pues lo primero es que se usó algunas variables que puedo reconocer como el upPressed que es cuando le das una declaración con un parámetro diciendo que al presionar esto dará la siguiente opcion, en el juego consta de que el cohete tiene 5 vidas y que al momento de que choque contra un meteorito pues ya habrá perdido vida y así hasta que pierda sus 5 vidas puede también otro detalle interesante es que puede movilizarse para esquivar los meteoritos y se direcciona con las flechitas de direccionamiento del teclado y otra cosa que hace es que dispara a los meteoritos y los divide haciendo así mas difícil la situación

Entre las cosas que más tengo referencia del codigo, es que en el setup mandamos a traer la ruta de las imágenes de los asteroides , definimos el color de fondo con la variable con lives declaramos las vidas que pudiera tener en el draw es donde le damos todas las acciones dinámicas a la nave un ejemplo es theta = heading2D(ship.rotation)+PI/2; que sirve para darle rotacion a la nave if(ship.checkCollision(asteroids)) aquí por ejemplo dice que si la nave choca contra el asteroide pues pierde una vida lives-- ,

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

Aquí por ejemplo es donde le decimos que se moverá al presionar en las direccionales del teclado y pues así se concluye esta experiencia del videojuego de la nave.