Challenge Mercado Libre

Autor: Anibal Villalva

Fecha: 30/11/2018

INTRODUCCIÓN:

Se crea un programa que predice el clima en un sistema solar lejano. Dicho sistema se compone de tres planetas Ferengi , Betasoide y Vulcano que tiene una traslación en una órbita circular con velocidad angular 1 grado/día, 3 grados/día y -5grados/día respectivamente. A su vez los mismos se encuentran a una distancia del sol de 500km, 2000km y 1000km respectivamente.

Se sabe que cuando los planetas están alineados con el sol el sistema solar experimenta un periodo de sequía.

Cuando los planetas no están alineados forman un triángulo, en ese caso, si el sol se encuentra dentro del triángulo, el sistema solar experimenta un periodo de lluvia, con pico de intensidad cuando el perímetro del triángulo está en su máximo.

Por otro lado, se sabe que las condiciones óptimas de presión y temperatura serán cuando los tres planetas están alineados entre sí, pero no con el sol.

En el siguiente trabajo se diseñará un programa informático de predicción del clima que pueda determinar en los próximos 10 años cuantos son los periodos de sequía, de lluvia, y condiciones óptimas. Además, el sistema informará que día será el pico máximo de lluvia en cada periodo.

Además, se generará una web donde se puede consultar el tipo de clima del día deseado.

DESARROLLO

Para la implementación del sistema informático se utilizaron los lenguajes de java (versión 7) y json.

Antes de comenzar a programar se hizo un análisis matemático que permitiera facilitar y agilizar el programa.

Se decidió usar como unidades de medida los grados para ángulos, minutos para el tiempo y metros para la distancia. De todos modos, en algunas ocasiones se requirió hacer las transformaciones, por ejemplo a días.

Como se sabe que el universo ajeno al sistema solar no tenía influencia en los cambios climáticos del mismo, se consideró que el planeta Ferengi, estaría siempre en la posición phi = 0 y R = 500km. De esta manera lo que giraría seria el universo en sentido contrario a la velocidad angular de este astro. Es decir que para este sistema de coordenadas el planeta Betasoide giraría a una velocidad de 2°/día y Vulcano a 6°/ en sentido antihorario.

Dado que el sistema solar tenia orbitas circulares, se eligió trabajar en coordenadas polares. De esta manera para conocer si estaban alineados los planetas con el sol, solo basta saber si los otros dos planetas Betasoide y Vulcano estarían en la posición de Phi = 0 o 180°. De todos modos, para ciertos cálculos convino usar las coordenadas cartesianas, por lo que la posición de los planetas se puede obtener en dichas coordenadas.

A continuación, se explicará la manera que se utilizó para el cálculo de cada periodo.

Periodo de Sequía:

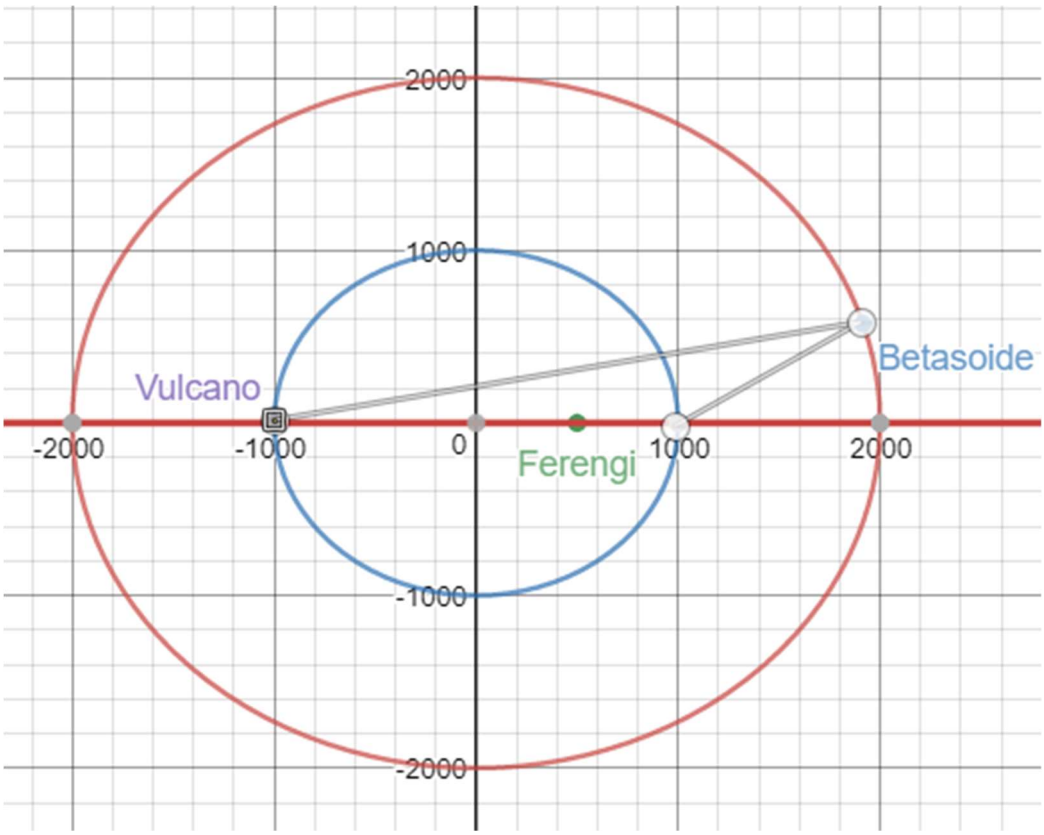
Como las coordenadas son polares y el planeta Ferengi está siempre en 0, cuando los otros dos planetas están en phi = 0 o 180°, los planetas están alineados con el sol. Esto se da solo en un instante de tiempo, sin embargo, para la predicción del clima se consideró que ese día habría sequía.

Periodo de condiciones óptimas:

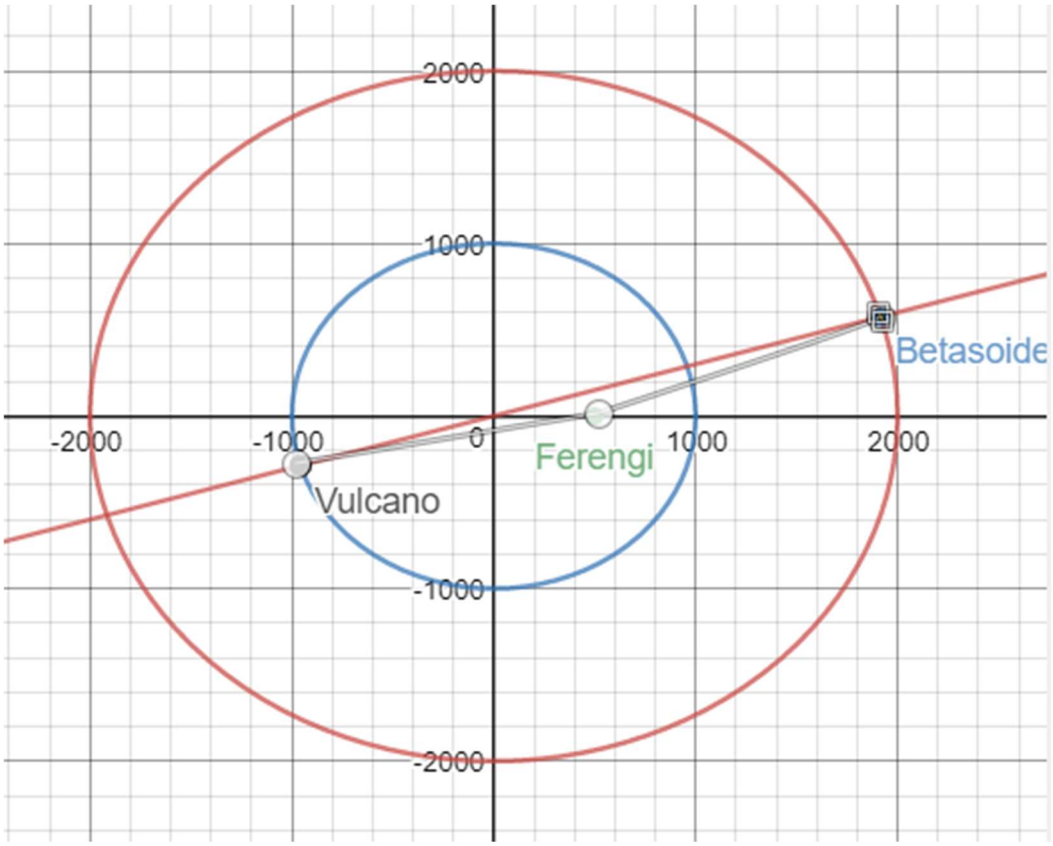
Este periodo sucede cuando los planetas están alineados, pero no con el sol. Al igual que en el periodo de lluvia, solo sucede en un instante, para este periodo también se considera que el periodo dura todo el día. Para calcular si los planetas están alineados se hace una traslación del eje de coordenadas cartesianas a la posición del planeta Ferengi. Matemáticamente es simplemente una resta. Luego se comparan los ángulos que se forman entre Betasoide y Ferengi y el eje X con el ángulo que forma Vulcano con Ferengi y el eje X. Si son equivalentes, entonces los tres planetas estarían alineados, si, el ángulo es distinto de 0 o 180, además estarían alineados sin el sol. Se consideran equivalentes dos ángulos a y b tal que a=n*360+b, siendo n un numero natural cualquiera. Esta consideración en el programa se tiene en cuenta calculando el ángulo absoluto de [0 a 360).

Periodo de Lluvia:

Este periodo sucede cuando los tres planetas forman un triángulo que contiene en su interior al sol. Teniendo en cuenta que Ferengi está “quieto” para este sistema de coordenadas, se puede observar que dada una posición del planeta Betasoide existe una región posible de que Vulcano este posicionada que permita formar un triángulo con el son dentro. Las condiciones de contorno para esta región son si Vulcano esta con phi = 180°, de manera que uno de los lados este tocando al sol. Tener en cuenta que si Betasoide está en 180° o 0° estarían alineados, sin embargo, un instante después Betasoide estaría formando un triángulo.



La otra condición de contorno se da cuando Betasoide y Vulcano tiene la misma pendiente, pero en cuadrantes opuestos. Es decir, la pendiente de uno dista en 180 del otro. De esta manera el lado comprendido entre Betasoide y Vulcano estaría conteniendo al Sol.



Como puede observar entonces dado la posición de Betasoide Vulcano puede estar con un phi = 180 hasta el ángulo opuesto al de Betasoide (phi de Betasoide + 180).

Para el cálculo del día más lluvioso, se toma una variable que guarde en todo momento el perímetro del triángulo. Cuando se termina el periodo de lluvia, guarda el valor máximo. Se entiende que podrían existir más de un máximo relativo dentro del periodo de lluvia, sin embargo, se comprende que el día de lluvia intenso se debe solo al máximo absoluto del periodo.

ANEXO 1 (FUENTES)

```
package Sistema;


import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

/**

 *

 * @author villalvan

 *

 */

public class SistemaMeteorologico {


	/**

	 * Es la funcion principal que se encarga de recorrer los dias y determinar el
clima.

	 * @param args

	 */

	public static void main (String [ ] args) {

		ArrayList<Integer> diasDeLluviaIntensa = new ArrayList<Integer>();

		Map<TipoClima,Integer>		PeriodosClimaticos		=		new
HashMap<TipoClima,Integer>();

		ArrayList<ClimaDia> clima = new ArrayList<ClimaDia>();


		Double PerimetroMaximo = 0.0;

		Double PerimetroActual = 0.0;

		Integer diaLluviaIntenso =0;

		Integer fecha = 0;

		Integer fechaAnterior = 0;


		Exportador exportar = new Exportador(".\\clima.json");


		PeriodosClimaticos.put(TipoClima.LLUVIA, 0);

		PeriodosClimaticos.put(TipoClima.OPTIMO, 0);

		PeriodosClimaticos.put(TipoClima.SEQUIA, 0);
```

```
              SistemaSolar sistemaSolar;



              TipoClima climaDia = TipoClima.DEFAULT;
           TipoClima climaActual = TipoClima.DEFAULT;
           TipoClima climaAnterior = TipoClima.DEFAULT;
           // Ferengi
           sistemaSolar = new SistemaSolar(new Posicion(5E5, 0.0), 1.0 );
           // Betasoide
           sistemaSolar.agregarPlaneta(new Posicion(2E6, 0.0), 3.0 );
           // Vulcano
           sistemaSolar.agregarPlaneta(new Posicion(1E6, 0.0), -5.0 );



              while (Tiempo.getMinutos()<10*Tiempo.ANIOENMIN){
              fecha = Tiempo.getDia();
              climaActual = determinarTipoClima(sistemaSolar);


              /*
               * Se fija si es un dia lluvia y compara con el perimetro del triangulo
maximo del periodo.
               * Si es mayor, entonces aun no llegue al pico y lo guarda como maximo.
               * Guarda en una variable el dia que fue el pico maximo de lluvia del
periodo.
               */
              if(climaActual == TipoClima.LLUVIA){
                     PerimetroActual = sistemaSolar.getPerimetro();
                     if (PerimetroActual > PerimetroMaximo){
                            PerimetroMaximo = PerimetroActual;
                            diaLluviaIntenso = fecha;
                     }
              }
              else {
                     /*
                      * La siguiente condicion se da por si el maximo perimetro se da
en el ultimo dia
```

```
                        * del periodo. Como se esta midiendo mas de una vez al dia, se
pone ">=".
                        * Luego se resetea el dia de lluvia intenso, para que no vuelva
a grabarlo.
                        */
                       if (PerimetroMaximo != 0){
                              diasDeLluviaIntensa.add(diaLluviaIntenso);
                              PerimetroMaximo = 0.0;
                       }


               }
               /* Sumo los acumuladores de los periodos
                * Si el dia anterior hubo otro clima, entonces empieza un nuevo periodo.
                * Ademas chequeo que no sea un tipo de clima indefinido en el enunciado.
                */
               if (climaActual != climaAnterior && climaActual != TipoClima.DEFAULT){
                      PeriodosClimaticos.put(climaActual,
PeriodosClimaticos.get(climaActual) + 1);
               }


               /*
                * Se Determina que clima hizo en el dia, ponderando el tipo.
                * Esto es porque para los casos de sequia y optimo, son casos puntuales
y
                * para mejorar la medicion se toman intervalos de tiempo menores a un
dia.
                * Ademas sabiendo que no puede haber en el mismo dia un caso de sequia
y optimo,
                * solo se tiene en cuenta que no sea de lluvia y default.
                */
               if(fechaAnterior.equals(fecha)){
                      if(climaDia != climaActual &&
                         (climaActual   !=   TipoClima.LLUVIA   ||   climaActual   !=
TipoClima.DEFAULT)){
                                    climaDia = climaActual;
                      }
                }
               else{ // Cambio de dia. Guardo el clima del dia de ayer.
                      clima.add(new ClimaDia(fecha-1,climaDia));
                      climaDia = climaActual;
```

```
            }


            climaAnterior = climaActual;

            sistemaSolar.actualizarT();

            fechaAnterior = fecha;

         }


            // En caso que el ultimo dia medido sea de lluvia, se guarda el dia de
lluvia intensa.

            if (climaActual == TipoClima.LLUVIA){

                   diasDeLluviaIntensa.add(diaLluviaIntenso);

            }




         System.out.println("Periodos      de      "      +      TipoClima.SEQUIA+":
"+PeriodosClimaticos.get(TipoClima.SEQUIA));

         System.out.println("Periodos      de      "      +      TipoClima.LLUVIA+":
"+PeriodosClimaticos.get(TipoClima.LLUVIA));

         System.out.println("Periodos      de      "      +      TipoClima.OPTIMO+":
"+PeriodosClimaticos.get(TipoClima.OPTIMO));

         System.out.println("Dias de lluvia intensa: "+diasDeLluviaIntensa);

         try{

            exportar.toJSON(clima);

         }

         catch (Exception e){

            System.out.println("No se pudo exportar");

         }


      }




      /**

       * Determina el Tipo de Clima del momento de acuerdo a la ubicacion de los
astros.

       * Se deja el periodo de Lluvia a lo ultimo ya que depende si los planetas forman
un

       * triangulo que contiene al sol. En los casos anteriores sucede en casos
puntuales.

       * @param sistemaSolar

       * @return TipodeClima
```

```
	 */
	private static TipoClima determinarTipoClima(SistemaSolar sistemaSolar){
		TipoClima periodoClima;
		if (esPeriodoSequia(sistemaSolar)){
			periodoClima = TipoClima.SEQUIA;
		}
		else if (esCondicionesOptimas(sistemaSolar)){
			periodoClima = TipoClima.OPTIMO;
		}
		else if (esPeriodoLluvia(sistemaSolar)){
			periodoClima = TipoClima.LLUVIA;
		}
		else{
	    periodoClima = TipoClima.DEFAULT;
	 }
		return periodoClima;
	}

	/**
	 * Esto sucede cuando estan alineados con el sol.
	 * @return
	 */
	private static Boolean esPeriodoSequia(SistemaSolar sistemaSolar){
		return sistemaSolar.estanAlineadosConElSol();
	}

	/**
	 * Esto sucede cuando los planetas forman un triangulo y contiene al sol.
	 * Si el perimetro es maximo tiene un pico de intensidad
	 * @return
	 */
	private static Boolean esPeriodoLluvia(SistemaSolar sistemaSolar){
		return sistemaSolar.trianguloContenieneSol();
	}

	/**
	 * Esto sucede cuando los planetas estan alineados, pero no con el sol.
```

```
        * @return
        */
       private static Boolean esCondicionesOptimas(SistemaSolar sistemaSolar){
              return sistemaSolar.estanAlineadosSinElSol();
       }
}
```

```
package Sistema;


import java.util.ArrayList;


/**
 *
 * @author villalvan
 *
 */
public class SistemaSolar {
       /*
        * En este ejemplo, como lo que me interesa es la posicion de los planetas, y siempre la posicion depende
        * de 1 planeta y el sol, para facilitar  es crear los planetas con velocidades relativas al eje formado por
        * el planeta principal y el sol.
        *
        */
       private ArrayList<Planeta> Planetas = new ArrayList<Planeta>();
       public Tiempo tiempo = Tiempo.crearTiempo();


       /*
        * El constructor se genera con un planeta al menos.
        * Considerandose que el sol esta en el eje de coordenadas y el planeta a una distancia r1.
        *  Ademas se supone que aunque exista una velocidad angular del planeta principal, lo que gira es el resto
        * y el planeta siempre mantiene su posicion con angulo phi = 0.
        */
       Double velocidadAngularSistema;
```

```
	public SistemaSolar(Posicion posicionPlaneta , Double velocidadAngular){
		this.velocidadAngularSistema = -velocidadAngular;
		Planeta planetaPrincipal = new Planeta(posicionPlaneta, 0.0);
		Planetas.add(planetaPrincipal);
	}

	public void agregarPlaneta( Posicion posicionPlaneta , Double velocidadAngular ){
		Planeta p = new Planeta(posicionPlaneta, velocidadAngular+this.velocidadAngularSistema);
		Planetas.add(p);
	}

	/**
	 *
	 * @return Devuelve la condicion que los planetas esten alineados con el sol.
	 */
	public Boolean estanAlineadosConElSol(){
		/*
		 * Para chequear que esten alineados solo se fija si el angulo es 0 para todos los planetas,
		 * Esto es factible ya que siempre el planeta principal esta con angulo 0, y la galaxia gira con un phi.
		 */
		for ( Planeta p : Planetas){
			if (p.getPhi() != 0 && p.getPhi() != 180)
				return false;
		}
		return true;
	}

	/**
	 *
	 * @return Devuelve la condicion que los planetas esten alineados sin el sol.
	 */
	public Boolean estanAlineadosSinElSol(){
		/*
		 * Resta la posicion de un planeta con el principal, luego el tercero con el principal
```

```
	 * y compara ambos angulos, si son iguales +- 180 entonces estan en la misma recta.
	 */
	Recta r = new Recta(Planetas.get(0).getPosicion(),Planetas.get(1).getPosicion());
	if (r.contienePunto(Planetas.get(2).getPosicion()))
		return true;

	return false;
}

/**
 *
 * @return Devuelve la condicion que el sol este contenido en el triangulo formado por los planetas.
 */
public Boolean trianguloContenieneSol(){
		Triangulo triangulo = new Triangulo(Planetas.get(0).getPosicion(),Planetas.get(1).getPosicion(),Planetas.get(2).getPosicion());
		if (triangulo.contieneOrigen())
			return true;
		return false;
}

/**
 * Calcula el perimetro del triangulo.
 * @return Devuelve el perimetro en metros.
 */
public Double getPerimetro(){
	Triangulo triangulo = new Triangulo(Planetas.get(0).getPosicion(),Planetas.get(1).getPosicion(),Planetas.get(2).getPosicion());
	return triangulo.getPerimetro();

}

/**
 * Genera la traslacion de los planeta de acuerdo al Delta de Tiempo determinado en el sistema.
```

```
        */
       private void actualizar(){
              for ( Planeta p : Planetas){
                     p.actualizar();
              }
       }

       /**
        *
        * @return Devuelve la lista con los planetas del sistema solar.
        */
       public ArrayList<Planeta> getPlanetas(){
              return Planetas;
       }

       /**
        * Actualiza el tiempo en pasado por parametro y actualiza la posicion de los
planetas.
        * @param tiempo medido en minutos
        */
       public void actualizarA(Integer tiempo){
              Tiempo.setMinutos(tiempo);
              actualizar();
       }

       /**
        * Actualiza el tiempo en el delta T y actualiza la posicion de los planetas.
        */
       public void actualizarT(){
              Tiempo.Actualizar();
              actualizar();
       }

}
```

```
package Sistema;
/**
 *
 * @author villalvan
 *
 */
public class Planeta {
	private Posicion p;
	private static Double anguloInicial;
	// Velocidad angular medida en Grados/minutos
	private Double vAngular;

	/**
	 * Planeta s
	 * @param p Poscion Polar
	 * @param velAngular (Grados/Dia)
	 */
	public Planeta (Posicion p , Double velAngular){
		this.p = p;
		anguloInicial= this.p.getAngulo().getAngulo();
		this.vAngular = velAngular.doubleValue()/Tiempo.DIAENMIN;

	}

	/**
	 *
	 * @return Devuelve la velocidad angular
	 */
	public Double getvAngularPorMin() {
		return vAngular;
	}
	/**
	 *
	 * @return Devuelve la posicion del planeta
	 */
	public Posicion getPosicion() {
		return p;
```

```
	}


	/**
	 *
	 * @return Devuelve el phi del planeta expresado en un angulo entre [0,360)
	 */
	public Double getPhi(){
		return p.getAngulo().getAnguloAbs();
	}


	/**
	 * Genera el movimiento en un periodo de tiempo T de acuerdo a la velocidad Angular.
	 * @param periodoTiempo en minutos
	 */
	private void moverA(Integer periodoTiempo){
		this.p.getAngulo().setAngulo(vAngular*periodoTiempo+anguloInicial);
	}


	/**
	 * Genera la traslacion del planeta de acuerdo al Delta de Tiempo determinado en el sistema.
	 */
	public void actualizar(){
		this.moverA(Tiempo.getMinutos());
	}
}




package Sistema;
/**
 *
 * @author villalvan
 *
 */
public final class TipoClima {
	private String clima;
```

```
	private TipoClima(String tipo){
		clima = tipo;
	}

	public String toString(){
		return clima;
	}

	public final static TipoClima
		LLUVIA = new TipoClima("Lluvia"),
		LLUVIAINTENSA = new TipoClima("Lluvia intensa"),
		OPTIMO = new TipoClima("Clima Optimo"),
		SEQUIA = new TipoClima("Sequia"),
	    DEFAULT = new TipoClima("Indefinido");

	public final static TipoClima[] tipo = {
		LLUVIA, LLUVIAINTENSA, OPTIMO, SEQUIA, DEFAULT
	};



}



package Sistema;
/**
 * Tiempo maneja el transpaso del tiempo se toma la unidad minima el minuto.
 * @author villalvan
 *
 */
public class Tiempo {
	private static Tiempo tiempo;
	private static Integer minutos;
	// Se toma como intervalo de tiempo a medir, 120 min (2hs), es decir que en un dia va a haber 12 mediciones.
	// Esto podria hacer una prueba de un tiempo considerable y aumentar el intervalo, si sigue dando igual,
	// Se mantiene el mayor.
```

```
	// Conviene si que el valor sea multiplo de 1440. En caso de modificarlo, deberia cambiar la funcion getDia


	public static final Integer DELTAT = 120;

	public static final Integer DIAENMIN = 1440;

	public static final Integer ANIOENMIN = DIAENMIN*365;


	private Tiempo (){

		minutos = new Integer(0);

	}


	public static Tiempo crearTiempo(){

		if (tiempo == null){

			tiempo = new Tiempo();

		}

		return tiempo;

	}

	/**

	 * Actualiza el tiempo en el Delta de T determinado en minutos.

	 */

	public static void Actualizar(){

		Tiempo.minutos += DELTAT;

	}


	/**

	 * Setea el tiempo actual en minutos.

	 * @param minutos

	 */

	public static void setMinutos(Integer minutos){

		Tiempo.minutos = minutos;

	}


	/**

	 * Calcula el dia de acuerdo a los minutos transcurridos.

	 * @return Devuelve el dia (Entero)

	 */

	public static Integer getDia(){
```

```
		return minutos/DIAENMIN;
	}


	/**
	 *
	 * @return Deveuelve el tiempo transcurrido en minutos.
	 */
	public static Integer getMinutos(){
		return minutos;
	}
}



package Sistema;


/**
 *
 * @author villalvan
 *
 */
public class Posicion extends Punto{
	/**
	 * Constructor en Coordenadas polares
	 * @param r Radio al origen
	 * @param phi Angulo
	 */
	public Posicion(Double r, Double phi){
		super(r, phi);
	}


	/**
	 * Calcula la distancia entra la coordenada y el parametro b
	 * @param b
	 * @return la distancia entre dos puntos.
	 */
	public Double distancia(Punto b){
```

```
		return Math.pow(Math.pow(b.getX()-this.getX(), 2) + Math.pow(b.getY()-this.getY(), 2),0.5);
	}

}
```

```
package Sistema;
/**
 *
 * @author villalvan
 *
 */
public class Triangulo {
	Posicion a;
	Posicion b;
	Posicion c;
	Double perimetro;
	public Triangulo(Posicion a, Posicion b, Posicion c){
		this.a = a;
		this.b = b;
		this.c = c;
		perimetro = a.distancia(b) + a.distancia(c) + b.distancia(c);
	}

	/**
	 *
	 * @return Determina si el triangulo contiene al origen.
	 */
	public Boolean contieneOrigen(){
		/*
		 * Solo se esta teniendo en cuenta un triangulo que tiene un vertice 1 en el origen.
		 *
		 * Si el planeta 1 esta en el semiplano Y > 0 El planeta 2 deberia estar en el semiplano inferior
		 * y ademas menor al angulo -phi del planeta 1.
```

```
                * Paralemente debe sudecer lo mismo si el planeta 2 esta en el semiplano
inferior, el planeta 1
                * debe estar en el semiplano superior y menor a - phi del planeta 2
                * Si es el angulo es 180 el sol siempre esta dentro.
                */
               if((b.estaSemiplanoSuperior()    &&    c.estaSemiplanoInferior()    &&
(b.getAngulo().getPendiente() >= c.getAngulo().getPendiente()  )) ||
                  (c.estaSemiplanoSuperior()    &&    b.estaSemiplanoInferior()    &&
(c.getAngulo().getPendiente() >= b.getAngulo().getPendiente() )) ||
                  (b.estaEnEjeX() || b.estaEnEjeX() ))
                      return true;
               return false;
       }

       /**
        *
        * @return Devuelve el perimetro.
        */
       public Double getPerimetro(){
              return perimetro;
       }



}



package Sistema;
/**
 *
 * @author villalvan
 *
 */
public class Recta {
       Punto p1;
       Double pendiente;
       static Double Error = 1.0;
```

```
	public Recta(Punto p1, Punto p2){
		this.p1 = p1;
		pendiente = (p1.menos(p2)).getAngulo().getPendiente();
	}
	/**
	 * La siguiente funcion verifica que el punto este dentro de la recta.
	 * para esto verifica que la pendiente de uno sea similar al otro.
	 * @param p3
	 * @return
	 */
	public Boolean contienePunto(Punto p3){
		Double pendiente2 = (p1.menos(p3)).getAngulo().getPendiente();
		if (Math.abs(pendiente2 - pendiente) < Error ){
			return true;
		}
		return false;
	}
}
```

```
package Sistema;
/**
 *
 * @author villalvan
 *
 */
public class Punto {

    private Double r;
    private Angulo phi;
   /**
    * Coordenadas en polares
    * @param r es el radio
    * @param phi es un angulo en grados
    */
    public Punto(Double r, Double phi){
```

```
       this.r = r;
       this.phi = new Angulo(phi);
    }


    /**
     * Constructo en base a coordenadas cartesianas
     * Se considera que como los valores superan los miles, se usa enteros
     * ya que el error es despreciable en decimos.
     * @param x Entero
     * @param y Entero
     */
    public Punto(Integer x, Integer  y){
       r = Math.pow(x*x+y*y, 0.5);
       phi = new Angulo( x, y);
    }


    /**
     *
     * @return Devuelve la distancia al origen.
     */
       public Double getRadio() {
              return r;
       }


       /**
        *
        * @return Devuelve el angulo del punto. No es absoluto puede ir de -infinito a
+infinito.
        */
       public Angulo getAngulo() {
              return phi;
       }


       /**
        *
        * @return Devuelve la commponente X de las coordenadas cartesianas.
        */
```

```
	public Integer getX(){
		return (int) (this.getRadio()*Math.cos(this.getAngulo().getAnguloRad()));
	}

	/**
	 *
	 * @return Devuelve la commponente Y de las coordenadas cartesianas.
	 */
	public Integer getY(){

		return (int) (this.getRadio()*Math.sin(this.getAngulo().getAnguloRad()));
	}

	public void setPhi(Angulo phi) {
		this.phi = phi;
	}

	/**
	 *
	 * @param p2
	 * @return Devuelve un vector con la resta entre ambos.
	 */
	public Punto menos(Punto p2){
		return new Punto(this.getX()-p2.getX(),this.getY()-p2.getY());
	}

	/**
	 *
	 * @return Devuelve la condicion que este en semiplano superior, podria estar en el eje X.
	 */
	public Boolean estaSemiplanoSuperior(){
		return phi.getAnguloAbs() <= 180;
	}

	/**
```

```
        *
        * @return Devuelve la condicion que este en semiplano inferior, podria estar en el eje X.
        */
       public Boolean estaSemiplanoInferior(){
              return (phi.getAnguloAbs() >= 180 || phi.getAnguloAbs() == 0);
       }

       /**
        *
        * @return Devuelve la condicion que este en el Eje X.
        */
       public Boolean estaEnEjeX(){
              return phi.getPendiente() == 0;
       }
}
```

```
package Sistema;

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;


import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonIOException;

/**
 *
 * @author villalvan
 *
 */

public class Exportador {
```

```
	String path;


	public Exportador(String path){
		this.path = path;
	}
	/**
	 * Exporta la lista de clima a un archivo con formato JSON
	 * @param dias
	 * @throws JsonIOException
	 * @throws IOException
	 */
	public void toJSON(ArrayList<ClimaDia> dias) throws JsonIOException, IOException{

		Gson gson = new GsonBuilder().setPrettyPrinting().create();

		try (FileWriter writer = new FileWriter(path)) {
			for(ClimaDia dia : dias){
				gson.toJson(dia, writer);
			}

		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}




package Sistema;
/**
 *
 * @author villalvan
 *
 */
public class Angulo {
	// El angulo esta expresado en Grados
```

```
	private Double angulo;


	public Angulo(Double angulo){
		this.angulo = angulo;
	}


	public Angulo(Integer x, Integer y){
		angulo = Math.toDegrees((Math.atan2(y.doubleValue(), x.doubleValue())));
	}


	/**
	 *
	 * @return angulo en grados
	 */
	public Double getAngulo(){
		return angulo;
	}


	/**
	 * Calcula las vueltas de un angulo
	 * @return devuelve un entero correspondiente a la cantidad de vueltas.
	 */
	public Integer getVueltas(){
		Double aux = angulo/360.0;
		return aux.intValue();
	}


	/**
	 * Calcula el angulo en Radianes
	 * @return Devuelve angulo en radianes.
	 */
	public Double getAnguloRad(){
		return Math.toRadians(angulo);
	}
	/**
	 * Calcula el valor absoluto de un angulo entre [0,360)
	 * @return el angulo en grados
```

```
	 */
	public Double getAnguloAbs(){
		Double anguloAbs;
		if (angulo<0)
			anguloAbs = angulo+((1-this.getVueltas())*360);
		else
			anguloAbs = angulo-(this.getVueltas()*360);
		return anguloAbs;
	}

	/**
	 * Devuelve la pendiente, es decir el angulo en el semiplano superior.
	 * Si el angulo absoluto esta en el semiplano inferior (180,360), le resta 180.
	 * @return Pendiente en grados [0,180]
	 */
	public Double getPendiente(){
		Double pendiente = this.getAnguloAbs();
		if(pendiente>180)
			pendiente-=180;
		return pendiente;
	}

	/**
	 * Suma dos angulos angulos
	 * @param sumando Definido en Grados
	 */
	public void sumar(Double sumando){
		angulo += sumando;
	}

	public void setAngulo(double angulo) {
		this.angulo = angulo;
	}

}
```

```
package Sistema;
/**
 *
 * @author villalvan
 *
 */
public class ClimaDia {

	Integer dia;
	String clima;

	public ClimaDia(Integer dia, TipoClima clima){
		this.dia = dia;
		this.clima = clima.toString();
	}
}
```

Para el API REST uso el https://console.cloud.google.com

Se creo el siguiente archivo js

```
var express = require('express');
var app = express();
var fs = require("fs");

/**
 * Trae todos los dias ./dias
 */
app.get('/dias', function (req, res) {
   fs.readFile( __dirname + "/" + "clima.json", 'utf8', function (err, data) {
      console.log( data );
      res.end( data );
   });
})

/**
 * Se genera la consulta, trayendo el dia deseado. Se tiene en cuenta que la lista empieza del dia 0,
```

```
 * y el archivo json, tiene todos los dias de manera ordenada.
 * Sino habria que hacer una busqueda en el fuente.
 * La manera de invocarlo es la siguiente
 * https://../clima?dia=x
 * donde x es el valor del dia a buscar.
 */
app.get('/clima', function (req, res) {
   // First read existing dia.
   fs.readFile( __dirname + "/" + "clima.json", 'utf8', function (err, data) {
      var dias = JSON.parse( data );
      res.end( JSON.stringify(dias[req.query.dia]));
         console.log("busca parametro " +req.query.dia);

   });
})

var server = app.listen(8080, function () {
   var host = server.address().address
   var port = server.address().port
   console.log("Example app listening at http://%s:%s", host, port)
})
```

ANEXO 2 (SALIDA)

Salida por consola

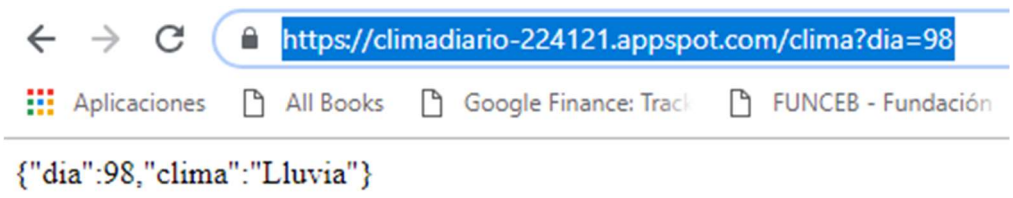
```
Periodos de Sequia: 21
Periodos de Lluvia: 81
Periodos de Clima Optimo: 142
Dias de lluvia intensa: [25, 72, 107, 154, 205, 252, 287, 334, 385,
432, 467, 514, 565, 612, 647, 694, 745, 792, 827, 874, 925, 972, 1007,
1054, 1105, 1152, 1187, 1234, 1285, 1332, 1367, 1414, 1465, 1512,
1547, 1594, 1645, 1692, 1727, 1774, 1825, 1872, 1907, 1954, 2005,
2052, 2087, 2134, 2185, 2232, 2267, 2314, 2365, 2412, 2447, 2494,
2545, 2592, 2627, 2674, 2725, 2772, 2807, 2854, 2905, 2952, 2987,
3034, 3085, 3132, 3167, 3214, 3265, 3312, 3347, 3394, 3445, 3492,
3527, 3574, 3625]
```

Archivo JSON generado


```
[
  {
    "dia": 0,
    "clima": "Indefinido"
  },
  {
    "dia": 1,
    "clima": "Indefinido"
  },
  {
    "dia": 2,
    "clima": "Indefinido"
  },
…
]
```


Consulta de URL:

https://climadiario-224121.appspot.com/clima?dia=98



CONCLUSIÓN

Con el programa generado se predijo que el sistema solar presentaría 21 periodos de sequía, 81 periodos de lluvia y 142 periodos donde el clima tendría condiciones óptimas de presión y temperatura. Además, se concluyó que los días de máximas precipitaciones serian lo siguientes.

25, 72, 107, 154, 205, 252, 287, 334, 385, 432, 467, 514, 565, 612,
647, 694, 745, 792, 827, 874, 925, 972, 1007, 1054, 1105, 1152, 1187,
1234, 1285, 1332, 1367, 1414, 1465, 1512, 1547, 1594, 1645, 1692,
1727, 1774, 1825, 1872, 1907, 1954, 2005, 2052, 2087, 2134, 2185,
2232, 2267, 2314, 2365, 2412, 2447, 2494, 2545, 2592, 2627, 2674,
2725, 2772, 2807, 2854, 2905, 2952, 2987, 3034, 3085, 3132, 3167,
3214, 3265, 3312, 3347, 3394, 3445, 3492, 3527, 3574, 3625.