



Universidad Autónoma de Chihuahua

Facultad de ingeniería

Graficación
O2Graf

Documentación de proyecto del primer parcial

Proyecto 1

Fundamentos de la Graficación

Equipo 6:

Rocío Rivera Corral	235704
Elsa Balderrama Sosa	235811
Claudia Medina Echavarría	235638
Denisse Robles Chacón	235572

Grupo 8SW1

Ingeniería en Software

A 26 de Febrero del 2013.

Tabla de Contenido

Introducción.....	3
Desarrollo del proyecto.....	3
Planteamiento del problema.....	3
Análisis.....	3
Código.....	7
Index.html.....	7
Medusa.js.....	8
Style.css.....	12
Resultado.....	13
Recomendaciones.....	13
Conclusiones.....	14
Referencias.....	14

Introducción

La graficación se ha convertido en un elemento común y cada vez más complejo de las interfaces de usuario. Ésta aporta desde simple atractivo a la aplicación hasta mejoras importantes en interactividad o visualización de información.

Este proyecto busca aplicar los conceptos básicos de la graficación, tales como dibujo de figuras, transformaciones básicas, animación, manejo de formatos gráficos, personalización de elementos de la UI, graficación procedural y proyección de figuras tridimensionales. Los objetivos de la actividad son los siguientes:

- Incrementar la familiaridad con las transformaciones básicas de la graficación
- Explorar las posibilidades que brinda la graficación en el desarrollo de software.
- Mejorar el entendimiento del funcionamiento detrás de las aplicaciones gráficas.

Desarrollo del proyecto

Planteamiento del problema

Las librerías de graficación suelen ser relativamente simples de usar, pues abstraen muchos cálculos y acciones al programador. Sin embargo, para dibujar elementos más complicados quizá no sea factible programar cada componente de la figura, sino acudir a algoritmos procedurales que modelen la estructura de la misma y se encarguen de construirla en base a criterios dados.

Para ejemplificar lo redactado anteriormente se realizará una animación de una medusa por medio de transformaciones iterativas y funciones matemáticas. Partimos de una animación flash que será replicada por medio de JavaScript y HTML5.

La animación original se encuentra en la siguiente URL:

<http://www.cristalab.com/ejemplos/inteligencia-artificial-medusa-c77l/>.

Análisis

Flash es un IDE para el lenguaje de programación ActionScript orientado a hacer animaciones, videojuegos y dibujar funciones matemáticas, este IDE cuenta con un lienzo y una línea de tiempo compuesta por fotogramas, cada fotograma indica una transformación que se aplica a sus instancias. Podemos hacer una analogía entre las instancias de Flash y los componentes que provee la paleta NetBeans, generalmente una instancia representa una imagen o agrupación de vectores. En pocas palabras, Flash abstrae la parte de la graficación y únicamente requiere recibir la lógica del programa.

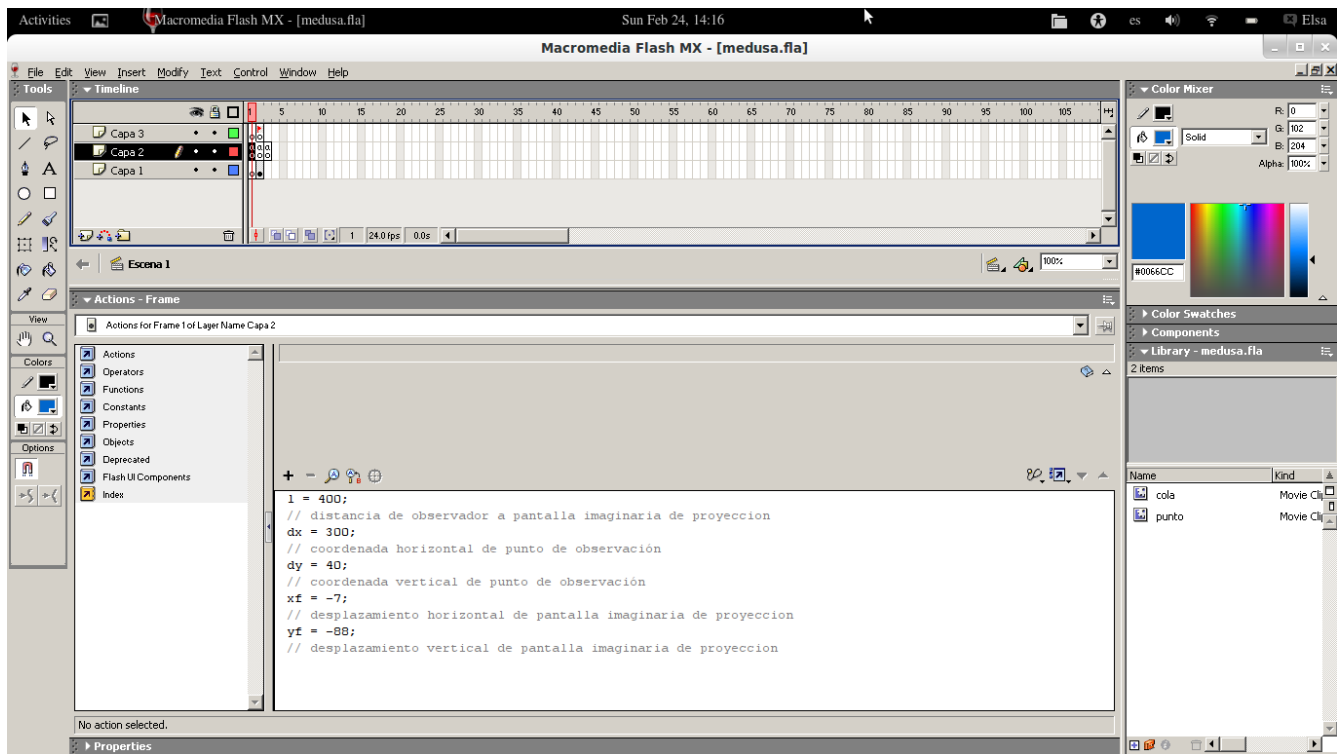


Figura 1: Inicialización de variables

La Figura 1 es una captura del proyecto de la medusa, nos muestra que el primer fotograma de la línea del tiempo está vacío, pero en la capa2 se encuentra un código para inicializar algunas variables, pero sus descripciones no son muy claras. Luego de analizar el programa llegamos a las siguientes conclusiones:

- **l** representa la profundidad del eje Z del escenario.
- **dx y dy** son necesarios porque las funciones por sí solas dibujan a la medusa fuera del escenario.
- **xf y yf** no parecen tener ningún efecto relevante, sus valores son muy pequeños respecto al tamaño del escenario.

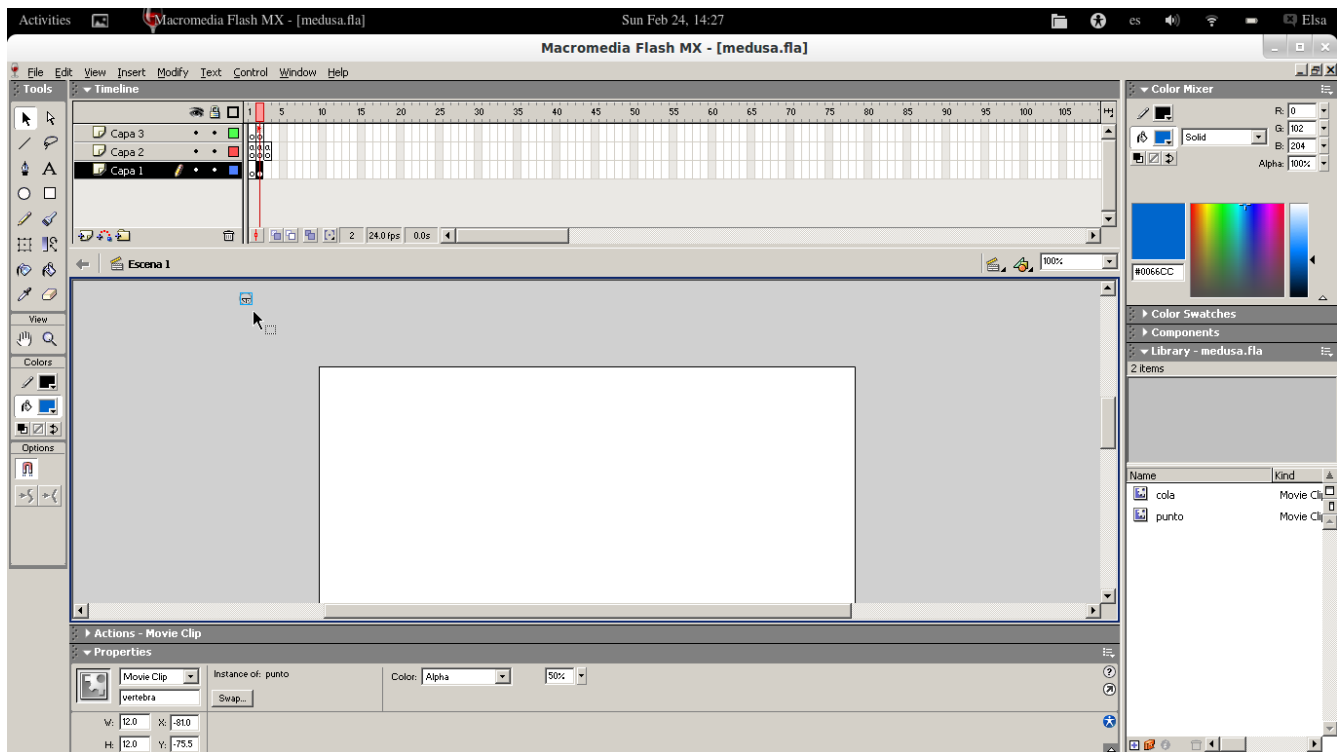


Figura 2: Instancia "vertebra"

En la Figura 2 se puede apreciar que en la capa 3 del segundo fotograma existe una instancia llamada "vertebra" que representa el centro de cada parte de la medusa. Se decidió no usarla en la reconstrucción y en su lugar utilizar un punto de origen en las figuras.

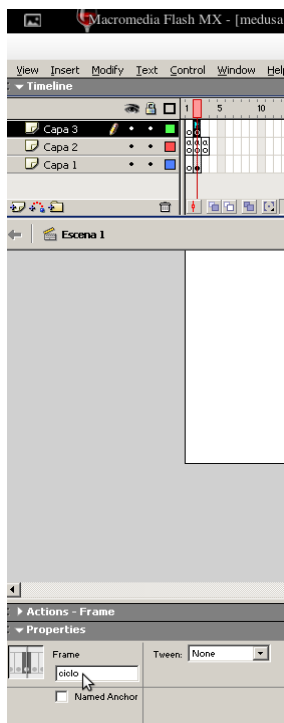


Figura 3: Ciclo

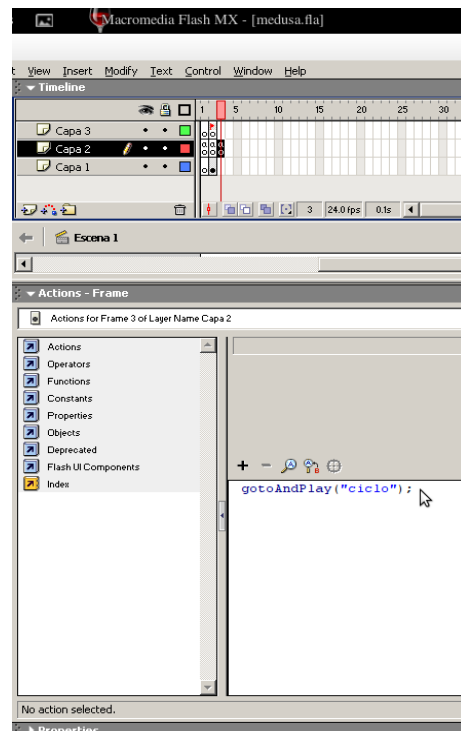


Figura 4: Repetición de ciclo

La Figura 3 y Figura 4 muestran que el segundo fotograma de la capa1 tiene el nombre de "ciclo" y el fotograma 3 indica que se redibujará una y otra vez el fotograma "ciclo".

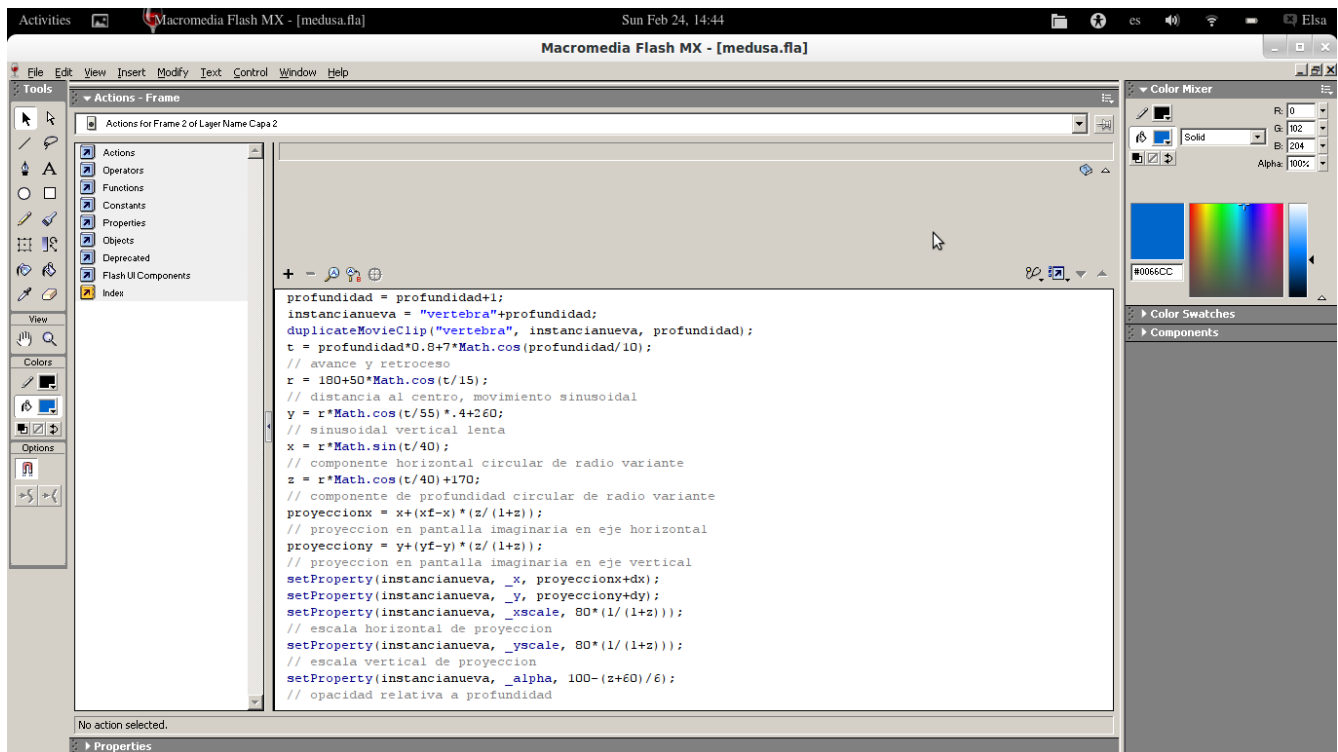


Figura 5: Funciones matemáticas

La Figura 5 muestra el resto del código, son las funciones que calculan cada desplazamiento de cada duplicado de la instancia “vertebra”. Este código se encuentra en la capa 2 del segundo fotograma, por lo tanto es el código que se repite y sus variables se recalculan en cada iteración. El mayor reto del proyecto fue hacer la analogía entre duplicar la “vertebra” en Flash y dibujar para cada desplazamiento la progresión correspondiente, ya que generar muchos clips y animarlos de manera independiente en JavaScript hubiera sido un proceso más complejo. Luego de analizar el programa se concluyó lo siguiente respecto a las variables:

- **profundidad** no se logró definir porque se llama profundidad, representa el punto actual de la animación que es la base de las progresiones entre un tiempo y otro, se decidió que el nombre “frame” era más adecuado.
- **t** representa la proporción entre avance y retroceso de la medusa.
- **r** es la frecuencia y radio de los movimientos oscilatorios de la medusa, provee desviaciones a la trayectoria elíptica que la hacen ver más realista.
- **x, y, z** son las nuevas coordenadas calculadas.
- **px y py** son las proyecciones de las coordenadas aplicándoseles una distorsión en función al eje z (perspectiva).
- En base a estas variables se calculan la escala y transparencia de las figuras a dibujar.

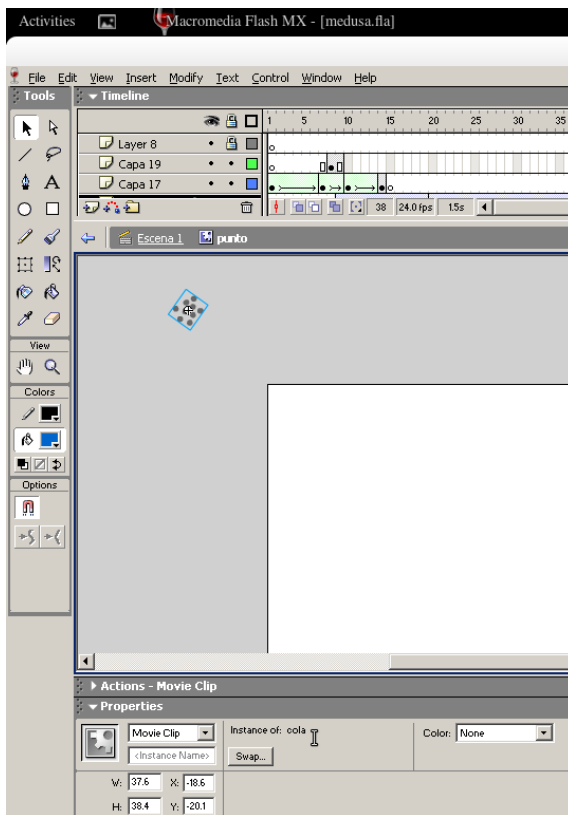


Figura 6: Contenido de la "vertebra"

Por último, la Figura 6 nos muestra que dentro de la instancia "vertebra" hay una animación llamada "punto" de un círculo que se expande y se convierte en muchos puntos pequeños, a su vez, esos puntos son una instancia que contiene una animación llamada "cola" en la que esos puntos se expanden y giran concéntricamente.

Código

El código está dividido en tres documentos: un **index.html** que contiene el lienzo donde se va a dibujar, un **medusa.js** que tiene el programa y un **style.css** que contiene la configuración de apariencia de la interfaz gráfica que sobrescribe a los valores predefinidos por el sistema.

Index.html

```
<!DOCTYPE/>
<html>
<header>
  <title>Medusa</title>
  <link rel="stylesheet" href="style.css" / >
  <script src="medusa.js"></script>
</header>
<body>
  <form>
    <input type="button" value="Botón">
    <input type="text" >
    <select>
      <option value="opcion1">opción1</option>
      <option value="opcion2">opción2</option>
    </select>
    <fieldset><legend>Panel</legend>
      <input type="radio"> radiol
      <input type="checkbox"> checkbox
    </fieldset>
  </form>
  <div id="wrapp">
    <div id="overlay"></div>
```

```

        <canvas id="lienzo" width="500" height="500"></canvas>
    </div>
</body>
<footer>
    Original (Flash) : <br>
    <a href="http://www.cristalab.com/ejemplos/inteligencia-artificial-medusa-
c771/">http://www.cristalab.com/ejemplos/inteligencia-artificial-medusa-c771/</a><br>
    - Adaptación: Elsa Balderrama.
</footer>
</html>

```

En el código de index.html lo importante es el objeto canvas, que es equivalente a instanciar un componente de Java. A demás se agregaron algunos elementos de UI a los cuales se les cambiará la apariencia predefinida por medio de la hoja de estilos y una imagen png con transparencia que se coloca sobre el objeto canvas.

Medusa.js

Se realizó primero un objeto Location, de tal manera que se dibujara la medusa estática con sus partes centradas en el mismo punto, posteriormente se realizaron los timers de tal manera que la medusa avanzara en línea recta, finalmente se insertaron las fórmulas tomadas del ejemplo original para crear un arreglo de puntos y se agregaron los atributos scale y alpha a la clase Location.

```

window.addEventListener('load', inicio);

var
    i, //contador genérico
    frame, //fotograma actual (incrementa)
    l, //rango de z
    dx, //desplazamiento de la coordenada x
    dy, //desplazamiento de la coordenada y
    t, //progresión de tiempo (aceleración)
    o, //frecuencia y radio de oscilación
    x, //x calculada
    y, //y calculada
    z, //z calculada
    px, //proyección en x (distorsión por z)
    py, //proyección en y (distorsión por z)
    nx, //x mostrada
    ny, //y mostrada
    a, //alfa
    s, //escala
    jellyfish, //objeto de animación
    steps, //capas de la medusa
    points = []; //lista de puntos por los cuales cruzará la medusa

```

/ Hasta aquí es un bloque de declaraciones, comienza con un manejador de eventos que espera a que la página se haya cargado completamente para poder ejecutar el resto del código. Luego se establecen las variables globales. */*

```

function Location(x, y, context){
    self = this;
    this.current = 0; //fotograma actual de la animación del punto
    this.context = context;
    this.x = x; //coordenada x del punto actual
    this.y = y; //coordenada y del punto actual
    this.scale;
    this.alpha; //transparencia
}

```


/* La clase Location representa un punto por el que está pasando la medusa, self=this sirve para mantener una referencia a la instancia para los métodos de la clase. La variable "current" almacena el estado de ese punto, es decir, qué parte de la medusa está en ese punto en un momento dado. La variable context (que se explica más adelante) se agregó por flexibilidad, ya que originalmente se había planeado utilizar un prerender antes de mostrar las figuras. */

```
Location.prototype = {
  tentacles : 100, //longitud de tentáculos
  dots : [ //posición de los tentáculos respecto al centro de la medusa
    {x : 7,      y: -3},
    {x : 3,      y: -5},
    {x : 7,      y: 4},
    {x : 1,      y: 5},
    {x : 4,      y: 1},
    {x : -6,     y: -3},
    {x : -2,     y: -7},
    {x : -1,     y: -1},
    {x : -8,     y: 2},
    {x : -4,     y: 7},
  ],
  circles : [ //radio de los anillos de la cabeza
    5, 12, 17, 22, 25, 27, 24, 15
  ],
}
```

/* Los prototipos se usan para declarar las funciones y propiedades que serán comunes de todos los objetos de la misma clase, en este caso, todas las instancias de la medusa tendrán la misma longitud, tentáculos y aros de la cabeza. */

```
animate : function(){
  if(this.current <= this.circles.length){
    this.redraw[0]();
  } else if (this.current > this.circles.length &&
    this.current <= this.circles.length + this.tentacles){
    this.redraw[1]();
  }
  this.current ++;
},
```

/* Si el estado actual del punto es un índice de la lista de círculos, dibuja el círculo de dicho índice, de lo contrario dibuja una capa de tentáculos. Luego incrementa 1 al estado actual. */

```
redraw : [
  function(){
    self.context.strokeStyle = 'rgba(0,0,0,'+ self.alpha +')';
    self.context.beginPath();
    self.context.arc(self.x, self.y,
(self.circles[self.current])*self.scale, 0, 2*Math.PI, false);
    self.context.stroke();
  },
]
```

/* redraw contiene 2 funciones, la primera dibuja un círculo; primero establece el color el trazo (incluyendo la transparencia), luego llama a la función "arc" para dibujar un círculo con centro en las coordenadas del punto y radio igual al valor correspondiente en la lista de círculos multiplicado por la tasa de redimensionamiento (propiedad scale). */

```
function(){
  self.context.save();
  var curdot = self.current - self.circles.length;
  self.context.translate(self.x, self.y);
  self.context.rotate(curdot * 0.02);
  var dispersion = 0.7+curdot*(curdot)*0.001;
```

```

        alpha2 = 0.4 - 1/curdot;
        self.context.fillStyle = 'rgba(0, 0, 0, ' + alpha2 * self.alpha + ')';
        for(var i in self.dots){
            self.context.beginPath();
            self.context.arc(self.dots[i].x * dispersion,
                self.dots[i].y * dispersion,
                (2-(curdot*0.012))*self.scale, 2*Math.PI, false);
            self.context.fill();
        }
        self.context.restore();
    },
]
}

```

/* La segunda función de redraw dibuja la capa de tentáculos que está pasando por el punto dado. Primero almacena el estado actual del lienzo, ya que se realizarán transformaciones difíciles de revertir, luego “curdot” obtiene la capa de tentáculos actual. La razón por la que se necesita trasladar el origen del lienzo es para poder hacer que la capa de tentáculos gire en torno al centro de la medusa (la rotación es mayor conforme los tentáculos se alejan de la medusa). Luego, la dispersión es un valor que modifica la ubicación de los tentáculos, simulando que se alejan del centro, mientras que la variable alpha2 hace que se incremente la transparencia de acuerdo a la capa que se está dibujando. Por último, se recorre la lista de tentáculos para dibujar cada punto de la capa, hay que aclarar que los tentáculos también se harán más delgados conforme se alejan; luego de dibujar los tentáculos se restaura el estado del lienzo para que sea usable por la próxima llamada. */

```

function inicio(){
    canvas = document.getElementById('lienzo');
    ctx = canvas.getContext('2d');

    i = 1;
    frame = 0;
    l = 300;
    dx = 200;
    dy = 0;

    calc();
    points.push({x: nx, y: ny});
    jellyfish = new Location(points[0].x, points[0].y, ctx);
    steps = jellyfish.tentacles + jellyfish.circles.length;

    spawnI = setInterval(spawn, 40);
}

```

/* La función inicio se ejecuta inmediatamente después de cargar la página y es la que dispara las demás funciones, la variable “canvas” guarda una referencia al objeto canvas que está en el HTML, luego en ctx se referencia una librería que incluye la mayoría de los navegadores (contexto) que actúa como una API para comunicarse con el canvas, que equivale a instanciar un objeto Graphics o Java2D en Java. Luego se inicializan las variables que no se recalculan, se crea el primer punto del arreglo de puntos para poder instanciar el objeto jellyfish de tipo Location. Por último se inicia la función spawn que se ejecutará cada 40ms. */

```

function spawn(){
    frame ++;
    calc();
    points.push({x: nx, y: ny});
    if(i <= steps){
        canvas.width = canvas.width;
        jellyfish.current = 0;
        for (var k=0; k <= i; k++){
            jellyfish.x = points[i-k].x;

```

```

        jellyfish.y = points[i-k].y;
        jellyfish.scale = s;
        jellyfish.alpha = a;
        jellyfish.animate();
    }
    i++;
} else {
    clearInterval(spawnI);
    playI = setInterval(play, 40);
}
}

```

/* La función spawn comienza a rellenar el arreglo de puntos, en cada iteración se reinicializa el lienzo y el estado de la medusa. Como todavía no se termina de dibujar la longitud total de la medusa, se iteran solo los puntos que ya se han agregado a la lista en orden inverso, de tal manera que el último punto que se agregó tendrá el estado más bajo de la medusa (la punta) y viceversa. Es necesario mencionar que el lienzo no se dibuja por medio de un for, sino que se dibuja una progresión de tiempo entera, es el intervalo el que hace que se dibuje la medusa completa luego de terminar el for. Una vez que se ha rellenado la lista de puntos de manera que sea superior a la longitud de la medusa, se detiene el intervalo y se inicia uno nuevo. */

```

function play (){
    frame ++;
    calc();
    points.push({x: nx, y: ny});
    canvas.width = canvas.width;
    jellyfish.current = 0;
    i = 1;
    while (i <= steps){
        jellyfish.x = points[points.length-i].x;
        jellyfish.y = points[points.length-i].y;
        jellyfish.scale = s;
        jellyfish.alpha = a;
        jellyfish.animate();
        i++;
    }
    points.shift();
}

```

/* La diferencia entre la función play y la función spawn es que play irá borrando de la lista aquellos puntos por los que la medusa ya pasó, evitando que la memoria se sature en algún momento. */

```

function calc(){
    t = frame*0.8+7*Math.cos(frame/10);
    if (frame > 1259){
        frame = 0;
    }
    o = 180+50*Math.cos(t/12);
    y = o*Math.cos(t/55)*0.4+260;
    x = o*Math.sin(t/40);
    z = o*Math.cos(t/40)+170;
    px = x-(x)*(z/(1+z));
    py = y-(y)*(z/(1+z));

    nx = px+dx;
    ny = py+dy;

    a = 1-(z+50)/500;
    s = 1.5*(1/(1+z));
}

```

/* Por último, la función calc realiza todos los cálculos tomados del proyecto original, se agregó una evaluación de la variable frame para evitar que se haga demasiado grande y hacer que se cicle en lugar de incrementar indefinidamente. */

Style.css

La mayor parte del archivo style.css está dedicado a organizar los elementos de la página, pero tiene algunos selectores que también modifican los elementos de interfáz de usuario predefinidos generalmente por el sistema operativo y equivale a utilizar librerías de temas con Java.

```
body {
    height: 100%;
    margin: 0px;
    padding: 0px;
    width: 100%;
}
canvas{
    border: 1px solid;
    display: block;
    margin: auto;
}
footer{
    font-size: 0.9em;
    margin: 10 50;
}
#overlay{
    background-image: url('overlay.png');
    height: 500px;
    position: absolute;
    width: 500px;
}
#wrapp{
    height: 500;
    margin: 20;
    width: 500;
}

/* A partir de aquí están los estilos de los elementos UI */

form{
    display: block;
    background-color: rgba(90, 90, 90, 0.5);
    padding: 10px;
    border: solid 2px #909090;
    font-size: 0.8em;
    right: 10%;
    margin-top: 30;
    position: absolute;
    width: 240;
    z-index : 2;
    box-shadow: -1px 1px 4px 0px rgba(0, 0, 0, 0.3);
}
input[type=text]{
    width: 100px;
}
input[type=button]{
    border: outset;
}
input, select, option{
    color: #F0F0F0;
    background-color: #121212;
    border: solid 1px;
}
```

La Figura 7 y Figura 8 comparan el antes y el después de aplicar las hojas de estilo al formulario.

Una captura de pantalla de un formulario web sin estilos. Contiene un botón con el texto 'Botón', un campo de texto vacío, un menú desplegable con el texto 'opción1', un panel con el texto 'Panel', un botón de radio con el texto 'radio1' y un botón de casilla con el texto 'checkbox'.

Figura 7: Antes de la hoja de estilos

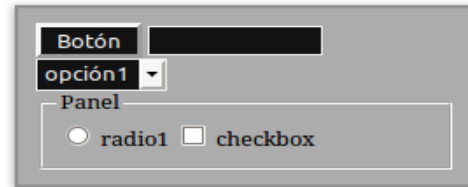
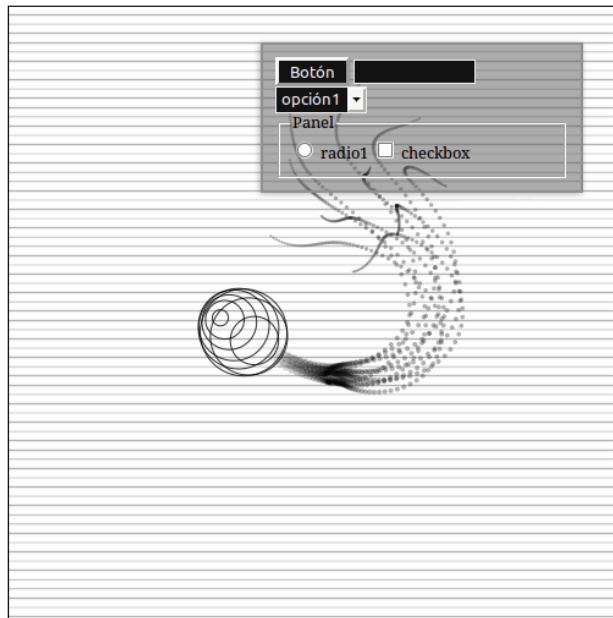
Una captura de pantalla del mismo formulario web, pero con estilos aplicados. El formulario ahora tiene un fondo gris y los elementos tienen bordes y colores definidos.

Figura 8: Después de la hoja de estilos

Resultado



Original (Flash) :
<http://www.cristalab.com/ejemplos/inteligencia-artificial-medusa-c77l/>
- Adaptación: Elsa Balderrama & Rocio Rivera.

Figura 9: Resultado del proyecto

El resultado final puede verse en la siguiente URL:http://www.papayaplayground.net.au.net/O2Graf_Proyecto1_Equipo6/

La Figura 9 muestra la apariencia del proyecto, aunque el formulario puede aparecer fuera del recuadro si hay suficiente espacio en la ventana.

Recapitulando, se han aplicado en el proyecto los siguientes conocimientos:

- Dibujo de figuras por medio de librerías.
- Personalización de elementos UI.
- Animación de figuras.
- Dibujo procedural.
- Proyección de objetos 3D en coordenadas 2D.
- Transformaciones elementales.
- Formatos de imagen.

Recomendaciones

La graficación por computadora da muchas posibilidades, pero para hacer cosas más complejas que dibujar figuras básicas es recomendable practicar ciertos paradigmas de programación (en este caso la POO) que faciliten el modelado de los problemas. Para estudiar el tema se sugiere también tener un total entendimiento de los elementos que se están usando y llevar a cabo las mejores prácticas de programación posibles, no solo para mejorar la comprensión del código sino para evitar la saturación de los recursos del hardware.

Conclusiones

Existen muchos ejemplos de que algo de creatividad se pueden graficar cosas muy interesantes y llamativas, pero para eso la graficación se apoya de otras disciplinas, principalmente de las matemáticas y es un buen campo de aplicación de éstas. La realización de este proyecto llevó al reforzamiento de los conocimientos adquiridos a lo largo del parcial, de manera condensada y aplicable a un mismo fin.

Referencias

1. Blanco Ricardo (Enero 2013). Plan de curso (Graficación). URL: <http://fing aulas.uach.mx/mod/resource/view.php?inpopup=true&id=21748>. Consultado el 21 de Febrero del 2013.
2. Del Razo Cinthia , Romero Jonathan, Herrera Marco, Bonilla Osiel (7 de Febrero del 2013). Formatos Gráficos de almacenamiento. URL: http://fing aulas.uach.mx/file.php/339/moddata/forum/1406/26505/02Graf_Equipo5_ExposicionTema3.pdf. Consultado el 23 de Febrero del 2013.
3. Inteligencia Artificial: Medusa. Cristalab (20 de Marzo del 2005). URL: <http://www.cristalab.com/ejemplos/inteligencia-artificial-medusa-c77l/>. Consultado el 21 de Febrero del 2013.
4. WHATWG (25 de Febrero del 2013). The Canvas Element. URL: <http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#the-canvas-element>. Consultado el 21 de Febrero del 2013.