Capítulo 3 Herramienta de Simulación

En este capítulo se muestra de manera general el funcionamiento de la herramienta de simulación. Se explican brevemente los pasos realizados para desarrollar el simulador, así como ejemplos y explicación de los conceptos de *línea del tiempo*, *casts, Actionscripts* y otros conceptos que se manejan *Flash y Director.*

3.1 ¿Cómo se hizo?

Para desarrollar el simulador de algoritmos se tuvo que buscar una herramienta que fuera muy poderosa tanto en programación como en animación. Así que se opto por utilizar las herramientas *Flash MX y Director MX* de *Macromedia*. Ambos programas son parte de una familia de programas multimedia que combinando sus cualidades son una herramienta muy poderosa. Las Figuras 3.1 y 3.2 muestran impresiones de pantalla de *Flash* y de *Director.*

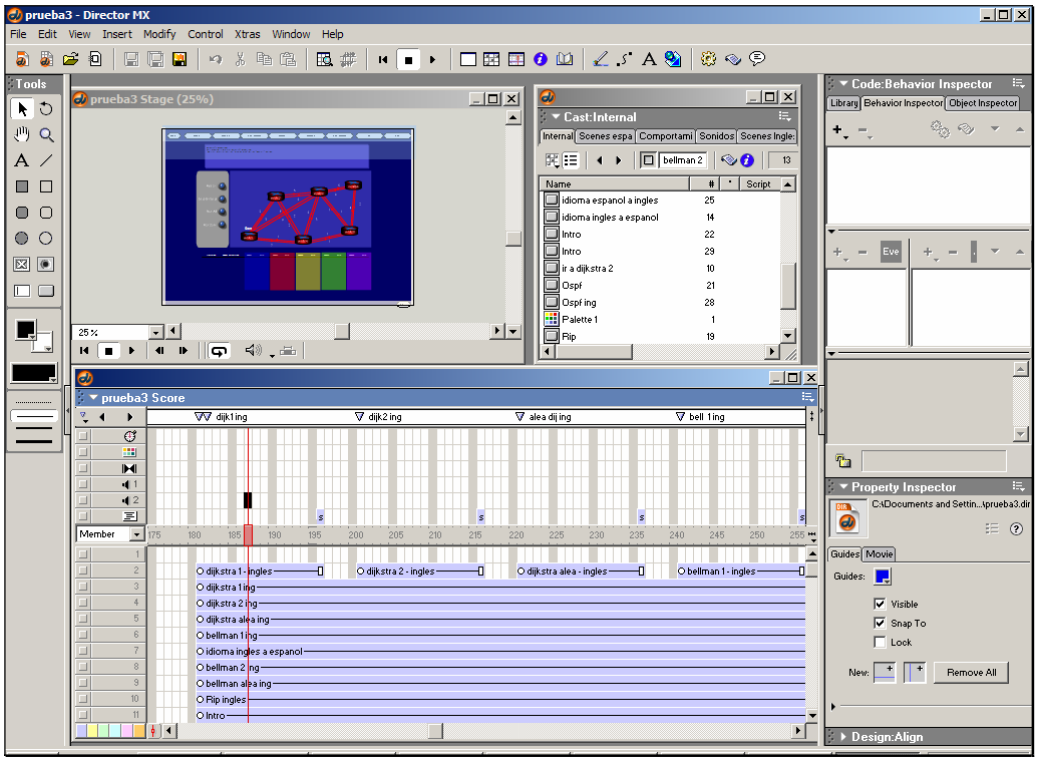

Figura 3.1 Impresión de Pantalla de Director MX

Combinando el poder que ofrece *Actionscripts* de *Flash* para hacer la programación de los algoritmos y la versatilidad que posee *Director* para realizar ambientes gráficos y animaciones, fue posible obtener la herramienta adecuada para realizar el *Simulador de algoritmos Dijkstra y Bellman-Ford.*. El trabajo fue dividido en dos etapas: la etapa de programación de los algoritmos y la etapa de compilación y unificación de los algoritmos. Es en esta última etapa donde se creo la interfase final hacia el usuario.

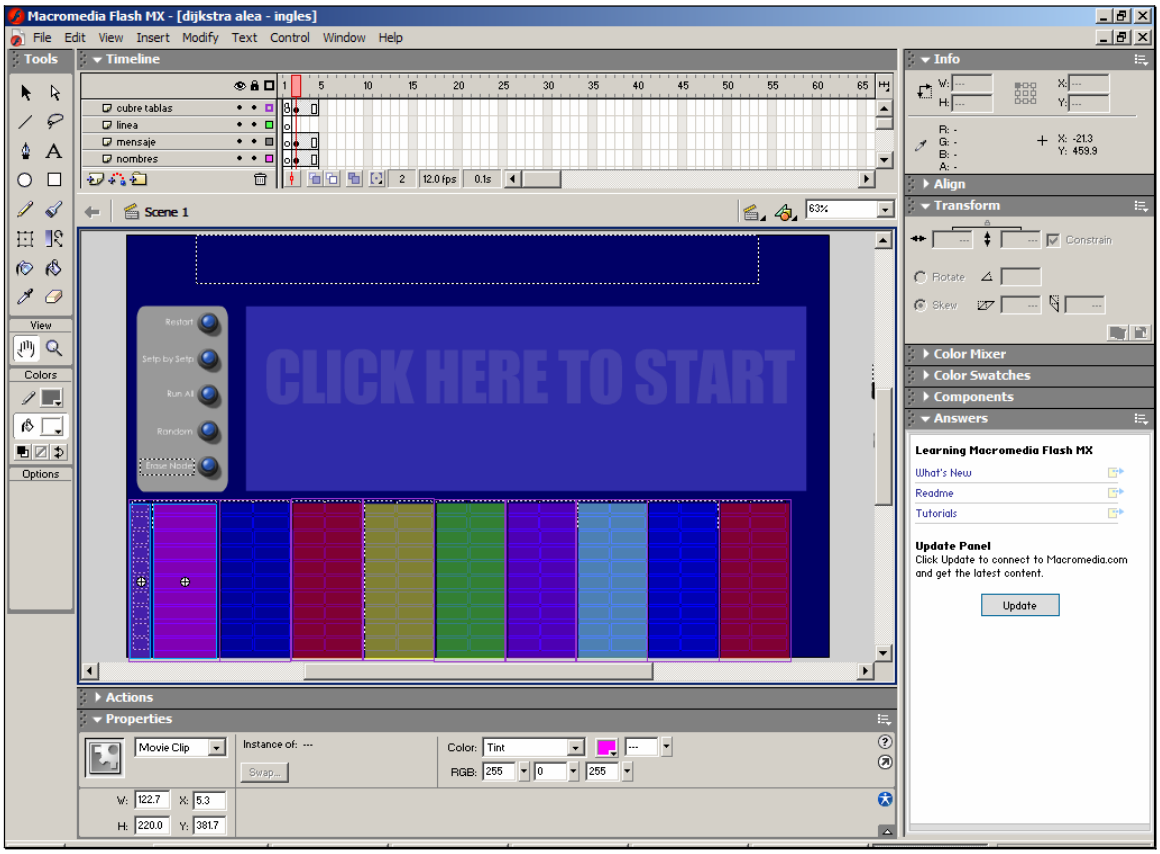

Figura 3.2 Impresión de Pantalla de Flash MX

3.1.1 Etapa de Programación

Antes de comenzar a programar los algoritmos se creo una librería gráfica en *Flash MX* que contuviera los elementos utilizados como los *routers,* los enlaces, los botones, tablas dinámicas de datos, punteros del ratón, etc.

La mayoría de los elementos son creados en una línea de tiempo que es controlada por código escrito en *Actionscripts*. Veamos un ejemplo de cómo se crearon los *routers.* Primero se dibujan los *routers*. En la línea de tiempo se le asigna etiquetas que serán

llamadas por el código dependiendo del estado que se quiera mostrar del *router*. Recordemos que si el *router* se encuentra en estado inicial éste se muestra de color rojo, cuando el *router* recibe un paquete cambia de estado rojo a un estado azul y finalmente cuando ha encontrado el camino final el nodo se presenta en color verde. Así en la misma línea de tiempo pero en distintas instancias de tiempo se dibujaron los *routers* de distintos colores y con distintas etiquetas. Las Figuras 3.3 a 3.5 muestran la línea de tiempo de los *routers* y sus etiquetas.

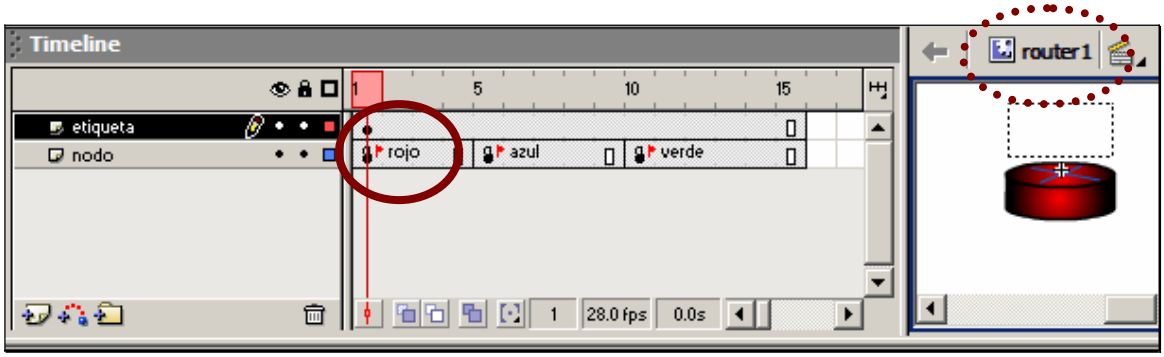

Figura 3.3 Línea del tiempo para router en rojo

De esta manera tenemos un elemento en la librería de elementos llamado *router1* con etiquetas distribuidas en el tiempo llamadas *rojo, azul,* y *verde.* Una vez creados los *routers* veamos como son llamados por el código para que la línea del tiempo cambie y muestre el *router* en el color deseado según el algoritmo se va simulando. En la Figura

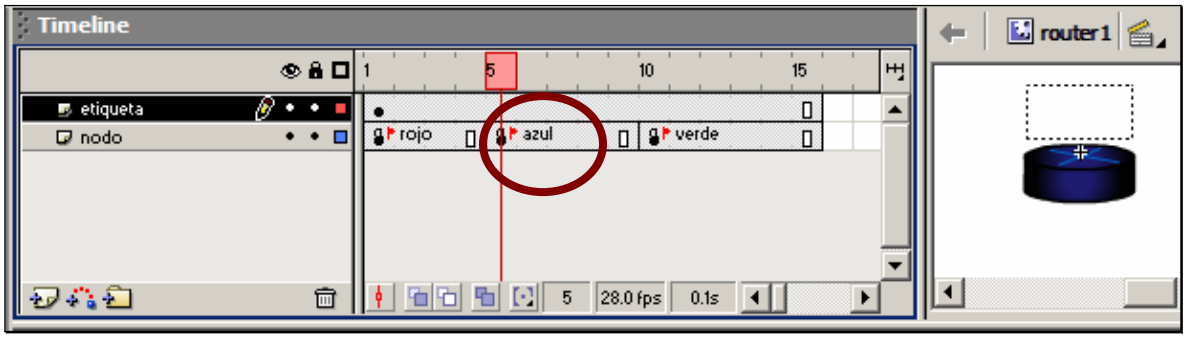

Figura 3.4 Línea del tiempo para router en azul

3.6 se muestra un fragmento del código en *Actionscripts*. Como podemos apreciar en las líneas 154 y 165 se hace un llamado al *router* a cambiar de estado. En la línea 154 se le indica que cambie del estado en el que se encuentra, al estado *verde*. Esto se logra con el

comando *gotoAndStop*. Internamente le esta diciendo que vaya a la etiqueta nombrada *verde* y detenga la línea del tiempo. Así logramos que el *router* cambie de color y se quede de color verde hasta que se desee cambiar de color nuevamente. En la línea 165 hace la misma acción pero en esta ocasión cambia a color azul.

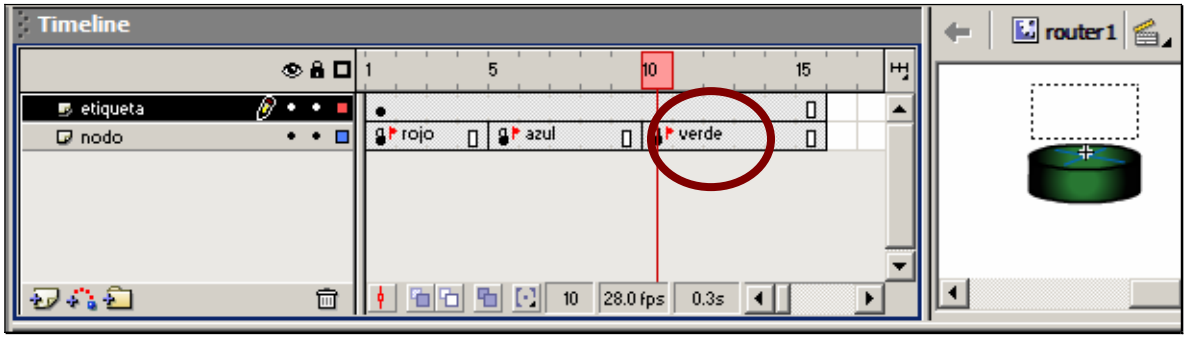

Figura 3.5 Línea del tiempo para router en verde

Como habrá notado, antes del comando *gotoAndStop* se encuentra entre corchetes la palabra *[routercin]* precedido por el comando *_root*. *Routercin* en este caso es un apuntador que nos indica que *router* es al cual deseamos cambiar de color. En la línea 162 verá que estamos asignando a la variable *routercin* la concatenación de la palabra *router* y una variable *z*. La variable *z* cambia constantemente mientras se simula el algoritmo, de esta manera la variable *z* varía entre 1 y 6 (nueve en el caso donde el usuario crea su propia topología). Así con solo modificar la variable *z* podemos generar los nombres de los *routers.* Después de concatenar la palabra *router* con la variable *z* tenemos: *router1, router2,… router6.* Estos nombres corresponden a los nombres de los *routers* creados en las librerías. De esta manera usando apuntadores podemos cambiar los colores de los nodos de una manera dinámica.

Combinando código de *Actionscripts* y manipulando adecuadamente la línea de tiempo fue como se generaron las animaciones utilizadas en el simulador. Una vez creadas las librerías de todos los elementos utilizados se procedió a la programación de los algoritmos.

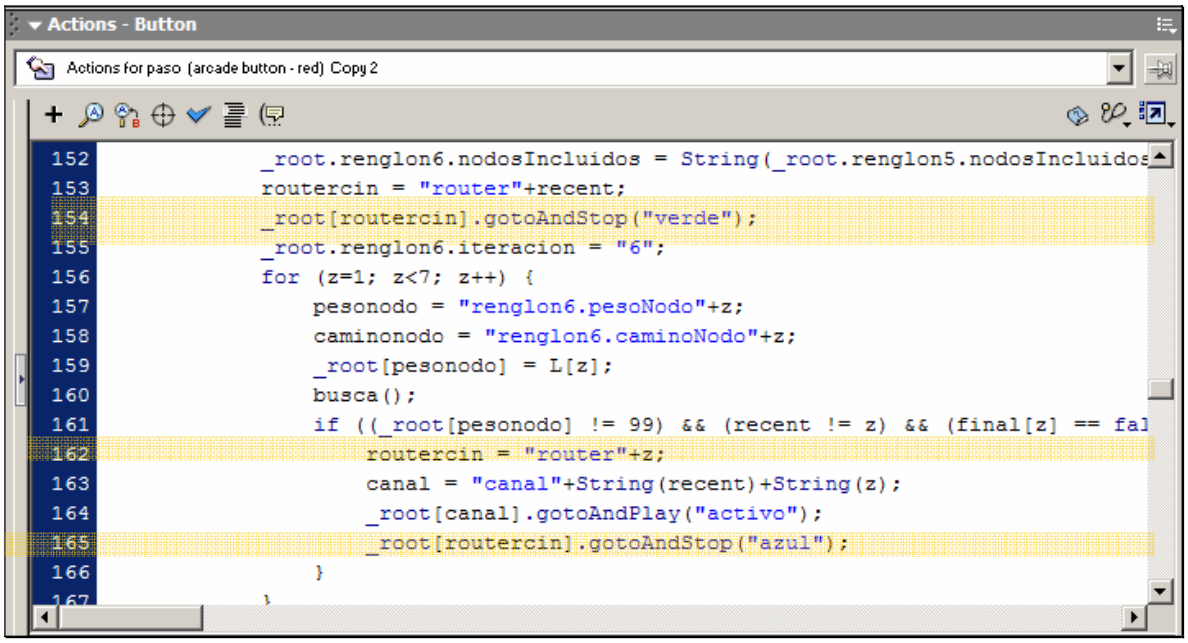

Figura 3.6 Fragmento de código de Actionscripts.

La programación de los algoritmos fue realizada con un lenguaje de programación orientado a objetos llamado *Actionscripts*. *Actionscripts* posee comandos muy similares a los comandos utilizados en *C++* como se puede apreciar en la Figura 3.6. Los algoritmos se tradujeron de pseudocódigo encontrado en la literatura a código compatible con *Actionscripts*. Además de traducir el código fue necesario agregar al pseudocódigo muchas líneas de código que permitieran la animación de cada uno de los elementos como se mostró en el ejemplo anterior. También se tuvo que desarrollar un algoritmo para encontrar el camino ya que los pseudocódigos encontrados en la literatura se limitan a dar el peso hacia cada nodo más no calculan el camino seguido.

3.1.2 Etapa de Compilación y Unificación

Una vez programadas todas las topologías y finalizadas las animaciones se recurrió de *Director* para compilar los elementos creados en *Flash.* Gracias a su compatibilidad *(ambos son de la familia Macromedia)* los archivos generados en *Flash* son importados e insertados en *Director* en librerías llamadas *Casts.* De esta manera se importaron las topologías creadas en *Flash.* Una vez importadas se crearon librerías para almacenar los elementos principales del menú. Y se agregó código muy sencillo a cada elemento del menú para que al ser accionados *corriera* los elementos almacenados en el *Cast*.

A diferencia de *Flash* que solamente permite crear archivos que son leídos en *Flash* o por medio de unos elementos llamados *plug-in, Director* nos permite crear un archivo cien por ciento ejecutable. A su vez nos permite crear elementos compatibles con Internet para su difusión a través de Internet, y dado que el objetivo de este proyecto tiene fines educativos se crearon ambas versiones: una compatible con Internet para ser accesado a través de Internet por medio de un *plug-in* y la versión ejecutable que no necesita ningún programa para ser ejecutado.