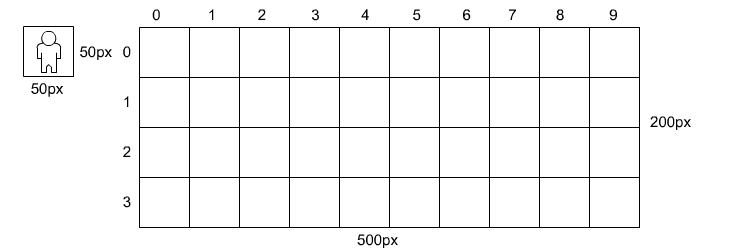
**Ejemplo del uso de Imágenes y TDA con SDL**

Tenemos un TDA llamado Persona el cual queremos dibujar en un punto de la pantalla y luego desplazarlo por ella. Para eso primero tenemos que tener en claro como es el escenario en el que se moverá y cómo está compuesto el TDA. Para este ejemplo vamos a suponer que:

1. El escenario está compuesto por una matriz de f (filas)\*c (columnas) donde cada casillero mide 50x50 pixeles.
2. La matriz se empieza a dibujar en el punto superior izquierdo de la pantalla (punto 0/0) y termina en el punto inferior derecho (punto f/c) por lo que podemos suponer que la medida de la pantalla es de 50\*f de alto y 50\*c de ancho.
3. El personaje (Persona) ocupa un espacio de 50x50px y tiene las coordenadas de f/c en las que se debe dibujar.

La imagen muestra un ejemplo de una matriz de 4 filas \* 10 columnas:



Ahora veamos los componentes de Persona TDA:

La primer parte guarda las coordenadas en las que se encuentra, su ancho y su alto.

typedef struct{

int f;

int c;

int anchoCasillero;

int altoCasillero;

SDL\_Texture \*imagen;

SDL\_Rect rectImag;

}Persona;

La segunda parte guarda un puntero a una imagen que posteriormente cargaremos (PNG) y un rectángulo donde se va a dibujar esa imagen.

Hasta acá todo muy simple, lo siguiente que sabemos sobre TDA es que tenemos que construir ésta estructura y luego setearla pero también (como es un juego en SDL) tendremos que dibujarla y por eso hay otra consideración:

1. si la imagen que vamos a mostrar es siempre la misma (o sea sin animación) entonces debemos **setearla una sola vez** y luego dibujarla las veces que sea necesaria.
2. Caso contrario si la imagen que vamos a mostrar cambia (hay animación) entonces debemos **setear la imagen cada vez** que queramos dibujarla.
3. Una tercera opción sería crear dentro del tda un arreglo con una lista de url que apunten a las imagens, pero no vamos a desarrollar este ejemplo.

Sabiendo esto vamos a tomar el caso 1 donde la imagen no cambia, entonces tendremos tres funciones:

1. CrearPersona();
2. DibujarPersona();
3. DestruirPersona();

Pasamos a explicar cada una:

1. CrearPersona:

void crearPersona(Persona &persona, int f, int c, int anchoCasillero, int altoCasillero, SDL\_Renderer\* renderer){

persona.f=f;//coordenada logica y

persona.c=c;//coordenada logica x

persona.imagen=IMG\_LoadTexture(renderer,”imag/persona.png”);

persona.rectImag.y=f\* altoCasillero;//coordenada de dibujo y

persona.rectImag.x= c\* anchoCasillero;//coordenada de dibujo x

persona.rectImag.w= anchoCasillero;//ancho

persona.rectImag.h= altoCasillero;//alto

}

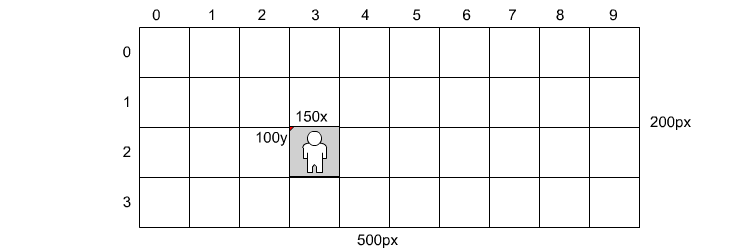
IMG\_LoadTexture(renderer,URL); es una función de la Liberia <SDL\_image.h> que devuelve un puntero hacia la imagen indicada en el parámetro URL (en nuestro caso ”imag/persona.png” entendiendo que imag es una carpeta que almacena el archivo persona.png y que se encuentra en la raíz del proyecto).

La relación que hay entre persona.imagen y persona.rectImage se realizará posteriormente en la función dibujarPersona().

Luego cargamos las coordenadas de aparición del rectángulo y acá hay algo interesante, preguntémonos: ¿por qué rectImag.y es igual a f\*altoCasillero? Y ¿por qué rectImag.x es igual a c\*anchoCasillero? Con el siguiente ejemplo se entiende fácil: supongamos que Persona esté en la coordenada f=3, c=2. El rectángulo quedará así:

persona.rectImag.y=3\* 50; //=150 coordenada de dibujo y

persona.rectImag.x= 2\* 50; //=100 coordenada de dibujo x



Al multiplicar las coordenadas con el ancho o el alto obtenemos la ubicación en píxeles que coincide con la matriz ya que tanto el cada casillero mide lo mismo que el Persona.

1. DibujarPersona:

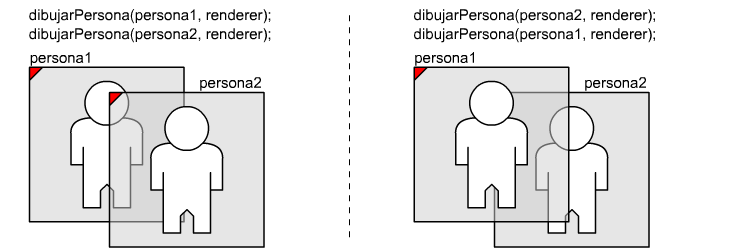
void dibujarPersona(Persona &persona, SDL\_Renderer\* renderer){

SDL\_RenderCopy(renderer, persona.imagen,NULL,&(persona.rectImag));

}

SDL\_RenderCopy es una función de la Liberia <SDL\_image.h> que plancha en el renderer (que sería nuestro lienzo de dibujo) la imagen pasada por parámetro en el rectángulo también pasado por parámetro.

Es importante destacar que podemos controlar la superposición de imágenes si estamos atentos al orden en el que llamamos esta función. Es útil tenerlo en cuenta cuando queremos dibujar por ejemplo el suelo y luego algún otro tda por sobre él, o cuando queremos superponer dos personajes, todo depende del orden en el que recorremos los elementos a dibujar:



1. DestruirPersona:

void destruirPersona(Persona &persona){

SDL\_DestroyTexture(persona.imagen);

}

SDL\_DestroyTexture libera la memoria utilizada para cargar la imagen png que se encuentra dentro de persona.imagen.

Miremos un poco el main.cpp para ver cómo interactúan estas tres funciones:

int main(int argc, char\*\* argv) {

/\*Datros que leo desde un archivo\*\*\*\*\*\*\*\*\*\*\*\*/

int filas = 8;int columnas = 10; int anchoCasillero = 50; int altoCasillero = 50;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int anchoVentana = columnas\*anchoCasillero;

int altoVentana = filas\*altoCasillero;

if(SDL\_Init(SDL\_INIT\_EVERYTHING)>=0){

SDL\_Window \*window;

SDL\_Renderer \*renderer;

window = SDL\_CreateWindow("Ejemplo Juego Persona", SDL\_WINDOWPOS\_CENTERED,

SDL\_WINDOWPOS\_CENTERED, anchoVentana, altoVentana,

SDL\_WINDOW\_RESIZABLE | SDL\_RENDERER\_PRESENTVSYNC);

renderer = SDL\_CreateRenderer(window,-1,SDL\_RENDERER\_ACCELERATED);

IMG\_Init(IMG\_INIT\_PNG);

Persona persona;

crearPersona(persona,2,3,anchoCasillero,altoCasillero,renderer); //aparece en el cuadrante 2,3

while(true){

//aquí según los eventos de teclado podríamos cambiar las coordenadas de persona… tarea para la casa ;)

SDL\_RenderClear(renderer);//borro el renderer

dibujarPersona(persona,renderer);

SDL\_RenderPresent(renderer);// dibuja el renderer, aplica todos los cambios que se hicieron en dibujarPersona()

SDL\_Delay(500);//milisegundos

}

destruirPersona(persona);

SDL\_DestroyRenderer(renderer);

SDL\_DestroyWindow(window);

IMG\_Quit();

SDL\_Quit();

}

getch();

return 0;

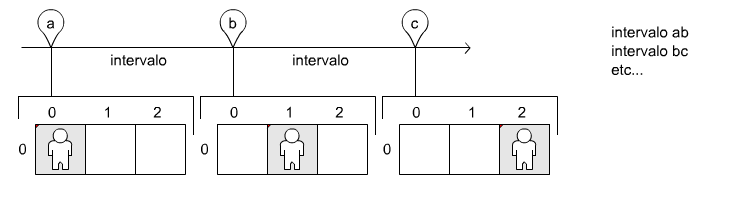
}

Éste es un ejemplo simplificado de un juego. Podríamos agregar una variable que nos permita salir del bucle while, variables donde almacenar la dirección en la que nos movemos según los eventos de teclado, y una función que realice el desplazamiento de Persona en el transcurso de un intervalo. Para desarrollar esto hay que entender los siguientes conceptos:

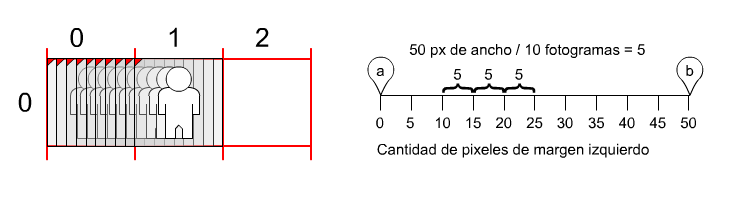
Fotograma: imagen única que forma parte de una secuencia de imágenes.

Animación: secuencia de fotogramas ordenados. Cada fotograma en la secuencia se mantiene visible durante un tiempo determinado hasta ser remplazados por el siguiente. Por lo general se indica en cantidad de fotogramas por segundo. Una animación fluida contiene entre 24 y 26 fotogramas por segundo o fps.

Intervalo: cantidad de tiempo transcurrido entre un suceso y otro. A fines prácticos diremos que un intervalo es el tiempo empleado para que nuestro personaje se desplace entre un casillero y otro.



En el juego nosotros tendremos que decidir cuánto tiempo va a transcurrir entre un punto y otro del intervalo (o sea cuanto va a durar nuestro intervalo) y cuantos fotogramas van a aparecer dentro de ese intervalo. Ya establecimos que nuestro personaje no va a tener animación pero si va a tener un desplazamiento. Para continuar el ejemplo diremos que en un intervalo habrá diez fotogramas y durará un segundo.



En la imagen vemos que dentro de un intervalo de un segundo actualizo la posición de la imagen diez veces, y que en cada actualización le agregamos una cantidad de pixeles a la coordenada x del rectángulo donde se dibuja la persona: para la primera vez le agrego 5, para la segunda vez 10, la tercera 15 etc… hasta los 50px.

Esto está acompañado de un algoritmo que dice:

Void correrHaciaLaDerecha(Persona &persona, int subIntervalo){

persona.rectImag.x= (persona.c\* 50) + ((50/10)\*subIntervalo) ;

}//50 es el ancho del casillero, y 10 es la cantidad de imágenes del intervalo.

Void reubicamosPersonaALaDerecha (Persona &persona){

persona.c=persona.c+1; //o persona.c++;

}

Int subIntervalo = 0; //valor de 0 a 10;

while(true){

If(subIntervalo == 10){

subIntervalo = 0;

reubicamosPersonaALaDerecha(persona);

}

correrHaciaLaDerecha(persona,subIntervalo);

SDL\_RenderClear(renderer);//borro el renderer

dibujarPersona(persona,renderer);

SDL\_RenderPresent(renderer);

SDL\_Delay(100);

// si multiplico 100 \* 10 = 1000 milisegundos = 1 segundo

//que es lo mismo que decir 10 imágenes por segundo…

subIntervalo++;

}

En este caso solo ejemplificamos el movimiento hacia la derecha. Depende de ustedes como resuelven el algoritmo para contemplar los desplazamientos hacia las cuatro direcciones controladas por los eventos de teclado, y contemplando situaciones del escenario cuando el subintervalo sea igual a 0 (por ejemplo el ataque de los villanos, las colisiones del personaje con los límites del tablero, o con otros elementos).