NGT

Neopontec Gaming Toolkit

For toolkit version 0.2

TUTORIAL 1

First steps

Hector Blanco de Frutos
hectorblanco@neopontec.com

1. A little introduction

The Neopontec Gaming Toolkit is a set of classes, functions and data types that can be used to automate or simplify some tasks needed to develop a game using SDL.
The main idea of the toolkit is to offer objects that can be used by developers to made some tasks that are used very often in a game or a multimedia application, such as drawing graphics and texts in the screen. The toolkit also offers support for drawing and managing sprites, taken and evolved from an excellent SDL tutorial made by Marius Andra in his Cone3D web site.

2. Starting the toolkit

First of all, you must specify the main attributes in order to initialize the toolkit properly. This is done by creating an object of the class *NGT_Engine*.
These are the attributes that you need to specify:

machine_target	The target machine in which will work the game or the application. In the current version of the toolkit, only *NGT_MACHINE_GENERIC* (PC's, Mac's etc) and *NGT_MACHINE_GP2X* (Gamepark Holdings GP2X media player) are supported.
video_bpp	Screen depth. Bits per pixel.
video_res_w	Horizontal screen resolution (width)
video_res_h	Vertical screen resolution (height)
video_fullscreen	Enable or disable full screen (0 or 1)
audio_frequency	Sampling frequency for the sound system.
audio_buffer	Data buffer that will be used for storing sounds and music.
audio_mode	SDL sound modes.

When *NGT_MACHINE_GP2X* is specified as *machine_target*, only the following attribute have effect: *audio_frequency* and *audio_mode*. The others are automatically specified by the toolkit.

After specifying the attributes, the toolkit must be initialized. This is done by calling the function *initialize()* from the NGT_Engine class object we created.

Example:

```
NGT_Engine main_engine;

main_engine.machine_target=NGT_MACHINE_GENERIC;
main_engine.video_bpp = 24;
main_engine.video_res_w = 800;
main_engine.video_res_h = 600;
main_engine.video_fullscreen = 0;
main_engine.audio_frequency = 44100;
main_engine.audio_buffer = 2048;
main_engine.audio_mode = AUDIO_S16SYS;
main_engine.initialize(screen);
```


After doing this, we can now draw to the screen, play sounds ...etc.

3. Managing and drawing surfaces

A surface is a SDL data type which stores the information of an image, a representation of a text ...etc. needed to draw it on screen.
The *NGT_Surface* class is a step over the *SDL_Surface* data type. It stores the image or text information, plus some important information and attributes such as the coordinates, and some methods to draw and manage this surface.
These are the attributes of the class:

x	Horizontal coordinate of the surface, in pixels.
y	Vertical coordinate of the surface, in pixels.
surface	SDL_Surface* object where the image or the text is stored.

Here you have an example which shows how to load an image from the hard drive, and draw it onto the screen.

Example:

```
NGT_Surface surface1;

surface1.x = 32;
surface1.y = 100;
surface1.LoadIMG(“images/test_image.png”);
surface1.draw(screen);
SDL_Flip(screen);
surface1.free();
```

The *LoadIMG(char*)* method loads an image from the file system specified by a char array, and stores it information into memory to manage it.

The *draw(SDL_Surface*)* draws the stored surface of the class into another surface. Usually this *other* surface is the main application surface, the working screen). To update the the application screen we must call to the *SDL_Flip(SDL_Surface*)* function from the SDL API.

When we are sure that we will not use anymore the surface, we must call the *free()* method from the *NGT_Surface* class. Doing this, the memory used by the surface will be freed in order to be used by other things of the application.

4. Managing and drawing texts

Dealing with texts is quite similar as doing it with surfaces. The process of managing and drawing a text into

the screen is very simple: you must load a TrueType font, provide a string, and call the drawing method.
First of all we must specify some attributes:

filename	Path and file name of the TrueType font you will use.
size	Vertical coordinate of the surface, in pixels.
surface	SDL_Surface* object where the image or the text is stored.
x	Horizontal coordinate for drawing the text.
y	Vertical coordinate for drawing the text.
color	SDL_Color attribute which defines the text color.
mode	Drawing mode that you will use, such as none or blended.

Once we have specified the attributes, we must call the *LoadTTF()* method, which will load the TrueType font and will initialize the object.
The next step is call the draw(text, surface) method to draw the text, where text is the text string we want to draw, and *screen* is the surface in which you will draw the text.

Example:

```
NGT_Font text;

// Set the attributes
text.filename = "fonts/FreeSans.ttf";
text.size=62;
text.x = 32;
text.y = 100;
text.mode = TEXT_BLENDED;
text.color = gamecolor.white;
// Load the TrueType font
text.LoadTTF();
// Draw the text
text.draw("A techdemo made by", screen);
SDL_Flip(screen);

// Clear the memory used by the text object,
// if you don't need it anymore.
text.free();
```

The last thing we must do with the font object is clear the memory that uses if we are sure that we'll not need it anymore, or if we want to load another TrueType font file into it. This is done by calling the *free()* method of the class.

5. Managing and drawing sprites

Dealing with sprites is very similar as doing it with surfaces, the difference is that you load a set of frames and draw all of them in a sequence, or only one.

The first thing we must do is declare the sprite and the spritebase. The first will be use to manage and control the sprite set, and the second is used to store the images “frames” that will compose the sprite set.
In the next step, we initialize the spritebase, loading the images contained into a specified directory, and later we initialize the NGT_Sprite class, with the *init(...)* method. The arguments of this methods are the spritebase object and the surface will be use to draw the sprites.
The next common step is setting attributes of the sprite class, such as the x,y coordinates, the first frame, and the speed.
Finally, calling the *startAnim()* method, and the *draw()* method, the toolkit will draw the appropriate frame into the screen.

Example:

```
// Declare the variables
NGT_Sprite spirals;
NGT_SpriteBase spiralsbase;

// Initialize and draw sprites
spiralsbase.init("2d/sprites/espiral");
spirals.init(&spiralsbase,screen);
// Set attributes
spirals.xset(400);
spirals.yset(200);
spirals.setFrame(0);
spirals.setSpeed(1);
// Start animation
spirals.startAnim();

int j = 0;
// Play the animation 4 times.
 while ( j < 4 ) {
      spirals.draw();
      // Update the screen
      SDL_Flip(screen);
      // Erase the screen contents
      SDL_FillRect(screen,NULL,0x010000);
      SDL_Delay(60);
      // When completing a play, grow the j.
      if ( (spirals.getFrame()+1) == spiralsbase.mNumframes ){
            j++;

      }
 }
```

6. Closing the toolkit

Before ending the game or the application we must take care on closing the toolkit, so it can switch the system video mode into its original state, and close the SDL libraries.
We do this calling the *finalize()* method of the NGT_Engine class.

Example:

```
int main(){

    NGT_Engine game_engine;

    /
          . . .
          GAME CODE
          . . .
    /

    game_engine.finalize();

    return 0;
}
```

Once this is done, the game or the application can be closed.