Introduction to Game Programming

Session 4 – Adding sound and texts

Nacho Cabanes
Nacho Iborra

IES San Vicente

Index of contents

1. Introduction

In this session we are going to add two minor (but important) aspects of the video game, such as music/sound effects, and texts. What we are going to add is:

- A background music in the title screen, and another one while we are playing the game
- A sound effect when choosing a character
- Put some texts in the game screen to show player energy and points

2. Adding texts

Texts are a very important component in videogames, since they can show relevant information, such as instructions of how to play, or scores. If we want to add texts to our Tao SDL video game, let's assume that we are going to use TTF fonts (the most common ones). Then, we need to add two libraries to the project: **SDL_ttf.dll** and **libfreetype-6.dll.** Both libraries should be added by now to our project, so we should not have to care about them.

Besides, we need to add to the project the TTF file(s) of the font(s) that we want to use. We can find some of them in *Windows/Fonts* folder (if we are using windows) or in some web pages such as *www.fontsquirrel.com*. You will be provided with a specific font for the video game.

2.1. Create the Font class

As we did for images, we are going to create our own *Font* class to deal with all the SDL methods required to load and draw texts from a given font. This class would have an *IntPtr* attribute to store the font, and a constructor to load the font from the corresponding file, with the desired size.

```
using System;
using Tao.Sdl;

class Font
{
    IntPtr fontType;

    public Font(string fileName, int fontSize)
    {
        fontType = SdlTtf.TTF_OpenFont(fileName, fontSize);
        if (fontType == IntPtr.Zero)
        {
            Console.WriteLine("Font type not found");
            Environment.Exit(2);
        }
    }

    public IntPtr GetFontType()
    {
        return fontType;
    }
}
```

2.2. Changes in Hardware class

Next step consists in adding a new method to our *Hardware* class. We will call it *WriteText*, and it will have the following parameters:

- The text to be written
- The coordinates (x, y) where the text must start
- The text color (in RGB components, separately)
- The font to use (a *Font* object)

```
public void WriteText(string text, short x, short y, byte r, byte g, byte b,
                      Font fontType)
{
    Sdl.SDL_Color color = new Sdl.SDL_Color(r, g, b);
    IntPtr textAsImage = SdlTtf.TTF_RenderText_Solid(fontType.GetFontType(),
        text, color);
    if (textAsImage == IntPtr.Zero)
        Environment.Exit(5);
    Sdl.SDL_Rect src = new Sdl.SDL_Rect(0, 0, screenWidth, screenHeight);
    Sdl.SDL_Rect dest = new Sdl.SDL_Rect(x, y, screenWidth, screenHeight);
    Sdl.SDL_BlitSurface(textAsImage, ref src, screen, ref dest);
}
```

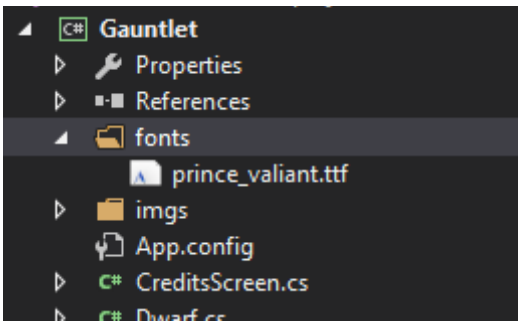
We are not going to see this method in detail, but it takes the text and generates an image with it (*textAsImage* variable). Then it prints it in the specified coordinates. We specify *screenWidth* and *screenHeight* as text width and height because we don't know which width and height it is going to have. This way, we make sure that the text will be completely shown in the screen (unless it is bigger than the screen itself).

There is one more change pending in *Hardware* class. At the end of the constructor, we must call the TTF_Init method to init the font management.

```
public Hardware(...)
{
    ...
    SdlTtf.TTF_Init();
}
```

2.3. Downloading and installing the font

As we said before, you will be provided with a file called “prince_valiant.ttf”. Create a subfolder called “fonts” in your project, and place this file inside the folder. Remember to set the “Copy always” property so that this file will be copied to the output folder.



2.4. Testing the font

Let's go to our *GameScreen* class. We define an attribute of type *Font,* and initialize it at the constructor.

```
class GameScreen: Screen
{
    MainCharacter character;
    int chosenPlayer;
    Font font;
    ...
    public GameScreen(Hardware hardware): base(hardware)
    {
        font = new Font("fonts/prince_valiant.ttf", 20);
    }
```

Then, add these lines in step #1 of the game loop to draw the “ENERGY” text at the bottom left corner, in red.

```
// 1. Draw everything
hardware.ClearScreen();
hardware.DrawSprite(Sprite.SpriteSheet, …);
hardware.WriteText("ENERGY:", 5, 550, 255, 0, 0, font);
hardware.UpdateScreen();
```

3. Adding sounds

We can also add sound effects and/or background music to our video games. To do this, we must add **SDL_mixer.dll** library to our project (you should have done this by now). As we did for texts, we are going to start by creating our own *Audio* class to work with, and use it in our main application.

3.1. The Audio class

Create a class called *Audio* with two attributes: a list of audios (*IntPtr*) that will be added to the project at each moment, and the number of channels that we are going to manage.

```
using System;
using System.Collections.Generic;
using Tao.Sdl;

class Audio
{
    List<IntPtr> audios;
    int channels;

    public Audio(int freq, int channels, int bytesPerSample)
    {
        this.channels = channels;
        SdlMixer.Mix_OpenAudio(freq, (short)SdlMixer.MIX_DEFAULT_FORMAT,
                               channels, bytesPerSample);
        audios = new List<IntPtr>();
    }
}
```

In the constructor, we specify the audio features: frequency (typically 22050 or 44100), number of available channels and number of bytes per audio sample (typically 4096).

Then, we are going to add two methods to add WAV files to the list and play them, respectively:

```
public bool AddWAV(string fileName)
{
    IntPtr file = SdlMixer.Mix_LoadWAV(fileName);
    if (file == IntPtr.Zero)
        return false;
    audios.Add(file);
    return true;
}

public void PlayWAV(int pos, int channel, int numberOfLoops)
```

```
{
    if (pos >= 0 && pos < audios.Count && channel >= 1 && channel <= channels)
        SdlMixer.Mix_PlayChannel(channel, audios[pos], numberOfLoops);
}
```

Regarding *PlayWAV* method, it plays the audio at position *pos* of the list, in the specified channel (a number between 0 and total number of channels), and the number of times indicated in the last parameter (-1 to infinite loop, 0 to play it once, 1 to play it twice and so on).

WAV is a format very easy to deal with. It allows us to determine which channel must be assigned to each audio. However, we can also play other audio formats, such as OGG, MID or MP3, but we need to add another less specific methods:

```
public bool AddMusic(string fileName)
{
    IntPtr file = SdlMixer.Mix_LoadMUS(fileName);
    if (file == IntPtr.Zero)
        return false;
    audios.Add(file);
    return true;
}

public void PlayMusic(int pos, int numberOfLoops)
{
    if (pos >= 0 && pos < audios.Count)
        SdlMixer.Mix_PlayMusic(audios[pos], numberOfLoops);
}
```

The main difference between these methods and the WAV ones is that with these generic methods we can't specify the channels for each audio file. We can use them to play background music, if we want to.

We can also add some methods to stop a sound whenever we want to:

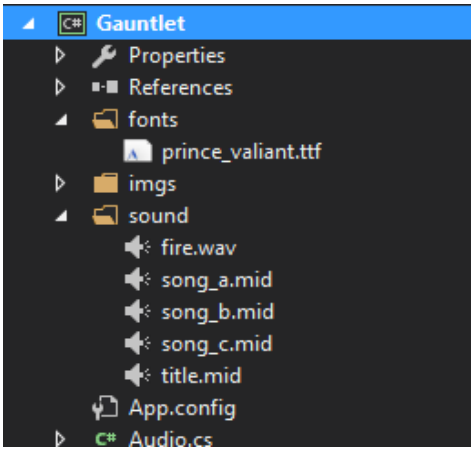
```
public void StopMusic()
{
    SdlMixer.Mix_HaltMusic();
}

public void StopChannel(int channel)
{
    SdlMixer.Mix_HaltChannel(channel);
}
```

The first one will be used to stop all the sounds started with *PlayMusic* method, and the second one will stop the sound played with *PlayWAV* method in a given channel.

3.2. Downloading and adding the audios

You will be provided with a set of audio files. Create a folder called “sound” in your project and copy these files inside. As usual, remember to set the “Copy always” property.



3.3. Testing the audios

Let's go to our *WelcomeScreen* class. We are going to add a background music to it, It will start playing as soon as the screen shows, and it will stop playing when we move to next screen.

First of all, we define an Audio object as an attribute, and initialize it in the constructor, loading the audio file that we are going to play.

```
class WelcomeScreen : Screen
{
    bool exit;
    Image imgWelcome;
    Audio audio;
    …
    public WelcomeScreen(Hardware hardware) : base(hardware)
    {
        exit = false;
        audio = new Audio(44100, 2, 4096);
        audio.AddMusic("sound/title.mid");
        ...
```

Finally, in the *Show* method, we start playing the audio before the *do..while* loop, and stop playing it after the loop. As we have only added one audio file, it will be the first one in the list (index 0).

```
audio.PlayMusic(0, -1);
do
{
}
while (!escPressed && !spacePressed);
audio.StopMusic();
```

4. Drawing in Tao SDL

To finish with this session, we are going to learn some basics about how to draw geometric figures with Tao SDL. What we are going to do is draw a thin, yellow line at the bottom, to separate the score text from the scene.

In order to draw these figures, we need to add *SDL_gfx.dll* library to our project. You can copy it from the resources of this session, and paste it in the root folder of the project (and set the “Copy always” property).

With this library, we have some methods available to draw some basic figures, such as rectangles, circles or polygons, with different colors. For instance, if we want to draw a line, we can use this method (add it to the *Hardware* class):

```
public void DrawLine(short x, short y, short x2, short y2, byte r, byte g,
byte b, byte alpha)
{
    SdlGfx.lineRGBA(screen, x, y, width, height, r, g, b, alpha);
}
```

The specified parameters are the X,Y coordinates to draw the line (both edges), the color in RGB format and the *alpha* element (transparency, going from 0 – completely transparent), to 255 – completely opaque).

So, if we want to draw a yellow line at the bottom of *GameScreen*, we can add this line in step #1 of the game loop:

```
// 1. Draw everything
hardware.ClearScreen();
hardware.DrawSprite(Sprite.SpriteSheet, ...
hardware.DrawLine(0, 500, 800, 500, 255, 255, 0, 255);
```

There are some other useful methods within this library, such as rectangleRGBA (to draw a rectangle, circleRGBA (for circles), filledPolygonRGBA (for polygons in general, specifying the array of X and Y coordinates)...

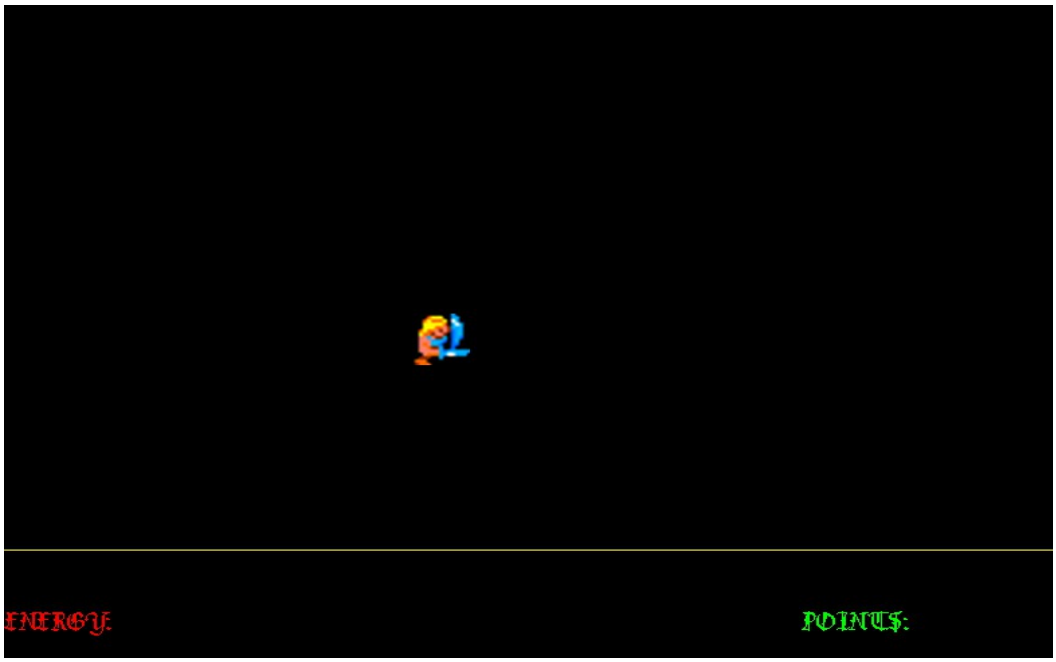
The following code draws a yellow rectangle with corners at (0, 500), (800, 500), (0, 520) and (800, 520). You can use this code instead of previous one if you want a thicker separation between the texts and the scene:

```
short[] vx = {100, 300, 300, 100};
short[] vy = {100, 100, 200, 200};
SdlGfx.filledPolygonRGBA(screen, vx, vy, vx.Length, 255, 255, 0, 120);
```

5. To think a little...

Now it is your turn. What you have to do now is:

- **(0,5 points)** Add another text in green called “POINTS”, at the bottom right corner of the game screen. Here we will show the player score in later sessions:



- **(1 point)** Add a sound effect “sound/fire.wav” to the *PlayerSelectScreen* class. It will play everytime we change the currently selected player.
- **(0,5 points)** Add a background sound (you can choose among “song_a.mid”, “song_b.mid” and “song_c.mid”) to the *GameScreen* class. This sound will start playing before the game loop, and will stop when we exit the loop.