Monogame の基本

suzusime

平成 28 年 4 月 30 日

全体の流れ

Monogame の雛型から作ったプログラムの本当の開始点（エントリポイント）は Program.cs の Main() ですが、そこで `game.Run();` が実行されたあとは、次のように Game1.cs の中の函数が順に呼ばれていきます。

まず最初に呼ばれるのは Initialize() です。この中にはゲームの最初にする必要のある、各初期化処理を書きます。次に呼ばれるのは Loadcontent() です。ここではゲームで必要な外部ファイルの読み込みをします。

そのあとは、Update() と Draw() が交互に呼ばれ続けます。ここが所謂メインループで、Update() の中でいろいろな計算をしたあとに Draw() の中でその結果を描画する、という処理を 1 フレーム（普通は 60 分の 1 秒）毎に延々と繰り返します。名に負うとおり、これがゲームの主要部分です。

そして、何らかの理由（例えばメニューから「終了」が選ばれるなど）でメインループから抜けたときは、UnloadContent() が呼ばれたあと、終了します。UnloadContent() の中では読み込んだ外部ファイルを破棄したりします。

外部ファイルの読み込み全般について

ゲームで読み込んで表示する外部ファイル（画像、音楽、フォントなど）は、Monogame から見える場所に置いておく必要があります。

基本的にはプロジェクトと同じフォルダに “Content” という名前のディレクトリ（フォルダ）を作り、その中に入れておきます。その中に更に画像用の “Graphics” という名前のディレクトリを作るなど、種類毎に整理しておくと良いでしょう。

外部ファイルの読み込みは、“Content” ディレクトリからの相対パスで行います。例えば、“Content” ディレクトリの中の “Graphics” ディレクトリの中の “character.png” という名前の画像ファイルを読み込むときは、

```
Texture2D playertexture = Content.Load<Texture2D>("Graphics/character.png");
```

のように指定します。

画像の描画

画像の描画をするためには Game1.cs に次のように幾つかの文を追加します。但し、編集しないで良い部分は省略されています。

```
public class Game1 : Game
{
      GraphicsDeviceManager graphics;
      SpriteBatch spriteBatch;
      Texture2D texture; //追加（画像を入れる変数の宣言）

      protected override void LoadContent()
      {
            spriteBatch = new SpriteBatch(GraphicsDevice);
            texture = Content.Load<Texture2D>("Graphics/character.png");//追加
                （画像の読み込み）
```

```
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            spriteBatch.Begin(); //追加（ここからspriteBatch への書き込みを開始する）
            spriteBatch.Draw(texture,Vector2.Zero,Color.White); //追加
                (spriteBatch に texture を書き込む)
            spriteBatch.End(); //追加（spriteBatch への書き込みを終える）
            base.Draw(gameTime);
        }
    }
```

Texture2D というのは、読み込んだ画像ファイル自体を入れる型（のようなもの）です。対して SpriteBatch というのは、画像ファイルを書き込む先である「画面」の型です。はじめに LoadContent() のところで Content.Load() を用いて character.png の画像データが入った Texture2D 型の実体を作り、それを毎フレーム Draw() のところで SpriteBatch の実体（実はこれも LoadContent() で作られています）に書き込むことで、画像が画面に表示されます。

音声の再生

まず、Game1.cs の最初に `using Microsoft.Xna.Framework.Audio;` と書いておきましょう。これを書くことによって音声関係の機能が使えるようになります。音声を読み込んで再生するには次のようにします。

```
    public class Game1 : Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        SoundEffect soundEffect; //追加（読み込んだ音声を入れる変数の宣言）
        SoundEffectInstance soundInstance; //追加
            (音声を再生するための実体を入れる変数の宣言)

        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
            soundEffect = Content.Load<SoundEffect>("Music/title"); //追加
                (音声の読み込み)
            soundInstance = soundEffect.CreateInstance(); //追加
                (soundEffect を再生するための実体を作る)
            soundInstance.IsLooped = false; //追加（ループするようにする）
        }

        protected override void Draw(GameTime gameTime)
        {
            if(soundInstance.State!=SoundState.Playing){ //追加
                (soundInstance が再生中でなければという条件)
                  soundInstance.Play(); //追加（soundInstanse を再生する）
            }
            GraphicsDevice.Clear(Color.CornflowerBlue);
            base.Draw(gameTime);
        }
    }
```

SoundEffect が読み込んだ音声を入れる型、SoundEffectInstanse がそれを再生するための実体を入れる型です。一つの SoundEffect の実体から複数の SoundEffectInstance の実体を作ることができます。

文字の描画

文字列の描画のためには、スプライトフォントと呼ばれる特殊なフォントファイルを作る必要があります。フォントファイルを作ったら、次のようにすることで文字を描画することができます。

```
public class Game1 : Game
{
      GraphicsDeviceManager graphics;
      SpriteBatch spriteBatch;
      SpriteFont font; //追加（フォントを入れる変数の宣言）

      protected override void LoadContent()
      {
            spriteBatch = new SpriteBatch(GraphicsDevice);
            font = Content.Load<SpriteFont>("Fonts/main");//追加（画像の読み込み）
      }

      protected override void Draw(GameTime gameTime)
      {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            spriteBatch.Begin(); //追加（ここからspriteBatchへの書き込みを開始する）
            spriteBatch.DrawString(font,"Test␣My␣Text",new Vector2(10,10),Color.
                White); //追加
                （spriteBatchにfontで書いた文字列を書き込む）
            spriteBatch.End(); //追加（spriteBatchへの書き込みを終える）
            base.Draw(gameTime);
      }
}
```

基本的には画像を描画したときと同じです。SpriteFont というのがスプライトフォントファイルを入れるための型で、その実体を使って spriteBatch に書き込むことで描画しています。

おまけ：画像、音声の形式について

コンピュータでは「デジタル」なデータしか扱うことが出来ません。自然界の連続的（アナログ）なもの（絵や音楽）は、離散的（デジタル）なモデルで出来る限り再現することで、コンピュータ上で（ある種擬似的に）扱っています。

このような連続量（データ）を離散量（データ）にする変換を一般的に **AD 変換**（analogue to digital conversion）といいます。ここでは、AD 変換とデジタルに変換された後の情報の扱い方について、代表的な例を見ていきたいと思います。

画像

現在のところ、コンピュータでの画像の扱い方の形式には、大きく分けて**ラスタ画像**と**ベクタ画像**があります。

ラスタ画像というのは、画像全体を細かい小さな正方形の枠にわけ、そのひとつひとつの色を指定することで画像を表すような画像形式です。即ち、正方形の大きさを十分細かく取ることで適切に近似できるだろうということを根拠とした手法です。AD 変換を行うのはデジタルカメラの CCD センサやスキャナです。

この小さな枠 1 つを 1pixel といいます。画像の大きさは pixel 単位で表すことができます。例えば「横 800pixel、縦 600pixel の画像」という風に言えます（全体では $800 \times 600 = 480000$ 個の pixel があり、これを 48 万画素ということがあります）。注意すべきは、これはデータとしての大きさであり、スキャンする前の画像の現実世界での大きさや画面に表示したときの画像の見た目の大きさなどは表さないということです。データの 1pixel が現実の大きさのどれほどに対応するかを表す指標として**解像度**があります。

ベクタ画像というのは、画像を数学的な曲線で表す方式です。例えば「座標 (0,4) と (3,8) を結ぶ線分を書く」とかそういう風に画像を記述します（実際は線分だけではなくベジェ曲線などの曲線が多く使われます）。画面に表示する際には、このままでは表示できませんから、計算を行ってラスタ画像に変換します（これを**ラスタライズ**といいます）。

この形式の利点はいくら拡大しても、大きさに合わせて計算するので（つまりそのための情報をもっているので）綺麗に表示されるということです。従って、写真などには適しませんが、幾何学的に表現できる図形ならばかなり正確に記述できます（とはいってもコンピュータで扱えるのは所詮有理数の一部である浮動小数点小数ですから限界はありますが)。その特性から、ロゴデザインやフォントによく用いられる形式です。

ラスタ画像に関しては、その各 pixel の色を記述する訳ですが、色をどのように、どれほど細かく記述するかという問題もあります。現在広く用いられているのは、光の強さを RGB の三原色にわけ、それぞれを 256 段階（8bit）で測定する、RGB24bit 色の形式です。このような、連続量を何段階に分割して離散化（量子化）するかを示す値を**量子化 bit 数**といいます。この場合、量子化 bit 数は 24bit です。なお、RGB の他に透過される具合（alpha channel）を 8bit 分追加した RGBA32bit 色もよく用いられます。他に、印刷目的の場合は印刷インクの 4 色（cyan, magenta, yellow, keyplate）の強さで表す CMYK 色が用いられることもあります。

このように各 pixel を 24bit なり 32bit なりで表現していくと、画像全体の情報量はかなり大きくなっていきます。そのために、たいていの場合画像は**圧縮**して保存されます。圧縮の形式は**可逆圧縮**と**不可逆圧縮**に大別されます。可逆圧縮は完全に元に戻るような形式、不可逆圧縮は完全には元に戻らないような形式です。可逆圧縮の代表例は PNG、不可逆圧縮の代表例は JPEG です。

音声

音声に関しても、画像と似たようなことが行われています。但し、ベクタ画像に相当するものが使われている例をあまり知らないので（MIDI ＋ FM 音源シンセサイザーという例がこれにあたるでしょうか)、ラスタ画像に相当するものの代表である**パルス符号変調**（PCM）を紹介します。

PCM では、まず受け取ったアナログ音声を一定時間毎に区切ります。この区切る周期を**サンプリング周波数**といいます。人間の可聴域は 20〜20000Hz とされているので、音楽 CD の場合はその 2 倍程度である 44.1kHz がサンプリング周波数として用いられています。

区切った後は、画像の時と同じように定めた量子化 bit 数に応じて音声信号の強さを数値化します。音楽 CD の場合は量子化 bit 数は 16bit です（とは言っても録音の際にはもっと高い量子化 bit 数やサンプリング周波数を用いて AD 変換していることが多く、最近はそれを利用したハイレゾ音源の音楽なるものが売られていたりしますね)。

圧縮形式には様々なものがあります。不可逆圧縮形式では MP3 が最も有名でしょう。可逆圧縮形式はあまり用いられませんが、FLAC などが代表例です。無圧縮形式である WAV が用いられることも結構あります。なぜ無圧縮形式が用いられるかといいますと、圧縮されたデータを再び元の形式に戻すためには計算が必要になり、端的に言って「重い」からです。特にゲームなど、不要な計算を減らしたい用途では、多少の容量の大きさには目をつむって無圧縮形式を用いることで負荷を減らすことを優先することがままあります。

動画

動画は画像と音声の複合で出来ていますから、その複合技で AD 変換することが出来ます。画像のほうに関しては映画フィルムの原理と同じく、30 分の 1 秒に 1 回とかそういった一定周期ごとに静止画を記録することで動画を再現しています。

動画で大きな問題になるのはそのデータ量の多さですから、様々な圧縮形式があります。その詳細には触れませんが、注意するべき点として、圧縮形式（コーデック）とコンテナという二つの概念の存在です。圧縮形式というのはデータ圧縮の方法（アルゴリズムなど）のことであり、コンテナというのは動画全体をまとめる際のデータ保管の方法です（拡張子はコンテナに依存します)。例えば、H.264 という圧縮形式の動画は mp4 や flv などのコンテナに格納されます。

上では触れませんでしたが、実は画像や音声にも圧縮形式とコンテナの区別はあり、例えば拡張子 bmp のファイルは、BMP 形式のコンテナですが、その中の画像の圧縮形式は非圧縮の他にも RLE のものや PNG のものがあります。

但し、画像、動画、音声のいずれにしても、コンテナと圧縮形式の組み合わせは多数になるため、扱うソフトウェアが対応していない組み合わせもたくさんあります。よって、特殊な事情がない限り、有名なファイル形式を用いた方が良いと思います。

今回のプロジェクトのようなゲームを制作する場合は、画像は 32bitRGBA の PNG 形式、音声としては ogg vorbis と無圧縮 wav あたりを用いるのが普通かと思います（但し今回は Monogame を使うため、独自形式を用います)。