中級 C#言語 ++

MASA

2011 年 8 月

1 続 はじめに

本講座は中級 C#言語 (ライティング) の続編です。この講座では XNA を使ったシューティングゲーム制作を通じて、より本格的なオブジェクト指向プログラミングの習得を目指します。

1.1 前提

本講座は前講座『中級 C#言語 (ライティング)』の内容をある程度習得していることを前提に進めます。想定する開発環境は VisualStuio 2010 + XNA 4.0、もしくは VisualStuio 2008 + XNA 3.1 です[*1]。

1.2 製作目標

タイトル画面付きの、ボス戦 1 つだけの弾幕シューティングゲーム[*2]を作ります。

1.3 注意書き

本講座でのプログラムの設計や書き方はあくまでひとつの例であり、絶対的なものではありません。できるだけ分かりやすい設計となるようにしていますが、自分の好みに合う設計を自分で考えてみてください。

2 製作開始〜絵を動かすまで

早速製作を始めます。まずはおさらいということで、自機[*3]のクラスを作成し、キー入力で操作できるようにするところまで作ります。

2.1 ファイルの作成

まずはプロジェクトを作成します。以下の説明はバージョンや環境によって若干違いがありますが、適宜読み替えてください。VisualStuio を起動し、メニューのファイル→新しいプロジェクト→ Windows ゲームを選択し、名前にプロジェクト名 (ここでは TextGame2 とします) を入力して OK をクリックします。

[*1] XNA 3.1 と 4.0 の間で仕様の違いは比較的大きいですが、本講座では問題になることはありません。
[*2] あずま風
[*3] 主人公の事

続いて、自機クラスを書くためのファイルを作成します。ソリューションエクスプローラーを開き、TextGame2 を右クリックして 追加→新しい項目→クラスを選択し、クラス名 (ここでは Ship とします) を入力します。

さらに、自機のグラフィックをゲームで使用するデータとして追加します。32*32 ピクセル程度の自機の絵を用意して、(プロジェクトのフォルダ)[*4]\TextGame2Content\texture\ に保存してください。ここではファイル名を ship.png とします。なお、texture フォルダは標準では作られていないので作成してください。画像の形式は大抵のものが使えますが、個人的には透過が使える PNG 形式がおすすめです。そして、ソリューションエクスプローラーの TextGame2Content(Content) にその texture フォルダをドラッグアンドドロップして追加、もしくは TextGame2Content(Content) を右クリックして 追加→新しいフォルダで texture というフォルダを作成し、そのフォルダを右クリックして 追加→既存の項目で先ほどの ship.png を追加してください。

XNA 4.0 の場合には最後にもう一つ。ソリューションエクスプローラーの TextGame2 を右クリック→プロパティ→ XNA Game Studio タブの XNA Framework プロファイルを HiDef から Reach に変更してください。これをしないと、グラフィック性能の低い一部の PC では動かなくなります。[*5]

2.2 自機クラスの作成

準備が終わったところで、続いて自機クラスのコードを書いていきます。

ソースコード 1　Ship.cs

```
1  using System;
2  using System.Collections.Generic;
3  using Microsoft.Xna.Framework;
4  using Microsoft.Xna.Framework.Graphics;
5  using Microsoft.Xna.Framework.Input;
6  namespace TextGame2
7  {
8          public class Ship
9          {
10                 Texture2D Texture;
11                 Vector2 Position;
12                 public Ship()
13                 {
14                         Position = new Vector2(400, 300);
15                         Texture = Global.Game.Content.Load<Texture2D>("texture\\ship");
16                 }
17                 public void Update()
18                 {
19                         const float SPEED = 4.5f;
20                         KeyboardState ks = Keyboard.GetState();
21                         if (ks.IsKeyDown(Keys.Up))
22                         {
23                                 Position.Y -= SPEED;
```

[*4] 標準では、マイドキュメントの Visual Studio 2010\Projects\(プロジェクト名) です
[*5] その代わり幾つかの機能が使えなくなりますが、高度なことに手を出さない限りは問題ありません。

```
24                          }
25                          if (ks.IsKeyDown(Keys.Down))
26                          {
27                                  Position.Y += SPEED;
28                          }
29                          if (ks.IsKeyDown(Keys.Left))
30                          {
31                                  Position.X -= SPEED;
32                          }
33                          if (ks.IsKeyDown(Keys.Right))
34                          {
35                                  Position.X += SPEED;
36                          }
37                  }
38                  public void Draw()
39                  {
40                          Global.Sprite.Draw(Texture, Position, Color.White);
41                  }
42          }
43  }
```

ソースコード 2　Game1.cs

```
1   using System;
2   using System.Collections.Generic;
3   using Microsoft.Xna.Framework;
4   using Microsoft.Xna.Framework.Audio;
5   using Microsoft.Xna.Framework.Content;
6   using Microsoft.Xna.Framework.Graphics;
7   using Microsoft.Xna.Framework.Input;
8   namespace TextGame2
9   {
10          public static class Global//
11          {//
12                  public static SpriteBatch Sprite;//
13                  public static Game1 Game;//
14          }//
15
16          public class Game1 : Microsoft.Xna.Framework.Game
17          {
18                  GraphicsDeviceManager graphics;
19                  SpriteBatch spriteBatch;
20                  Ship Ship;//
21                  public Game1()
22                  {
23                          graphics = new GraphicsDeviceManager(this);
24                          Content.RootDirectory = "Content";
25                          graphics.PreferredBackBufferWidth = 640;//
26                          graphics.PreferredBackBufferHeight = 480;//
```

```
27                      Global.Game = this;//
28              }
29              protected override void Initialize()
30              {
31                      base.Initialize();
32                      Ship = new Ship();//
33              }
34              protected override void LoadContent()
35              {
36                      spriteBatch = new SpriteBatch(GraphicsDevice);
37                      Global.Sprite = spriteBatch;//
38              }
39              protected override void Update(GameTime gameTime)
40              {
41                      Ship.Update();//
42                      base.Update(gameTime);
43              }
44              protected override void Draw(GameTime gameTime)
45              {
46                      GraphicsDevice.Clear(Color.CornflowerBlue);
47                      Global.Sprite.Begin();//
48                      Ship.Draw();//
49                      Global.Sprite.End();//
50                      base.Draw(gameTime);
51              }
52      }
53  }
```

以上です。Game1.cs のほうは、プロジェクト作成時に自動生成されたコードからコメント等不要な行を取り除き、行末に//と書いてある行を書き加えたものです。このコードでの//は説明上の目印なので、実際にプログラムを書くときには不要です。

それでは Ship.cs から解説をしていきます。

1–5 行目の using は参照する名前空間を指定しています。Microsoft.Xna.Framework 名前空間にはベクトルを表す Vector2 や色を表す Color などが入っています。Microsoft.Xna.Framework.Graphics には画像データを表す Texture2D や画像描画に使う SpriteBatch などが入っています。これら 1–4 行目に書かれているものは使うことが多いので、ゲームで使うすべてのソースコードに書いておいていいと思います。Microsoft.Xna.Framework.Input には Keyboard や KeyboardState などの入力を処理するのに使うクラスが入っているので、入力を必要とするクラスのソースコードに書きましょう。using の指定は名前から中身が大体推測できるようになっているので、勘で適当に書いておけばたいていは問題ありませんし、使いたいクラスが IntelliSense で表示されない場合は適当に調べて書けばいいでしょう。

続いて 8 行目以降で Ship クラスを記述しています。10 行目で画像を保持するために Texture2D 型の Texture フィールド、11 行目で位置を保持するために Vector2 型の Position フィールドを宣言しています。12–16 行目の**コンストラクタ**で Position と Texture を初期化しています。Position は適当に画面の中央にしています。

2.2.1 コンテントパイプライン

さて、ここで 15 行目について詳しく説明します。この行では先程追加した画像データを読み込んでそれを Texture へ代入しています。XNA では画像や音などゲームで使うデータを読み込むのに、直接画像ファイルなどを読み込むのではなく**コンテントパイプライン**というものを使います。TextGame2Content というプロジェクトがそのコンテントパイプラインを担当している部分で、これに ship.png を追加したことでコンテントパイプラインにこの画像が追加されたのです。コンテントパイプラインに追加されたデータはアセットと呼ばれ、データの種類 (画像、音、といった区分) ごとに処理されてゲーム用のデータとして保存されます。この処理の際に画像データ間の拡張子の違いは吸収されるため、アセットの名前には拡張子が付きません。

ゲームからアセットを読み込むためには Game クラスの Content プロパティの Load 関数を使います。Load 関数はジェネリクス[*6]というものを使っているため、Load と書いた後に¡ ¿のカッコを書いて、その間に必要な型名を書いてから引数の () を書きます。現時点では Texture2D 型のものしか使わないので、Load¡Texture2D¿(引数) という書き方として覚えておけば十分です。Load 関数の引数はアセットの場所です。ファイルの場所を表すパスの区切り文字には \ を使いますが、C#の文字列中で \ を書くためには \\ と 2 つ続けて書く必要があります[*7]。先程も述べたように、アセットの名前には拡張子がつかないので注意が必要です。

Update メソッドの 20 行目ではキーボードの入力情報を取得し、KeyBoardState 型の ks ローカル変数に保存しています。同様にマウスの状態やゲームパッド[*8]の状態を取得することもできますが、詳しくは自分で調べてください。21–36 行目で、キーボードの↑↓←→キーが押されているかを確認し、Position の Y,X の値を変更して自機を動かしています。

ここで**定数**というものを使っています。19 行目がそれで、普通の変数宣言の前に const とつけることでその値が変更不能な定数として扱われます。定数は変数などとの区別をつけるために、一般的に全て大文字で書くとされていて、また単語の区切りには SHIP_SPEED のようにアンダーバーを使います。各方向の移動量を直接 4.5f と書くのではなく定数を使って書くことで以下のようなメリットがあります。

- あとから値を変更するときに定数の値を一箇所変更するだけで済む
- 4.5f という数値が直接コード上に出るよりも意味が分かりやすい
- ただの変数よりも若干実行速度が速い

Draw メソッド内の 40 行目では画像データを Position が示す座標に描画しています。Global.Sprite というのは Game1.cs の方で定義した SpriteBatch への参照です。このように、画像データを描画するには SpriteBatch クラスの Draw メソッドを使います。三番目の引数の Color.White の説明は面倒なので割愛します。適当に他の Color に書き換えれば大体わかるかと。

それでは次に Game1.cs のコードの解説です。

まず 10–14 行目で静的クラス[*9] Global を定義しています。これは C 言語などでいうグローバル変数[*10]を

[*6] C++ でいうテンプレートのようなもの。返り値の型を決めるために使っています。

[*7] TEX と同じような理由です

[*8] ただし Xbox360 のコントローラーを公式ドライバで接続している場合に限る。他のゲームパッドの状態を取得する場合には XNA 以外のライブラリを使う必要があります。

[*9] インスタンス化できないクラスのこと。インスタンスによらない関数や変数などをまとめて持つために用いられる。System.Math なども静的クラス。

[*10] プログラム内のどこからでも参照できる変数のこと。デメリットもあるため C#では使えないようになっている。

使うためのトリックで、このクラスの静的変数としてよく使う Game1 への参照や SpriteBatch への参照を保持しています。

16 行目以下は自動生成された Game1 クラスを編集したものです。自動生成された部分はまだ理解しなくても別に問題ありません。また 16 行目のクラスの宣言方法については後で詳しく解説します。

20 行目で Ship 型のフィールド Ship を宣言し、32 行目で Ship のインスタンスを作成してこれに代入しています。復習となりますが、オブジェクトの変数は宣言しただけではオブジェクトは実際に作られず、インスタンス化したものを代入して初めて使えるようになるので気をつけてください[*11]。

25,26 行目ではウィンドウの大きさを設定しています。デフォルトでは横長の 800*480 のサイズですが、これを一般的な 640*480 に変更しています。27 行目では Global の Game へ自分自身への参照を代入しています。this とは自分自身への参照を指すキーワードです。

37 行目では Global の Sprite に SpriteBatch への参照を代入しています。41 行目では Ship の Update メソッドを呼び出して、自機の状態を更新しています。Game の Update メソッドは標準では 1 秒間に 60 回実行されます。

44–51 行目の Draw メソッドの中身について説明します。Draw メソッドも Update メソッドと同様に 1 秒に 60 回、Update メソッドを実行した後に呼び出されてゲーム画面を描画します。46 行目の Clear メソッドはゲーム画面を引数に指定された色で塗りつぶしています。これを行わないと前回の Draw で描画されたゲーム画面がそのまま残って、その上に新しく描画してしまうため残像が残っているようになります。

48 行目の Ship.Draw では Ship の Draw メソッドを呼び出していますが、その前後を Global.Sprite.Begin と Global.Sprite.End メソッドで挟んでいます。実は SpriteBatch の Draw メソッドを呼び出す前には Begin、Draw が完了した後には End をそれぞれ実行する必要があります。Ship の Draw メソッド内部では SpriteBatch の Draw メソッドを使っているため、Begin と End で挟む必要があるのです。なお、複数回 SpriteBatch の Draw メソッドを呼び出す場合はそれら全ての呼び出し一つ一つを挟む必要はなく、最初に Draw を呼び出す前に一度 Begin を、最後に Draw を呼び出した後に一度 End を呼び出すだけで OK です[*12]。

これでコードの解説は終了です。ミスがなければ実行すると自機のグラフィックが表示され、キーボードの↑↓←→で移動するはずです。

3 キャラクターを増やす with 継承

自機の表示ができたところで、続いて敵、敵弾 (以下では弾、Bullet と呼びます)、自機のショット (以下ではショット、Shot と呼びます) を作成しましょう。ここで本講座のメインテーマである**クラスの継承**が登場します。

3.1 概念的なお話

敵などを実装する前に、それぞれに必要となるであろう要素を考えてみます。位置、画像、それから状態を更新するメソッド、描画するメソッド... これらは自機、敵、弾、ショット全てに共通して必要な要素です。これらの要素を各クラス全てについて個別に定義するのは面倒です。

[*11] これを忘れると NullReferenceException、いわゆる「ぬるぽ」が発生します。
[*12] Draw 呼び出し一つ一つを Begin–End で挟むと大量に Draw したときに実行速度が著しく低下します。

そこで登場するのが**継承**という概念です。位置、画像、Update メソッド、Draw メソッドを持つクラスを作成して、そのクラスをベースにより具体的な自機や敵などのクラスに派生させるというのが継承です。ベースとなるクラスの名前を Character、派生した先のクラスを Ship,Enemy,Bullet,Shot などとすると、Character を**基底クラスやベースクラス**、Ship や Enemy などを**派生クラス**といい、**Ship は Character を継承する**などと言います。

3.2 初めての継承

一気に敵や弾、ショットを実装するのは大変なので、まずは継承を使った形に自機を書きなおします。

ソースコード 3　Character.cs

```
1  using System;
2  using System.Collections.Generic;
3  using Microsoft.Xna.Framework;
4  using Microsoft.Xna.Framework.Graphics;
5  namespace TextGame2
6  {
7          public class Character
8          {
9                  private Texture2D Texture;
10                 protected Vector2 Position;
11                 public Character()
12                 {
13                 }
14                 protected void SetTexture(string name)
15                 {
16                         Texture = Global.Game.Content.Load<Texture2D>(name);
17                 }
18                 public virtual void Update()
19                 {
20                 }
21                 public virtual void Draw()
22                 {
23                         Global.Sprite.Draw(Texture, Position, Color.White);
24                 }
25         }
26 }
```

ソースコード 4　Ship.cs

```
1  using System;
2  using System.Collections.Generic;
3  using Microsoft.Xna.Framework;
4  using Microsoft.Xna.Framework.Input;
5  namespace TextGame2
6  {
7          public class Ship : Character
8          {
```

```
9                   public Ship()
10                          : base()
11                  {
12                          Position = new Vector2(320, 240);
13                          SetTexture("texture\\ship");
14                  }
15                  public override void Update()
16                  {
17                          const float SPEED = 4.5f;
18                          KeyboardState ks = Keyboard.GetState();
19                          if (ks.IsKeyDown(Keys.Up))
20                          {
21                                  Position.Y -= SPEED;
22                          }
23                          if (ks.IsKeyDown(Keys.Down))
24                          {
25                                  Position.Y += SPEED;
26                          }
27                          if (ks.IsKeyDown(Keys.Left))
28                          {
29                                  Position.X -= SPEED;
30                          }
31                          if (ks.IsKeyDown(Keys.Right))
32                          {
33                                  Position.X += SPEED;
34                          }
35                          base.Update();
36                  }
37          }
38  }
```

Game1.cs は変更不要です。

3.2.1 アクセス指定子とカプセル化

まずは Character.cs で基底クラスとなる Character を定義しています。9 行目で Texture フィールド、10 行目で Position フィールドを宣言していますが、その前に protected と書いてあるのを見てください。protected や 18 行目に書いてある public、またここには書いていませんが private というキーワードを**アクセス指定子**[*13]といいます。クラスのメンバ変数やメンバ関数の前にアクセス指定子をつけることで、それらの公開範囲を指定することが出来ます[*14]。このように、private を付ける、もしくは何も付けない場合は[*15]そのクラス内からのみアクセス可能、protected を付けた場合は派生クラス内からもアクセス可能、public を付けた場合はどこからでもアクセス可能になります。「それなら何も考えずに public と書いておけばいいじゃないか」と思うでしょうが、ここにはオブジェクト指向の根幹をなす考え方の一つである**カプセル化**という考えが関わってきます。

[*13] 他にもう一つ internal というものもありますが、ここでは扱いません。
[*14] クラスの前も public, internal, private のアクセス指定子を付けられますが、普通は public で構いません。
[*15] アクセス指定子を付けない場合は private として扱われます

表 1　アクセス指定子とアクセス可能範囲

	private	protected	public
そのクラス内	○	○	○
派生クラス内	×	○	○
一般のクラス内	×	×	○

カプセル化とは、オブジェクトの中の動作をできるだけ隠蔽して独立性を高めることで、予期せぬバグの発生を防いだり保守性を高めたりしようという考えです。オブジェクトの中身をすべて公開すると外部から間違って操作されやすかったり、設計を変更したときに様々なところまで変更が及ぶことになったりするので、外部との接点を減らすのです。公開する必要がないところは隠蔽するとともに、できるだけ公開するものが少なく独立性が高いクラスを設計することも重要です。また、公開されているメンバが少ないと IntelliSense の候補が少なくなって見やすいという利点もあります。

さて、この場合で考えると Texture は描画するときに必要となりますが、描画メソッドは Character クラス自体が持っていてそれで完結しているので外部のクラスや派生クラスから参照する必要はありません。よって private と書いています。Position は今のところ、外部から参照する必要は見当たりませんが、Ship を動かすときに使っているため派生クラスから参照可能な protected を指定しています。

メソッドについても考えると、コンストラクタ、Update、Draw はいずれも外部のクラスから扱われる必要があります。そのため public を指定しました。

14 行目の SetTexture メソッドを見てください。最初に書いた Ship ではこの関数はありませんでしたが、Enemy や Bullet、Shot でもアセット名を指定してテクスチャを設定する処理は必要になるであろうことからその処理をひとつの関数としました。このメソッドは派生クラスから使うことを前提としているので protected を指定しました。

続いて Update メソッドです。各種キャラクターで共通して行うべき更新処理は特にないので Update メソッドの中身は空です。ただ、派生クラスは全て Update メソッドを持つので書いておきます。18 行目にある virtual についてはあとで触れます。

Draw メソッドは最初に書いた Ship のものと同一です。

続いて Ship.cs です。このファイルでは Character を継承した Ship クラスを定義しています。継承したクラスを書くには 7 行目のように、クラス名の後に：で区切って基底クラスの名前を書きます。

続いてコンストラクタです。継承をするとコンストラクタの書き方がすこし変わり、9–10 行目のようにコンストラクタの名前を書いた後に：で区切って base(引数) と書く必要があります。こう書くことで、基底クラスのコンストラクタが base() の括弧内で指定した引数で実行され、そのあと派生クラスのコンストラクタが実行されます。

Update メソッドの説明に入る前にすこし脱線して、先ほどの virtual と 15 行目の override、それから継承の持つもう一つの性質について解説します。

3.2.2　オーバーライド

実は継承を使うことにはもうひとつのメリットがあります。

ソースコード 5　オーバーライドの例

```
1 Ship s = new Ship();
```

```
2 s.Update();
3 s.Draw();
4 Character c = new Ship();
5 c.Update();
6 c.Draw();
```

1,2 行目が動くのは明らかです。Ship 型の変数 s に Ship 型のインスタンスが代入されて、Ship クラスの Update メソッドを呼びだしています。では 3 行目はどうでしょう。これも動きます。3 行目を実行すると、Ship の基底クラスである Character の Draw メソッドが呼び出されます。このように、Ship では定義されていないメソッドでも基底クラスにあるメソッドであれば呼び出すことが可能です。そのため Ship.cs の 13 行目でも Character クラスの SetTexture メソッドを Ship から呼び出せています。

それでは 4 行目はどうでしょう。Character 型の変数に Ship 型のインスタンスを代入しています。これも実は動きます。派生クラスのインスタンスは基底クラスの型へ変換が可能なのです。ただし、逆に基底クラスのインスタンスを派生クラスの型の変数へ代入することは出来ません (Ship sh = new Character(); というのは不可能ということ)。この行の結果として、Character 型のインスタンスへの参照として変数 c が定義されます。

5 行目は置いておいて、6 行目を考えます。6 行目で Character クラスの Draw メソッドが呼び出されるというのはいいでしょう。

では 5 行目が問題です。Character クラスの Update が呼び出されるか、それとも Ship クラスの Update が呼び出されるか。答えは、Ship クラスの Update です。c は Character 型のインスタンスなのに何故 Ship クラスの Update になるのかというと、Update メソッドが[*16]**名前が似ているものにオーバーロード overload があります。これは同じ名前のメソッドでも引数が異なれば複数定義可能という機能です。**されているからです。詳しい説明は難しくなるので省略しますが、メソッドをオーバーライドすることで基底クラスのインスタンスとして扱われていても派生クラスのメソッドが呼び出されるのです。メソッドをオーバーライドするには、オーバーライドされる基底クラスのメソッドの前に virtual と書いて仮想関数[*17]に指定します。そしてオーバーライドする派生クラスのメソッドの前に override と書きます。

派生クラスで基底クラスと同一名称のメソッドを定義するのにはもうひとつ new というキーワード[*18]を使って書くやりかたがあります。こちらの方法だと 5 行目のように書くと Character の Update が呼び出されますが、あまり使わないので詳しくは説明しません。

それでは本編に戻ります。Update メソッド内 35 行目の base.Update では基底クラス Character の Update を呼びだしています。このようにして呼び出した場合はオーバーライドしていても必ず基底クラスのメソッドが呼び出されます。また、base という参照を使うことで基底クラスの持つ private でないメンバにアクセスすることが可能です。なお、現時点では Character の Update メソッドでは何も行っていないため base.Update をする必要はありませんが、あとで Character の Update に何か追加した時のことを考えて呼びだしています。

Ship の Draw メソッドは Character の Draw メソッドそのままで構わないため、記述していません。派生クラスで特別な処理が必要になったときにオーバーライドして記述すればいいのです。

[*16] オーバーライド
[*17] 説明はしない
[*18] インスタンスを生成するときに使う new とは全く関係ありません

これで実行すると、最初に書いたのと全く同様に動作するはずです。Ship クラスひとつだけだと継承のありがたみは全くありませんが、ここからクラスが増えてくると威力を発揮します。

3.3 敵、弾、ショットの実装

それでは他のクラスを実装していきます。まずはコンテントパイプラインに自機と同様に敵、弾、ショットの絵を追加してください。サイズはそれぞれ 64*64,16*16,8*8 としました。続いて各クラス用のソースファイルを作成します。ここでは Enemy.cs, Bullet.cs, Shot.cs としました。そして以下のようなコードを書きます。

ソースコード 6　Enemy.cs

```
1  using System;
2  using System.Collections.Generic;
3  using Microsoft.Xna.Framework;
4  namespace TextGame2
5  {
6          public class Enemy : Character
7          {
8                  int Count;// 何回Update が呼ばれたかを数える変数
9                  public Enemy() : base()
10                 {
11                         SetTexture("texture\\enemy");
12                         Position = new Vector2(320, 100);
13                 }
14                 public override void Update()
15                 {
16                         Count++;
17                         if (Count % 30 == 0)// 30フレームごとに発射
18                         {
19                                 Fire(3.5f, MathHelper.PiOver2);
20                         }
21                         base.Update();
22                 }
23                 void Fire(float speed, float angle)// 速さと角度を指定して弾を撃つ
24                 {
25                         Global.Game.Bullets.Set(Position, new Vector2((float)Math.Cos(angle) * speed
                               , (float)Math.Sin(angle) * speed));
26                 }
27         }
28 }
```

ソースコード 7　Shot.cs

```
1  using System;
2  using System.Collections.Generic;
3  using Microsoft.Xna.Framework;
4
5  namespace TextGame2
```

```
6  {
7          public class ShotManager
8          {
9                  Shot[] Shots;
10                 public ShotManager()
11                 {
12                         Shots = new Shot[64];
13                         for (int i = 0; i < Shots.Length; i++)
14                         {
15                                 Shots[i] = new Shot();
16                         }
17                 }
18                 public void Update()
19                 {
20                         for (int i = 0; i < Shots.Length; i++)
21                         {
22                                 if(Shots[i].Flag)// 有効なやつを探して更新
23                                 {
24                                         Shots[i].Update();
25                                 }
26                         }
27                 }
28                 public void Draw()
29                 {
30                         for (int i = 0; i < Shots.Length; i++)
31                         {
32                                 if(Shots[i].Flag)
33                                 {
34                                         Shots[i].Draw();
35                                 }
36                         }
37                 }
38                 public void Make(Vector2 pos, Vector2 vel)
39                 {
40                         for (int i = 0; i < Shots.Length; i++)
41                         {
42                                 if(!Shots[i].Flag)// 無効になっているのを探す
43                                 {
44                                         Shots[i].Set(pos, vel);
45                                         return;
46                                 }
47                         }
48                 }
49         }
50         public class Shot : Character
51         {
52                 Vector2 Velocity;// 速度
53                 public bool Flag;// 現在有効かを示す
54                 public Shot() : base()
```

```
55                  {
56                          Flag = false;
57                          SetTexture("texture\\shot");
58                  }
59                  public void Set(Vector2 pos, Vector2 vel)// 有効にする
60                  {
61                          Position = pos;
62                          Velocity = vel;
63                          Flag = true;
64                  }
65                  public override void Update()
66                  {
67                          Position += Velocity;// 速度を位置に加えて進める
68                          base.Update();
69                  }
70          }
71  }
```

ソースコード 8　Bullet.cs

```
1   using System;
2   using System.Collections.Generic;
3   using Microsoft.Xna.Framework;
4   namespace TextGame2
5   {
6           public class BulletManager
7           {
8                   Bullet[] Bullets;
9                   public BulletManager()
10                  {
11                          Bullets = new Bullet[256];
12                          for (int i = 0; i < Bullets.Length; i++)
13                          {
14                                  Bullets[i] = new Bullet();
15                          }
16                  }
17                  public void Update()
18                  {
19                          for (int i = 0; i < Bullets.Length; i++)
20                          {
21                                  if (Bullets[i].Flag)
22                                  {
23                                          Bullets[i].Update();
24                                  }
25                          }
26                  }
27                  public void Draw()
28                  {
29                          for (int i = 0; i < Bullets.Length; i++)
```

```
30                          {
31                                  if (Bullets[i].Flag)
32                                  {
33                                          Bullets[i].Draw();
34                                  }
35                          }
36                  }
37                  public void Set(Vector2 pos, Vector2 vel)
38                  {
39                          for (int i = 0; i < Bullets.Length; i++)
40                          {
41                                  if (!Bullets[i].Flag)
42                                  {
43                                          Bullets[i].Set(pos, vel);
44                                          return;
45                                  }
46                          }
47                  }
48          }
49          public class Bullet : Character
50          {
51                  Vector2 Velocity;
52                  public bool Flag;
53                  public Bullet() : base()
54                  {
55                          SetTexture("texture\\bullet");
56                  }
57                  public void Set(Vector2 pos, Vector2 vel)
58                  {
59                          Position = pos;
60                          Velocity = vel;
61                          Flag = true;
62                  }
63                  public override void Update()
64                  {
65                          Position += Velocity;
66                          base.Update();
67                  }
68          }
69  }
```

ソースコード 9　Game1.cs

```
1  (using 略)
2  namespace TextGame2
3  {
4          public static class Global{略 }
5          public class Game1 : Microsoft.Xna.Framework.Game
6          {
```

```
7                   GraphicsDeviceManager graphics;
8                   SpriteBatch spriteBatch;
9                   public Ship Ship;
10                  public BulletManager Bullets;
11                  public ShotManager Shots;
12                  public Enemy Enemy;
13                  public Game1(){略 }
14                  protected override void Initialize()
15                  {
16                          base.Initialize();
17                          Ship = new Ship();
18                          Enemy = new Enemy();
19                          Bullets = new BulletManager();
20                          Shots = new ShotManager();
21                  }
22                  protected override void LoadContent(){略 }
23                  protected override void Update(GameTime gameTime)
24                  {
25                          Ship.Update();
26                          Enemy.Update();
27                          Bullets.Update();
28                          Shots.Update();
29                          base.Update(gameTime);
30                  }
31                  protected override void Draw(GameTime gameTime)
32                  {
33                          GraphicsDevice.Clear(Color.CornflowerBlue);
34                          Global.Sprite.Begin();
35                          Shots.Draw();
36                          Enemy.Draw();
37                          Bullets.Draw();
38                          Ship.Draw();
39                          Global.Sprite.End();
40                          base.Draw(gameTime);
41                  }
42          }
43  }
```

量が多いですが、あまり難しいことはやっていないので説明はちょくちょく端折ります。ソースコードに書かれているコメントで説明を済ませているところもあります。

これを実行すると自機と敵が表示され、0.5 秒間隔で敵が真下に弾を撃ってきます。ショットを撃つ機能はまだつけていません。

細かい説明の前に全体的な設計について説明します。まず、Enemy, Shot, Bullet が Character を継承しているというのはいいでしょう。その上で、各クラスがそれぞれ必要とする機能を個別に実装しています。

それからひとつ重要なのが複数存在するキャラクターの管理です。Ship や Enemy はこのプログラムの場合 1 つずつしか存在しませんが、Shot や Bullet は大量に存在するため配列を使って管理しています。ゲーム全体を管理する Game1 クラスから for ループで配列一つ一つに対して処理を行うのはやや面倒であり、また

生成する処理などでも若干込み入ったことを行っているので、弾の配列を管理する BulletManager クラスとショットの配列を管理する ShotManager クラスを作成し、Game1 クラスから Ship や Enemy を扱うのと同様にひとつのインスタンスに対して Update や Draw を呼び出すだけで処理できるようにしました。

また、複数キャラクターを管理する手法として**オブジェクトプール**という手法を採用しました。オブジェクトプールとは、大きめな配列を用意して使用するインスタンスをあらかじめ全て生成しておき、インスタンスが必要になった時点で使われていないインスタンスを有効化して使用する、という手法です。必要になったときにインスタンスを生成するという手法に比べて、やや美しくない、用意しておいたインスタンスを全て使いきってしまうとそれ以上出せなくなる[*19][*20]、という短所がありますが、動作速度が安定している、構造が比較的簡単、という長所もあります。

それでは、順番が前後することになりますがショットを例にオブジェクトプールの実装を見てみます。Shot クラスでオブジェクトプールに関係するところは 2 点、53 行目の Flag フィールドと 59–64 行目の Set メソッドです。Flag はそのインスタンスが現在有効かどうかを示すフィールドで、Set メソッドは初期位置と速度を指定してそのインスタンスを有効化するメソッドです。

ShotManager クラスは Shot の配列を管理するクラスです。まずコンストラクタの 12 行目で配列のインスタンスを作成し、さらに 13–16 行目で配列の各要素へ Shot のインスタンスを作成して代入しています。復習ですが、配列を使うときはまず配列変数を宣言し、それから配列変数に配列のインスタンスを代入して、さらに配列の各要素を初期化するという 3 ステップが必要です。配列のインスタンスを作成しただけでは各要素の値は設定されていないので注意してください[*21]。なお、ここではショットの配列のサイズを 64 としましたが、画面上にどれぐらいショットが出るかというゲームデザインに合わせて適当に設定してください。少なすぎるとキャラオーバーが起きますし、あまり多すぎてもメモリの無駄遣いとなって動作速度の低下などを招きます。

Update メソッド、Draw メソッドでは Flag が true になっている有効な Shot のインスタンスに対してそれぞれ Update と Draw を実行しています[*22]。Flag が false になっている無効なインスタンスは存在しないものとして扱っています。

次の Make メソッドは Shot のインスタンスを有効化するメソッドです。Flag が false で無効になっている使われていないインスタンスを探し、その無効なインスタンスを有効化して処理を終了しています。

Bullet クラスと BulletManager クラスに関しても行っていることは全く同じです[*23][*24]。弾のほうがショットよりも画面に出る数が多くなるであろうことが予測されるので配列の大きさを 256 にしてあるだけです。

Enemy クラスについてはオブジェクトプールを使っていないのでもっとシンプルです[*25]。敵は弾を撃つので、弾を撃つための Fire メソッドを作っておきました。

[*19] ゲームの世界ではこれをキャラオーバーと呼びます。

[*20] 使いきってしまったときに配列を拡張して追加のインスタンスを作って回避する、という手もありますが、瞬間的に大きな負荷がかかるため常に使える手法ではありません

[*21] ぬるぽ

[*22] 「Update と Draw がほとんど同じ処理を行っているのが気持ち悪い!もっとまとめたい!」という人がもしいたら、デリゲートという仕組みを上手く使うと幸せになれるかもしれません。ただし処理速度は若干犠牲になることが予想されます

[*23] 「Shot と Bullet が全く同じ処理を行っているのが気持ち悪い!もっとまとめたい!」という人がいるかもしれませんが、現時点では確かに同じであるものの後々処理内容が異なってくるので分けてあります。また、Shot と Bullet に継承関係を持たせたいと思うかもしれませんが、継承とは基本的に「抽象的なものを基底クラスとしてより具体的なものを派生クラスにする」という考え方なので Shot と Bullet の間の関係ではあまりふさわしくありません。

[*24] 「ShotManager と BulletManager がほとんど同じ処理を行っているのが気持ち悪い!もっとまとめたい!」という人がもしいたら、ジェネリッククラスという仕組みを上手く使うと幸せになれるかもしれません。

[*25] 敵が複数出るゲームにするのであれば Shot などと同様の機構にすればいいでしょう。

細かいことになりますが、XNA では浮動小数点型として float 型を使っていて、.NET のライブラリでは浮動小数点型として double 型を使っています[*26]。そのため、Math.Cos 関数[*27]などは返り値が double 型である一方、Vector2 クラス[*28]のコンストラクタの引数は float 型であるため、**型キャスト**が必要になります。

型キャストとはある型の値を別の型に変換する処理のことで、変換したい値の前に (float) のように () で囲んで型名を書きます。float から double への変換は型キャストを明記しなくても大丈夫ですが、double から float への変換はきちんと型キャストを書く必要があります[*29]。型キャストは float と double 間だけでなく、int などの整数型と float などの浮動小数点型の間[*30]、基底クラスと派生クラスの間[*31]などでも行えます。

それからもうみっつ細かいことを。Enemy.cs の 19 行目で Fire メソッドを使っていますが、その引数として 3.5f と書かれています。普通に 3.5 と書くとそれは double 型の値として扱われますが、数値の末尾に f とつけると float 型として扱われるようになります。Fire の 2 番目の引数で MathHelper.PiOver2 と書いています。MathHelper クラスは XNA Framework のライブラリに所属するクラスで、数学的な定数やちょっとした数学関数が定義されているクラスです。PiOver2 は $\frac{\pi}{2}$ を表しています。それから、スクリーン上の座標は y 軸が一般的な座標と上下反転しているので、$\frac{\pi}{2}$ は真下方向を表しているので注意して下さい。

Game1 クラスでは特に変わったことはしていません。逆に言うと、Game1 クラスで特に変わったことをしなくても済むような設計をすると美しいです。一つ触れるとすると、35–38 行目の描画コードです。呼び出した順に描画されていくので、ゲーム画面の見やすさを考えて描画の順番を決める必要があります。ここではショットや敵本体は他のものに隠れてもあまり問題がなく、弾や自機が他のものに隠れると不便なためこのような順番にしました。

4　遊べるようにする

ここまででプログラミングの技術的な話は大体終わりです。続いてこの作品をゲームとして遊べるように改造していきます。

4.1　位置を調整する

先ほどのコードの実行結果を見て気付いたかもしれませんが、実は Position で示している位置が画像の左上の点に相当しています。それだと敵が左上の点から弾を撃っているように見えて不自然なので、Position の位置を中央として画像が描画されるように改良します。また、これとは関係ありませんがずっと実行し続けていると弾がキャラオーバーを起こして弾が出なくなってしまいます。それを防ぐために、画面外に出た弾やショットを消去する仕組みを作ります。

ソースコード 10　Character.cs

```
1 (using 略)
2 namespace TextGame2
3 {
```

[*26] double 型は float 型よりも高精度。ゲーム用のデバイスの仕様などを考えてこのようになったものと思われます。
[*27] .NET 標準ライブラリにある System.Math クラスの静的関数。ラジアンの角度を引数としてそのコサインを返す。Sin も同様。
[*28] こちらは XNA Framework のライブラリ
[*29] 情報が失われるため
[*30] 浮動小数点から整数の変換では小数点以下切り捨てになります
[*31] Character 型の変数に Ship のインスタンスが入っているときにそれを Ship 型として扱うなど。Ship のインスタンスが入っているときにそれを Enemy 型などにキャストしようとするとエラーになります

```
4       public class Character
5       {
6               protected Texture2D Texture;
7               protected Vector2 Position;
8               protected Vector2 Size;
9               public Character() {略 }
10              protected void SetTexture(string name)
11              {
12                      Texture = Global.Game.Content.Load<Texture2D>(name);
13                      Size = new Vector2(Texture.Width / 2f, Texture.Height / 2f);
14              }
15              public virtual void Update(){略 }
16              public virtual void Draw()
17              {
18                      Global.Sprite.Draw(Texture, Position − Size, Color.White);
19              }
20              protected bool IsOverWindow()
21              {
22                      return (Position.X + Size.X < 0
23                       || Position.X − Size.X > Global.Game.WIDTH
24                       || Position.Y + Size.Y < 0
25                       || Position.Y − Size.Y > Global.Game.HEIGHT);
26              }
27      }
28  }
```

ソースコード 11　Shot.cs の Shot の Update および Bullet.cs の Bullet の Update

```
1   public override void Update()
2   {
3           Position += Velocity;
4           if (IsOverWindow())
5           {
6                   Flag = false;
7           }
8           base.Update();
9   }
```

ソースコード 12　Game1.cs の Game1

```
1   public class Game1 : Microsoft.Xna.Framework.Game
2   {
3           public readonly int WIDTH = 640;
4           public readonly int HEIGHT = 480;
5           (以下フィールド宣言略)
6           public Game1()
7           {
8                   graphics = new GraphicsDeviceManager(this);
9                   Content.RootDirectory = "Content";
10                  graphics.PreferredBackBufferWidth = WIDTH;
```

```
11                  graphics.PreferredBackBufferHeight = HEIGHT;
12                  Global.Game = this;
13          }
14          (以下略)
15  }
```

描画位置を中心にする処理は、単純に描画する位置を画像サイズの半分だけずらしているだけです。画像サイズの半分の大きさの Vector2 を作るのには、画像の幅と高さを表す Texture2D クラスの Width プロパティと Height プロパティを使いました。

画面外に出た弾やショットを消すのには、まず Character クラスで画面外に出ているかどうかを判定する IsOverWindow 関数を作り、その関数の結果画面外に出ていたら Flag を flase にして無効化しています。この際ゲーム画面のサイズを直接数値で書くよりも定数を使ったほうがいいため、Game1 クラスで画面の幅と高さを表す WIDTH 定数と HEIGHT 定数を作りました。この時、最初に定数を作るのに使った const キーワードではなく readonly キーワードを使っています。どちらも定数を作るのに使うという点では同じですが内部的な動作が異なるので、クラスの持つ定数としては readonly キーワードを使って定義しています[*32]。

4.2 自機の動きを改良する

続いて自機の動きの改造です。まず何よりもショットの発射、また斜め移動をしたときに移動の「速さ」が変わってしまう現象の修正と、自機が画面外まで移動しないようにします。

ソースコード 13　Ship.cs の Ship

```
1  public class Ship : Character
2  {
3          int Count;
4          public Ship() : base(){略 }
5          public override void Update()
6          {
7                  KeyboardState ks = Keyboard.GetState();
8                  Move(ks);
9                  Attack(ks);
10                 base.Update();
11         }
12         void Move(KeyboardState ks)
13         {
14                 float speed = 4.5f;
15                 int x = 0, y = 0;
16                 if (ks.IsKeyDown(Keys.Up)) y−−;
17                 if (ks.IsKeyDown(Keys.Down)) y++;
18                 if (ks.IsKeyDown(Keys.Left)) x−−;
19                 if (ks.IsKeyDown(Keys.Right)) x++;
20                 if (x != 0 && y != 0)
21                 {
22                         speed ∗= (float)Math.Sqrt(0.5);
```

[*32] const は C 言語で言う#define マクロを使った定数で、readonly は宣言と同時の代入、もしくはコンストラクタでの代入だけが認められている変数のようなもの。そのため const はインスタンスごとに異なる値を設定するなどのことができない。

```
23                  }
24                  Position.X += x * speed;
25                  Position.Y += y * speed;
26                  if (Position.X - Size.X < 0) Position.X = Size.X;
27                  if (Position.X + Size.X > Global.Game.WIDTH) Position.X = Global.Game.WIDTH -
                        Size.X;
28                  if (Position.Y - Size.Y < 0) Position.Y = Size.Y;
29                  if (Position.Y + Size.Y > Global.Game.HEIGHT) Position.Y = Global.Game.HEIGHT
                        - Size.Y;
30          }
31          void Attack(KeyboardState ks)
32          {
33                  if (ks.IsKeyDown(Keys.Z))
34                  {
35                          Count++;
36                          if (Count % 10 == 0)
37                          {
38                                  Fire(6.5f, -MathHelper.PiOver2);
39                          }
40                  }
41          }
42          void Fire(float speed, float angle)
43          {
44                  Global.Game.Shots.Make(Position,
45                   new Vector2((float)Math.Cos(angle) * speed,
46                   (float)Math.Sin(angle) * speed));
47          }
48  }
```

行数を削るため if 文でブロックを使わずに書いている所があります。

Update メソッド内に移動処理や攻撃処理を全て書いていると長くなって見通しが悪いため、Move メソッドと Attack メソッドに処理を分割しました。このように、プログラミングを進めていく過程で既に書いてあるソースコードを編集し整理や改良をすることをリファクタリングと言います。

それではじめに Attack メソッドの中身を見ます。引数として渡された KeyboardState で Z キーが押されていたらカウンタを進め、カウンタが 10 の倍数となるたびにショットを発射しています。この方式だとキーを押した瞬間にはショットがでないという事が起きますが、大した時間にはならないため無視しています。この欠点を改善した連射方式も考えられますが、説明が面倒なので自分で考えてください。Fire 関数は Enemy で書いたものとほとんど同一です。Z キーでショットの発射としているのは、キーボードの配置的に ZXC 左 Shift キー辺りが押しやすく、またゲームで用いられることが多いのでそれに倣いました。

Move メソッドは斜め移動をしたときに速さが $\sqrt{2}$ 倍になるのを防ぐため、キー入力で直接 Position を変更するのではなく一旦 x と y という変数に方向を保存し、x y がどちらも 0 でない、つまり縦方向と横方向両方に移動するときに速さに $\frac{1}{\sqrt{2}}$ を掛けています。

自機が画面外に出ないようにする処理は先程の IsOverWindow の処理と似た考え方です。もう少し手抜きをした書き方をしたくなりますが、この書き方が一番正確なようです。

4.3 当たり判定を付ける

今度はキャラクターとキャラクターが接触しているかの判定、いわゆる当たり判定の実装と、自機や敵のライフの実装を行います。当たり判定の実装にはさまざまな方法がありますが、簡単さを考えて今回は円と円の当たり判定を考えます。

円と円の当たり判定の理屈は簡単です。円の中心座標を x, y として、半径を r としたとき、円 A と円 B が接触しているかを表す式は、

$$\sqrt{(A.x - B.x)^2 + (A.y - B.y)^2} < A.r + B.r$$

です。円の中心間の距離が半径の和よりも短ければ接触している、というわけです。実際にプログラミングするときは平方根の計算が若干重いので、両辺を自乗した形で計算します。

以下のコードを書く前に、スプライトフォントという機能を使うためコンテントパイプラインをいじります。ソリューションエクスプローラーでコンテントプロジェクトの TextGame2Content(Content) を右クリック→追加→新しい項目→スプライトフォントを選択し、ファイル名をここでは font.spritefont として作成します。作成するとこのファイルが開かれるので、¡Size¿16¡/Size¿となっているところの数値を適当に変更します。ここでは 64 としました。

ソースコード 14　Character.cs の Character

```
1  public class Character
2  {
3          protected Texture2D Texture;
4          protected Vector2 Position;
5          protected Vector2 Size;
6          protected float Radius;
7          (中略)
8          protected bool GetHit(Character target)
9          {
10                 return (Position − target.Position).LengthSquared() < (Radius + target.Radius) ∗ (
                       Radius + target.Radius);
11         }
12 }
```

ソースコード 15　Ship.cs の Ship

```
1  public class Ship : Character
2  {
3          int Count;
4          int Life;
5          public Ship()
6                  : base()
7          {
8                  Position = new Vector2(320, 240);
9                  SetTexture("texture\\ship");
10                 Radius = 6;
11                 Life = 3;
```

```
12          }
13          public void Damage()
14          {
15                  Life−−;
16                  if (Life == 0)
17                  {
18                          Global.Game.GameOver();
19                  }
20                  Global.Game.Bullets.Clear();// 被弾したら全弾クリア
21          }
22          (以下略)
23  }
```

ソースコード 16　Enemy.cs の Enemy

```
1
2   public class Enemy : Character
3   {
4           int Count;
5           int Life;
6           Color Color;
7           public Enemy() : base()
8           {
9                   SetTexture("texture\\enemy");
10                  Position = new Vector2(320, 100);
11                  Radius = 32;
12                  Life = 200;
13          }
14          public override void Update()
15          {
16                  Color = Color.White;// 普段は白
17                  Count++;
18                  if (Count % 30 == 0)
19                  {
20                          Fire(3.5f, MathHelper.PiOver2);
21                  }
22                  base.Update();
23          }
24          void Fire(float speed, float angle){略 }
25          public override void Draw()
26          {
27                  Global.Sprite.Draw(Texture, Position − Size, Color);
28          }
29          public void Damage()
30          {
31                  Life−−;
32                  Color = Color.Red;// ダメージを受けたら赤くフラッシュ
33                  if (Life == 0)
34                  {
```

```
35                          Global.Game.GameClear();
36                  }
37          }
38  }
```

ソースコード 17　Shot.cs の Shot

```
1   public class Shot : Character
2   {
3           Vector2 Velocity;
4           public bool Flag;
5           public Shot() : base()
6           {
7                   Flag = false;
8                   SetTexture("texture\\shot");
9                   Radius = 4;
10          }
11          public void Set(Vector2 pos, Vector2 vel){略 }
12          public override void Update()
13          {
14                  Position += Velocity;
15                  if (IsOverWindow())
16                  {
17                          Flag = false;
18                  }
19                  ProcessHit();
20                  base.Update();
21          }
22          void ProcessHit()
23          {
24                  if (GetHit(Global.Game.Enemy))
25                  {
26                          Global.Game.Enemy.Damage();
27                          Flag = false;// 命中したショットを消す
28                  }
29          }
30  }
```

ソースコード 18　Bullet.cs

```
1   (using 略)
2   namespace TextGame2
3   {
4           public class BulletManager
5           {
6                   (中略)
7                   public void Clear()// 全弾消す
8                   {
9                           for (int i = 0; i < Bullets.Length; i++)
10                          {
```

```
11                                  Bullets[i].Flag = false;
12                          }
13                  }
14          }
15          public class Bullet : Character
16          {
17                  Vector2 Velocity;
18                  public bool Flag;
19                  public Bullet() : base()
20                  {
21                          SetTexture("texture\\bullet");
22                          Radius = 4;
23                  }
24                  public void Set(Vector2 pos, Vector2 vel){略 }
25                  public override void Update()
26                  {
27                          Position += Velocity;
28                          if (IsOverWindow())
29                          {
30                                  Flag = false;
31                          }
32                          ProcessHit();
33                          base.Update();
34                  }
35                  void ProcessHit()
36                  {
37                          if (GetHit(Global.Game.Ship))
38                          {
39                                  Global.Game.Ship.Damage();
40                          }
41                  }
42          }
43  }
```

ソースコード 19　Game1.cs

```
1   (using 略)
2   namespace TextGame2
3   {
4           public static class Global{略 }
5           public enum State
6           {
7                   Normal,
8                   Gameover,
9                   Clear,
10          }
11          public class Game1 : Microsoft.Xna.Framework.Game
12          {
13                  (フィールド宣言略)
```

```
14          SpriteFont Font;
15          public State State;
16          public Game1(){略 }
17          protected override void Initialize()
18          {
19                  base.Initialize();
20                  Ship = new Ship();
21                  Enemy = new Enemy();
22                  Bullets = new BulletManager();
23                  Shots = new ShotManager();
24                  State = TextGame2.State.Normal;
25          }
26          protected override void LoadContent()
27          {
28                  spriteBatch = new SpriteBatch(GraphicsDevice);
29                  Global.Sprite = spriteBatch;
30                  Font = Content.Load<SpriteFont>("font");
31          }
32          protected override void Update(GameTime gameTime)
33          {
34                  if (State == TextGame2.State.Normal)
35                  {
36                          Enemy.Update();
37                          Bullets.Update();
38                          Ship.Update();
39                          Shots.Update();
40                  }
41                  else
42                  {
43                          if (Keyboard.GetState().IsKeyDown(Keys.Escape))
44                          {
45                                  Exit();
46                          }
47                  }
48                  base.Update(gameTime);
49          }
50          protected override void Draw(GameTime gameTime)
51          {
52                  GraphicsDevice.Clear(Color.CornflowerBlue);
53                  Global.Sprite.Begin();
54                  Shots.Draw();
55                  Enemy.Draw();
56                  Bullets.Draw();
57                  Ship.Draw();
58                  switch (State)
59                  {
60                          case State.Normal:
61                                  break;
62                          case State.Gameover:
```

```
63                                          spriteBatch.DrawString(Font, "Game␣Over", new Vector2(120,
                                                200), Color.Black);
64                                          break;
65                                  case State.Clear:
66                                          spriteBatch.DrawString(Font, "GAME␣CLEAR", new Vector2(100,
                                                200), Color.Red);
67                                          break;
68                                  default:
69                                          break;
70                          }
71                          Global.Sprite.End();
72                          base.Draw(gameTime);
73                  }
74                  public void GameClear()
75                  {
76                          State = TextGame2.State.Clear;
77                  }
78                  public void GameOver()
79                  {
80                          State = TextGame2.State.Gameover;
81                  }
82          }
83  }
```

やたらと長いですが、難しいことはやっていません。当たり判定のコードは実質的に Character.cs の GetHit 関数だけで、他はダメージの処理やゲームオーバーの処理などが中心です。

GetHit 関数では円と円の中心間の距離の自乗を、Position 同士の引き算の結果で Vector2 の LengthSquared 関数を使用することで求めています。Position の X と Y それぞれについて手書きで計算するよりも用意された関数を使ったほうが見やすく楽なのでこうしました。

各キャラクターのコンストラクタでは当たり判定の半径を設定しています。敵やショットの当たり判定は見た目通りかそれよりも大きめ、敵弾の当たり判定は見た目よりやや小さめ、自機の当たり判定は見た目よりも極端に小さい[*33]、というのが弾幕シューティングゲームでよくあるスタイルなのでそれに倣いました[*34]。

Enemy ではショットが当たったときに表示が赤っぽくなるようにしました。Update → (もしあれば)Damage → Draw という順番で実行されるように書いたため、Damage が実行されたフレームだけ表示色が変わります。なお、色を変えるために Draw メソッドをオーバーライドして処理をカスタマイズしています。

当たり判定を行うときは、2 つのキャラクターのうちどちらが GetHit 関数を呼び出しても結果は変わりませんが、Ship が Bullet に対して GetHit を使うよりもその逆のほうが public にするものを減らせるような設計なのでそうしました。

Game1 クラスは当たり判定をすること自体ではソースコードを変更する必要はありませんでしたが、ライフの実装に伴いゲームオーバーとゲームクリアを実装したため色々変わっています。

まず 5–10 行目でゲームの状態を表す列挙体 State を定義して、15 行目で Game1 に State 型のフィールド State を持たせています。この State を使って処理を分岐させています。

[*33] 時には 1 ピクセルの点でしかないということも
[*34] その割には自機の当たり判定はやや大きいですが

Update メソッド内では、State が Normal の時は普通の処理を行い、そうでないとき、すなわちゲームオーバーかゲームクリア状態の時は処理を停止させて、エスケープキーが押されてゲームが終了するのを待機しています。

GameClear メソッドと GameOver メソッドはどちらも名前のとおりです。ゲームクリアなどで State を変更する以外にも処理を行うようになった時のことを考えて、Ship などのクラスから直接 State を操作するのではなくメソッドを呼び出すように設計しました。

それでは最後に SpriteFont について。スプライトフォントはゲーム画面に文字を描画する機能です。日本語の表示が面倒だったり描画位置の調整が行いにくかったり文字の装飾を行うのが困難だったりと使い勝手があまりよくないので、ノベルゲームのように多様な文字列を表示する必要があるときぐらいしか使う価値はありませんが一応紹介しています。

先ほどコンテントパイプラインに追加したのが文字の設定を記述するファイルで、Content の Load 関数を使って SpriteFont クラスとして読み込みます。描画するときには SpriteBatch の DrawString クラスを使います。注意点としてはこれも SpriteBatch での描画なので、テクスチャの描画と同様に Begin と End で囲まれた間に書く必要があること、描画する座標は文字列の左上の点を指定すること、です。

このプログラムでは Draw メソッド内で switch 文を使って State によって描画内容を分岐させています。

このサンプルプログラムでは解説の都合上、ショットを当てたときや弾を喰らった時の処理は非常にシンプルにしていますが、実際にゲームを作るときはエフェクトを付けたり無敵時間を付けたりといった処理を付けるべきですし、またゲームクリア後やゲームオーバー後の処理も色々と考えるべきでしょう。ライフの表示を行うなどの工夫も必要です。

4.4 ボスの攻撃をまともにする

それでは『遊べるようにする』編の最後に、シューティングゲームとして成立するようにボスの攻撃を強化してみます。

ソースコード 20　Enemy.cs の Enemy

```
1  public class Enemy : Character
2  {
3          int Count;
4          int Life;
5          Color Color;
6          float a;
7          float b;
8          public Enemy()
9                  : base()
10         {
11                 SetTexture("texture\\enemy");
12                 Position = new Vector2(320, 100);
13                 Radius = 32;
14                 Life = 80;
15                 a = 0;
16                 b = MathHelper.PiOver2;
17         }
18         public override void Update()
```

```
19          {
20                  Color = Color.White;
21                  Count++;
22                  Attack();
23                  base.Update();
24          }
25          void Attack()
26          {
27                  a += MathHelper.Pi / 110f;
28                  b += (float)Math.Cos(Count * MathHelper.Pi / 120) / 80f;
29                  if (Count % 10 == 0)
30                  {
31                          FireWay(3.0f, a, 5, MathHelper.TwoPi);
32                  }
33                  if (Count % 30 == 6)
34                  {
35                          FireWay(3.7f, GetShip(), 4, MathHelper.Pi / 6f);
36                  }
37                  if (Count % 5 == 0)
38                  {
39                          FireWay(4.5f, b, 2, MathHelper.PiOver4);
40                  }
41          }
42          float GetShip()
43          {
44                  return (float)Math.Atan2(Global.Game.Ship.Position.Y - Position.Y, Global.Game.
                        Ship.Position.X - Position.X);
45          }
46          void Fire(float speed, float angle)
47          {
48                  Global.Game.Bullets.Set(Position, new Vector2((float)Math.Cos(angle) * speed, (float)
                        Math.Sin(angle) * speed));
49          }
50          void FireWay(float speed, float angle, int way, float width)
51          {
52                  if (way == 1)
53                  {
54                          Fire(speed, angle);
55                          return;
56                  }
57                  float sw = width / (way - 1);
58                  float an = angle - width / 2f;
59                  for (int i = 0; i < way; i++)
60                  {
61                          Fire(speed, an);
62                          an += sw;
63                  }
64          }
65          (以下略)
```

```
66 }
```

ソースコード 21　Ship.cs の Ship の Attack メソッド

```
1  void Attack(KeyboardState ks)
2  {
3          if (ks.IsKeyDown(Keys.Z))
4          {
5                  Count++;
6                  if (Count % 10 == 0)
7                  {
8                          Fire(6.5f, −MathHelper.PiOver2);
9                          Fire(6.5f, −MathHelper.PiOver2 − MathHelper.PiOver4 / 6f);
10                         Fire(6.5f, −MathHelper.PiOver2 + MathHelper.PiOver4 / 6f);
11                 }
12         }
13 }
```

こんな感じにしてみました。攻撃をする部分が長くなってきたので Attack メソッドに分離しています。また単発の弾を撃つ Fire メソッドだけではやりにくいので、Way 弾[*35]を撃つ FireWay メソッドと、自機への角度を取得する[*36]GetShip メソッドを作成しました。それと、攻撃に使う値を保存するためにフィールド a, b を作成しました[*37]。

敵の攻撃パターンを作るのには別に決まりなどがあるわけではなく、勘で適当に作っているだけです。本格的にシューティングを作ろうと思う人がいるのなら、色々なゲームから学ぶのが一番でしょう。

なお、敵の攻撃を強化しすぎてなかなかダメージが通らないため、自機のショットも 3 倍にして強化しています。こういったバランス調整も勘で適当にやりましょう。

5　作り込む

あとはネット上の資料などを参考にがんばってください。

5.1　タイトル画面を作る

ゲームプレイを管理するクラスにこれまでの処理をまとめて、それと対になるようにタイトル画面を管理するクラスを作るといいかもしれません。

5.2　リプレイ機能を作る

ファイルの入出力機能を使ってキーの入力情報を保存/再生するといいかもしれません。

[*35] シューティング用語。ある範囲の角度へ扇状に一定間隔で等速の弾を撃つこと
[*36] シューティング用語で言う自機狙いを作るため
[*37] こういう見て意味が分からない変数名はあまり使うべきではありませんが、規模とめんどくささを鑑みてこうしています。

5.3 音をつける

XACT というものを使います。

6 終わりに

最後が打ち切りのような感じになりましたが、全てを解説することはとうてい不可能なことなので、自力でやり方を調べていくことが今後のゲームプログラミングでは必須です。情報源は検索すればそれなりに見つかりますが、代表的なものを以下に挙げてこの講座を終わりとします。

■XNA デベロッパーセンター — XNA Game Studio 4.0　http://msdn.microsoft.com/ja-jp/library/bb200104.aspx
公式の解説です。ライブラリのクラスの解説や基本的な技術の解説など多くの情報が掲載されていますが、ややわかりにくいのが欠点です。

■App Hub　http://create.msdn.com/ja-JP/
公式の XNA 開発者用サイトです。サンプルコードなどのラーニングコンテンツが豊富ですが、こちらもややわかりにくいことが多いです。

■ソーサリーフォース　http://sorceryforce.com/xna/index.html
一般の方のサイトです。Tips 形式ということになっていますが、XNA の基本的な使い方はほぼ網羅されていて、サンプルコードも豊富です。

■ひにけに XNA　http://blogs.msdn.com/b/ito/
日本人の XNA(元) 開発チームの方のブログです。Tips 的な解説が多めです。