プログラミング教育におけるゲームプログラム

Games in Computer Programming Courses

小林　健一郎
Ken-ichiro KOBAYASHI

（平成24年10月 2 日受理）

1．はじめに

2008年よりプログラミング初等教育（C++[1]）の題材として、ゲーム（ゲームの制作）を利用している。本学プログラミング教育における学生が到達すべき目標として、著者が想定しているものは以下のようなものである。

Level 1　基礎事項を理解し使える。
Level 2　簡単なプログラムを自力で書くことができる。
Level 3　簡単な設計を行い、設計に基づいてプログラムを書くことができる。

Level 1 の基礎事項とは、C++においては、cout、cin、getline、if文、switch文、for文、while文、do-while文、関数（既存のものの利用と自作）、クラス（既存のものの利用と自作）、継承、仮想関数、配列、vectorなどである。上記の目標は、標準的なものと考える。しかし、従来の教科書にあるような題材（サンプルプログラム）の利用は、今日の大学教育の場で難しくなってきている。その理由のひとつは、学生にとってそれらが「おもしろくないから」である。

たとえば、カーニハン・リッチー著「プログラミング言語C」[2] にあるサンプルは、「摂氏華氏対応表」「入力された文字数のカウント」「二分探索の実装」「スタックの実装」「クイックソートの実装」などである。本学 1 年次後期に実施する半期講義（プログラミング基礎、Level 1 とLevel 2 を目指す）の受講者は80〜100名程度であるが、その大部分は、上記のようなサンプルに興味を示さないのが実情である。

しかし、題材としてゲームプログラムを選択することで、多くの学生に興味を持たせ、Level 1 とLevel 2 の目標に到達する学生を 8 割程度（単位取得者数/受講者数）にすることができた。プログラミング基礎の最終課題は「300行を越すプログラム」であるが、これは、全くの初心からはじめる半期講義の成果としては悪くないと考える。

著者は、続く 2 年次の半期講義（システム設計基礎、受講者数60〜80名程度で、Level 3 を目指す）でもゲームを利用している。また、10名程度であるが、ゼミ研究ではDirectXとC++によるGUIゲーム制作を行っている。これらの講義の経験から、「プログラミング教育に利用しやすいゲーム」に関する知見が蓄積された。本小論ではそれを論じたい。

2．基礎教育におけるゲーム

学生の興味を引くプログラムとしては、「インタラクティブなもの」が良いと思われる。従来のサンプルには、「データを与えると、それを処理し解析した結果を示す」というものが多かった。これは、計算機のイメージに合うものだが、近年のコンピュータは、単なる計算機ではなく、「返事をするもの」であることが望まれている。サンプルプログラムをインタラクティブにすることで、学生に興味を持たせることができると考えるのである。

初年度に教えるプログラミング基礎において、著者は「Hello World」「あいさつ」「占い」「数当て」「モンスターを育てるゲーム」「アドベンチャー」といった題材を使った。

「Hello World」は、Hello Worldと画面に出力する有名なプログラムである。このプログラムは言うまでもなくインタラクティブではない。

「あいさつ」は、「相手に名前や年齢を聞き、それに対して適当なあいさつをかえすプログラム」である。これが、インタラクティブなプログラムの基礎になる。また、これにより、データ型、変数、入力、if文を教えることができる。「あいさつの仕方」を各自に工夫させることで、学生は、「動くプログラムを自分で書いた」と感じることができるようである。

「占い」は「内部で発生させた乱数に応じて異なる出力（大吉、吉など）をするプログラム」である。このプログラム自体はインタラクティブではないが、今後のゲーム製作に欠かせない乱数を教えることができる。また、出力する定型文章は複数あるので、配列やvectorを教えることもできる。

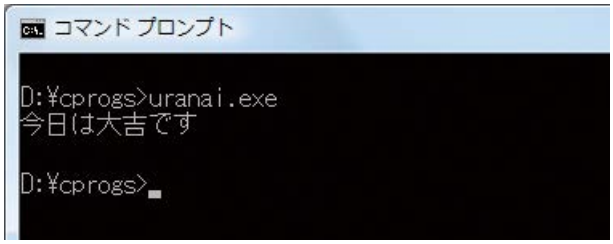

図２－１　占いの実行画面

「数当て」は「内部で発生させた乱数を当てるゲーム」である。これは、「占い」で学んだ乱数を応用した、インタラクティブなプログラムである。ここから、次第に「自分でプログラムを書く」ということを考えさせることができる。

「モンスターを育てる」で、クラス（モンスターのクラス）を教える。モンスタークラスには、はじめ、メンバ変数として「エネルギー」、メンバ関数として「歩く」などを持たせる。その後、メンバ関数に「食べる」「遊ぶ」などを付け加え、さらにコンストラクタやデストラクタも追加する。

```
class Monster
{
    int energy;
```

```
public:
    void walk();
    void eat();
    void play();
};
```

このクラスを利用するプログラムでは、クラスはMonsterのみとし、そのオブジェクトをmainで生成し利用する。このようにクラスを作っていくことで、オブジェクト指向プログラミングに入っていけると考えるのである。また、mainの中を工夫することで「ゲームらしさ」を増していくことができる。その際には、for文などを教えることができる。

```
int main()
{
    Monster m;
    int a;
    for(int i = 0; i < 100; ++i) {
        cout << "どうしますか？" << endl;
        cout << "1 食べる　2 歩く　3 遊ぶ　0 終了" << endl;
        cin >> a;
        cout << "\n";
        if(a == 1) m.eat();
        else if(a == 2) m.walk();
        else if(a == 3) m.play();
        else break;
    }
}
```

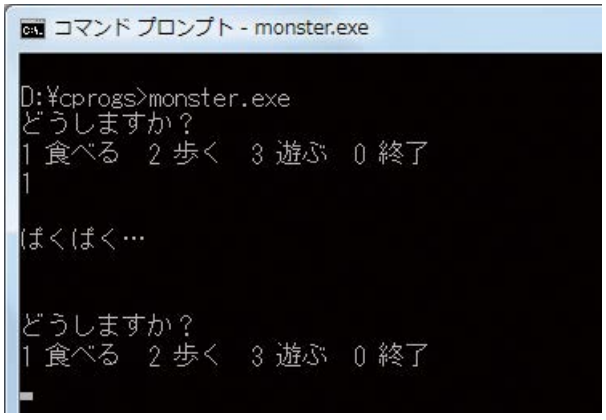

図２－２　モンスターゲームの実行例

更に、モンスタークラスの派生クラス（空飛ぶモンスターなど）を作り、継承や仮想関数を教えることができる。ただし、最初の半期講義では、仮想関数は紹介程度となり、実際的な使用例を示すことは難しい。

「アドベンチャー」は、「ユーザの選択によって場面が切り替わり、ユーザはゴール（ゴールの場面）を目指す」というゲームである。ただし、「場面」とは「はじまりの町」「静かな海」など、単に文字を表示するだけである。

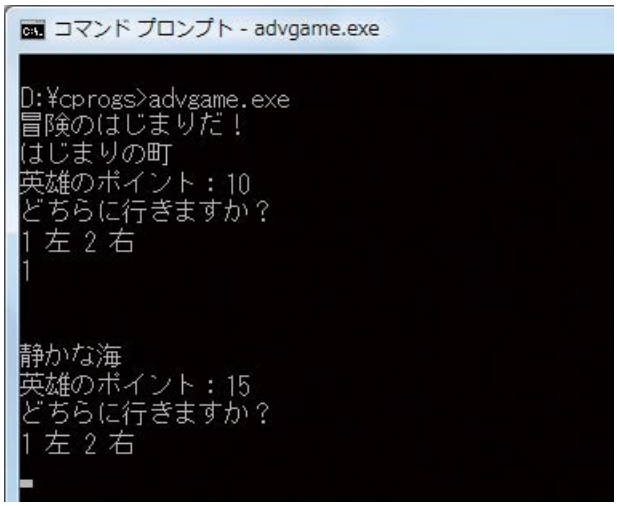

図２－３　アドベンチャーゲームの実行例

著者の授業用サンプルでは、「場面」をクラスとし、複数の「場面」を「アドベンチャー」の中に置くことにする。こうして、クラスの合成を教えることができる。また、このアドベンチャーの中に、モンスターや妖精などを置くことで、（簡単ながら）さらにオブジェクト指向的なプログラミングを身につけていくことができると考えるのである。

```
class Bamen
{
    …
};

class Adventure
{
    vector<Bamen> bs;
    vector<Monster> ms;
public:
    void play();
};
```

```
int main()
{
    Adventure a;
    a.play();
}
```

3．設計教育におけるゲーム

2年次に実施されるシステム設計基礎は、プログラミング基礎を受け、プログラムにおける設計を教えることになる。目標は、はじめに書いた「Level 3　簡単な設計を行い、設計に基づいてプログラムを書くことができる」である。

著者は、「3つのミニゲームをまとめたゲーム」「動物園ゲーム（動物園を管理するゲーム）」「スケジュール帳」などを題材にした。

「3つのミニゲームをまとめたゲーム」の3つのミニゲームとは、「アドベンチャー」「クイズ」「迷路」で、それらはプログラミング基礎で扱ったゲームか、それに類似するものである。ここで強調したことは、「小さいプログラムの統合」を、保守管理しやすい形で行うということである。

「動物園を管理するゲーム」は、「ユーザが動物園の園長になって動物の世話を管理する。世話の仕方によって動物の状態が変化し、それによって動物園の来訪客が増減し、動物園の収益が増減する」というものである。

これは、プログラミング初心者には、やや大きめのプログラムであり、そのようなプログラムにこそ設計が威力を発揮するというよい例になっていると考える。

「スケジュール帳」は「起動時に当日の予定と今後の予定を表示し、また、今後の予定を書き込めるプログラム」である。これは、ゲームではない。ゲームで行ったのと同様の考察で、このようなプログラムも書けることを示したのである。

以下、システム設計基礎で中心的なサンプルとした「動物園ゲーム」の詳細を説明する。

設計を記述するツールとしては、UML[3]を使う。著者が重視しているのは、要求分析時に使うユースケース図、設計時に使うクラス図とステートチャート図である。ユースケース図は、「そのプログラムがどのように使われるか」を直観的に書き表すものである。

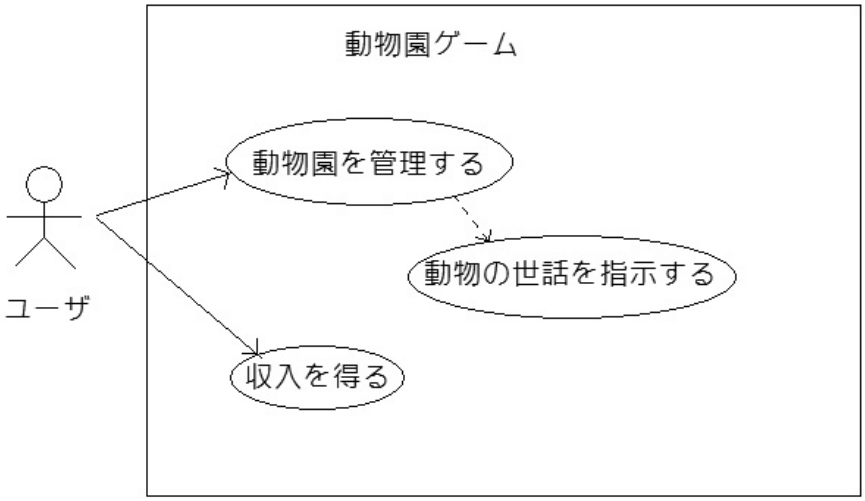

図３－１　動物園ゲームのユースケース図

ユースケース図には詳細が書き込めず、したがって補足が必要になる。そのため、「そもそもユースケース図は不要」という意見もある。しかし、著者は、他の多くのUML支持者と同様、開発者とユーザ・クライアントの双方が理解できる直観的な図として有用と考える。

ただし、図３－１は、あくまでも説明のためのサンプルであって、実際の要求分析では、当然より複雑なものを描く必要がある。しかし、半期講義ではあまり深く掘り下げることはできないので、続く講義における課題と考える。

クラス図は、そのプログラムの「登場物」の静的な構造や関係を記述する図であり、ステートチャート図は「登場物」の個々の状態変化を記述するものである。

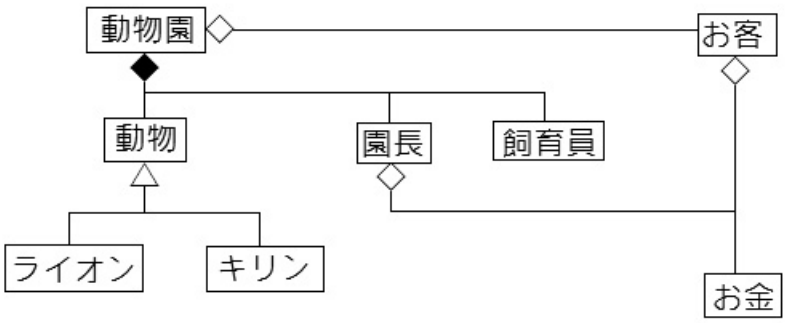

図３－２　動物園ゲームのクラス図

クラス図で、黒いひし形は「(本質的に）属している」、白いひし形は「(一時的に）属している」、白い三角は「一種である」などを表している。

オブジェクト指向型のプログラミングでは、このような図が設計となるのである。

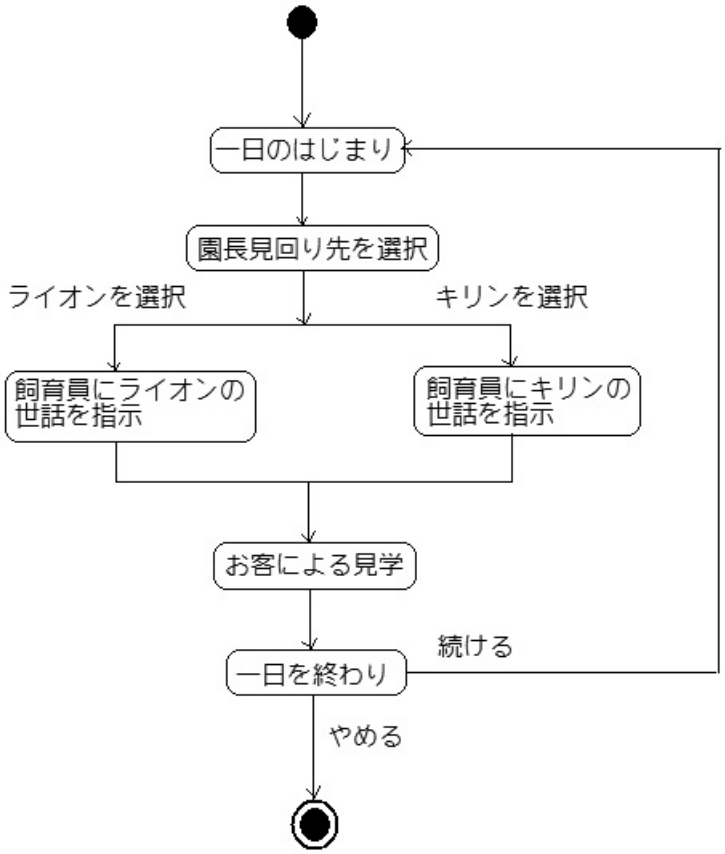

図３－３　動物園オブジェクトのステートチャート図

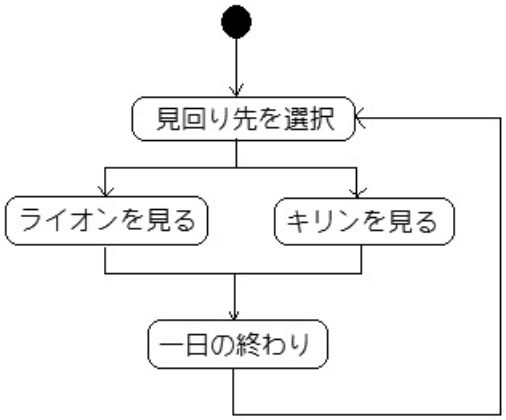

図３－４　園長オブジェクトのステートチャート図

ステートチャート図では、1重の黒丸がオブジェクトの開始、2重の黒丸がオブジェクトの終了を表している。ステートチャート図はフローチャート図に似ているが、個別のオブジェクトの状態変化（動作）を表す図であり、異なるものである。このように個々のオブジェクトの動作を切りだすことで、プログラミングがしやすくなる。たとえば、園長クラスのコードを書く場合、園長の動作のみを実装すればよいのである。そして、そのように作り上げたクラスを組み合わせて全体を作るのがオブジェクト指向型のプログラミング技法である。

この動物園ゲームは、オブジェクト指向型の設計、プログラミングを学ぶのに適した題材であると考える。

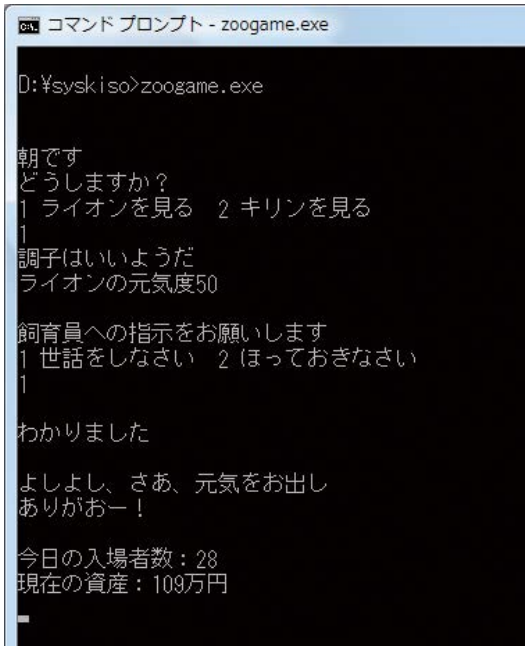

図３－５　動物園ゲームの実行例

４．GUIゲームにおける有用なツール

著者はゼミで、10人前後の学生にプログラミングを指導している。それは、Windows上でC++/DirectX[4]を利用するものであり、その目標はLevel 3およびその先である。ゼミ学生は、プログラミング基礎、システム設計基礎、システム分析等を履修し、1年半のプログラミング暦を持つ学生ということになる。ただし、GUIプログラムの経験はないため、ゼミでは、GUIプログラミングの指導にも時間を割かなければならない。

実際的なゲームになると、たとえば、「玉当てゲーム（玉を目標物にあてるゲーム）」のような単純なものであっても、その実装はそれほど簡単ではない。この場合、

・玉の表示

・玉の発射とその後の自動移動
・玉と目標物（的）の当たり判定
・当たった場合のアクション

などが必要になるからである。

図４－１　玉当てゲーム

プログラミングの初等教育の目標は、学生がこのようなプログラムを自力で書けるようになることと言えるだろう。実際、GUIやライブラリを理解しなければできないが、それを理解すれば、プログラミング暦１年半の学生でも十分書くことができる。学生が達成感をおぼえる程度のゲームとしては、「玉当て」部分だけではなく、開始画面や終了画面での操作なども作らせなければならないが、そのようなゲームも開発可能である。GUIの学習も含めると、だいたいその程度で１年間のゼミが終了してしまうことになるが、著者は、それ以上を望むものではない。１年半の初等教育を受けた学生には適当なサンプルであると思われる。

しかし、このような課題を与えると、その結果、「どの学生が作るゲームもほとんど同じようなものになる」ということが起こってしまう。「玉当てゲーム」に必要な要素はほとんどすべてのゲームプログラムに必要な要素であり、多くの場合、それを学習している間に時間切れになってしまう。つまり、「玉当てゲーム」より先に進む時間がなくなってしまうのだ。

プログラミング技術の獲得という意味では、それでもまったく問題ないと考えるのだが、それでは、学生の方でモチベーションを維持できないようである。「みなが同じようなゲームを作っている。そしてそれは前年度、前々年度の学生が作ったものとほぼ同じである」ということになるからだ。著者は、ゼミ学生の「オリジナルなものを作りたい」という気持ちは尊重したいと思う。そこで、多くの学生が共通して使いたいと思う「もの」は著者のライブラリ[5]で与えることにした。学生は、それらの「もの」を利用して、その上に独自のプログラムを書いていくのである。そもそもDirectXというライブラリを使っているのであり、その上に著者のライブラリを使っても本質的な違いはないと考える。当然、自ら試して学ぶべきプログラミングのテクニックも変わらない。

学生に与えるライブラリ（「もの」のコード）は、学生とともに考えたものである。学生が何か「もの」を必要と感じた場合、まず学生が開発し、後にそれが他の学生にも有用であると判断された場合、それを著者のライブラリに組み込んでいるのである。それによ

り、次の年の学生はそのコードを既存のものとしてスタート時から使うことができる。
　著者ゼミにおいて、作られた「もの」を順を追って説明する。

2007年度　ゲームフレーム（枠組み）
2008年度　静止画像オブジェクト、オブジェクトの当たり判定
2009年度　アニメーションオブジェクト、弾丸と発射台
2010年度　ゲームフレームの整理
2011年度　オブジェクトのジャンプ、状態を持つオブジェクト
2012年度　弾丸の改良、当たり判定の改良、ゲーム内ゲーム

　ゲームフレームは、クラスKGameFrameで、Windowsにおけるメッセージ処理とDirectXの基本処理をカプセル化したものである。これにより、学生独自のゲームのクラスは次のように書くことができる[5]。

```
class Game : public KGameFrame
{
    // ゲームの持つデータ
    •••
public:
    // ゲーム画面の大きさと背景画像を指定
    Game() : KGameFrame(800, 600, "bg.bmp") {}
    void init();                        // ゲームの初期化
    void onJoyStick(int i);             // ジョイスティックによる操作
    void onKeyDown(WPARAM w); // キーボードによる操作
    void show();                        // 処理と表示
    •••
};
```

　カプセル化とは、「複雑だが定型のコードを関数などにまとめてしまい、ライブラリのユーザはそれを呼び出すだけにする」という意味である。これによりゲームの構造が見やすくもなる。
　静止画像オブジェクトは、単純に画像を画面に表示するものである。これはGUIゲームには必須であると思われる。
　オブジェクトの当たり判定とは、「２つのオブジェクトが重なっているかどうか」を判定するものである。これは座標を調べるだけであり、プログラムとしては難しくないが、数値の処理を苦手とする学生も多く、また、バグを作ってしまった場合、気づきにくい。そのため、静止画像オブジェクトのメンバ関数として当たり判定関数をライブラリに加えた。これはオブジェクトを四角形と考え、その四角形の重なりを判定するだけである。そのため、より細かい当たり判定を行いたい学生は、さらに独自の当たり判定コードを書いている。たとえば、人体の画像A、Bの２つが重なり合う場合、「Aの手とBの足が接触し

ているか」など、きめ細かく判定するのには、独自の工夫がいるのである。

ゲームのフレーム、静止画像、当たり判定を使うことで十分にゲームを作ることができる。その後に多くの学生が望んだのは、アニメーションであった。アニメーションは静止画像を短い時間で取り変えることで実現できる。したがって、静止画像を表示できれば容易に実装でき、実際、初期のゼミ学生たちは、そのようにコードを書いていた。しかし、ゲームにおけるアニメーションは、実は、「華を添える存在」であっても、それ自体がゲーム内できわめて重要であることは少ない。そこで、これもカプセル化して簡単に使えるようにしたのが、アニメーションオブジェクトである。

初心者が作り始めてすぐに「おもしろい」と思えるゲームは、前述のような玉当てゲーム、あるいは、敵味方で玉を射ち合うシューティングゲームである。そのようなゲームを簡単に開発できるよう「弾丸と発射台」もライブラリに入れた。

発射台は弾丸を発射し、弾丸は発射された方向に等速直線運動をし、弾丸とオブジェクトの当たり判定をすることができる。そして、弾丸がオブジェクトに当たれば、当たりのアクション（アニメーション）を表示するものである。最初のバージョンでは弾丸は静止画像であったがその後改良して弾丸もアニメーションにできるようにした。

この「弾丸と発射台」は、著者のライブラリの中で、ただひとつ「具体的なもの」である。それ以外は、ゲームフレーム、静止画オブジェクト、アニメーションオブジェクトなど一般的なものなのである。実は、「弾丸と発射台」は、はじめ「学生に提供するもの」としてではなく、著者のライブラリの使用例として作ったものである。しかし、ゼミ学生の多くは、このコードを（もちろん、許可を得て）流用してゲームを作るようになった。前述のように、これを独自に実装していると、なかなかその先に進めないからである。そこで、これをそのままライブラリに加えたのである。つまり、現在のライブラリでは、図４－１の内容は自分で実装せずに実現できるのである。

著者は、これにより、「学生が工夫をする機会を減らした」とは考えない。「弾丸と発射台」を使う学生のほとんどは、シューティング自体を目的とせず、シューティングを内部に含む（もっと大きな）ゲームを考えるからである。逆に、弾丸の様態に興味を持った学生は、ライブラリのコードを参考にしながらより高機能の「弾丸と発射台」を開発している。

著者のライブラリ中、「具体的なもの」は「弾丸と発射台」のみであるが、あとひとつ一般的とは言えない機能として「ジャンプ機能」をすべてのオブジェクトに付与した。これは、「ゲーム画面中でオブジェクトを上に向かってジャンプさせると、あたかも重力を感じるように、もとの高さに戻ってくる」という機能である。

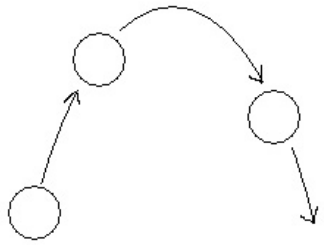

図４－２　オブジェクトのジャンプ機能

著者のライブラリでは、すべてのオブジェクトを一定時間ごとに再描画するように作ってあるので、その処理の間で、縦方向の座標を変化させていくことでこれを容易に実現することができる。その座標計算は、縦方向の座標をy、縦方向の速度をvy、重力加速度をgとすると、一定時間ごとに

```
y += vy;
vy -= g;
```

を実行すればよいだけである。これは物理法則を単純にコード化しているもので、かなりリアルに見えるものである。

しかし、「決められた高さに一定のステップで到達し、それからもとの高さに戻る」ように、初速度とg（仮想世界なのでプログラマが決める）を決めるにはちょっとした数学的思考が必要となる。そこで、この機能もはじめからつけることにしたのである。

ライブラリの改良に伴って、学生が作るゲームのクオリティも上がってきた。おそらくその結果、学生がほしいと思う「もの」や機能も変わってきている。たとえば、弾丸のアニメーション化などがそうであるが、特に大きなものは、「状態を持つオブジェクト」である。

たとえば、「仮想世界の中を主人公が旅をする」というアドベンチャーゲームを考えたとしよう。その場合、一般には、主人公オブジェクトが画面内を移動するように作るだろう。ゲームの作り始めにおいては、主人公の画像は、「正面を向いた静止画」のみでよいと考える学生が多い。しかし、ゲームの骨格が完成すると、それに対し「横を向いた静止画」や「歩いているアニメーション」も欲しくなる。つまり、同じ主人公が、「正面を向いて止まっている状態」「横を向いて止まっている状態」「横を向いて歩いている状態」など、いくつもの状態を取るようにし、それぞれの状態に応じて表示する画像やアニメーション、また、処理を変えるようにしたいのである。

もちろん、このような状態の変化を学生自身が扱うことは困難ではない。実際、それは前節で見た「園長オブジェクト」の実装と同様である。しかし、多くのオブジェクトについてそれを実装しだすと時間がかなりかかることから「状態を持つオブジェクト」をライブラリに組み込むことにしたのである。

基本的なアルゴリズムは、オブジェクトに状態を表す変数を持たせ、その変数の値に応じて、画像や処理を変える、というものである。「状態を持つオブジェクト」のクラスKOjectXの概要は次のようになる。

```
class KObjectX : public KObject
{
    •••
    int snum_;      // 状態を表す変数
public:
    KObjectX() : snum_(0) {}
    ~KObjectX();
```

```
    void addSState(•••);        // 静止状態の追加
    void addAState(•••);        // アニメーション状態の追加
    void setStateNum(int n);   // 状態番号を指定して状態の変更
    •••
};
```

ここでKObjectは従来のオブジェクトのクラスである。KObjectXは、KObjectと状態番号をカプセル化したクラスと言うことができる。

なお、従来のオブジェクトKObjectと混在させている理由は、

・過去に書かれコードを流用したい。
・学生によっては「状態を持つオブジェクト」を必要としない。

である。

ライブラリの最新の改良は「オブジェクトの当たり判定の改良（拡張）」である。たとえば、「移動するオブジェクト（アドベンチャーゲームの主人公など）が他のオブジェクト（岩など）にぶつかる」などということを表現したいことがある。その場合、たとえば、「主人公は岩にぶつかりその方向には進めなくなるが、もとの方向には戻れる」などとしたい。

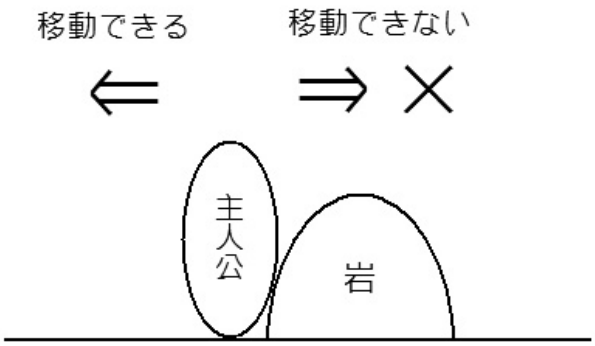

図４－３　岩にぶつかった主人公

これを従来の当たり判定のみで行うとコードが少し複雑になる。たとえば、「主人公は岩にぶつかるとその方向に進めなくなる」を「主人公が岩と重なったら動きを止める」とするのは簡単である。しかし、そうすると、「もとの方向には戻れる」が実現できなくなる。そのとき、主人公が岩と接触しているためである。「主人公が岩と重なったら、岩のある方向への動きを止める。他の方向へは動ける」としなければならないのである。その場合、主人公と岩が単純に重なっているかを調べる当たり判定ではなく、「どちらの側で重なっているか」を調べなければならなくなる。結局、これも座標計算になるのである。

そこで、座標計算を避けるため、オブジェクトの当たり判定を改良したのである。KObjectには、次のようなメンバ関数をつける仕様にした。

```
    bool intersect(const KObject& o) const;
    bool intersect(int dx, int dy, const KObject& o) const;
```

上のintersect()は従来の当たり判定関数で、引数に与えられたオブジェクトと自分が重なっていた場合にtrueを戻すものである。下の方は、「自分がx方向にdx、y方向にdy動いたら、引数に与えられたオブジェクトと重なるか」を判定する関数である。この関数を使うことにより、「主人公が進むと岩に当たってしまう場合、そちらには進めない」とすることで先の困難を回避できるのである。

次に「ライブラリに入れるか入れないか」の判断について、例を述べる。

KGameFrameは最初の年から与えたものであるが、その後、学生から「ゲームには再開（再起動）機能があった方がよい」という意見が多数出た。著者ははじめ「ゲームが終了すればプログラム自体も終了させればよい」と考え、そのようなフレームにしていたのだが、ゲーム終了後にもウィンドウを閉じず、そのままゲームを再開させるオプションを持たせたいと考える学生が多かったのである。

多くの学生がそれぞれの工夫で「再開オプション」を実装したため、「再開の枠組み」はライブラリの追加候補になった。はじめに述べたように、多くの学生に共通するニーズがあるならば、その年度は各学生に実装させるが、次年度のライブラリに私が書いたコードを追加するという方針である。しかし、それはライブラリには組み込まなかった。「ゲームの再開」とは、プログラム的には「再初期化」ということである。多くの場合、最初期化は、「『終了時のデータ』の初期データからのずれ」を補正することになる。もしこれをゲームフレームの中にいれるなら、たとえば、コードは次のようになるだろう。

```
class Game : public KGameFrame
{
    •••
public:
    •••
    void init();            // ゲームの初期化
    void reset();           // ゲームの再初期化
    •••
};
```

ただし、reset()の処理はinit()で行う処理の一部と同じはずである。したがって、コードの重複を避けるためには、init()内でreset()を呼び出すようにすべきである。これは、プログラミング手法としてはよく知られたものである。しかし、「ゲームクラスでは、reset()を作り、必ずinit()内でそのreset()を呼び出すこと」という利用規約を作れば、それはわずらわしいものになるだろう。もちろん、これをカプセル化し自動化する（したがって、学生には見えなくする）ことも可能である。しかし、そこまでカプセル化してしまうと、ゲームフレームの全体像がつかみにくくなり、学生への教育効果が薄くなると考える。

そこで、「ゲームの再開」は学生各自が実装するように残したのである。

ゲーム内ミニゲームのフレームについては、現在検討中である。前節で述べたようにCUIプログラムでは簡単である。しかし、GUIプログラムでは、さまざまな入力がランダムに来るため難しくなる。たとえば、ユーザは、ボタンやキーを好きなタイミングで押すと考えられる。そのすべての情報を、すべてのミニゲームに送るようにすると、各ミニゲームが大きな情報を受け取る仕様になる。しかし、具体的なミニゲームで必要な情報は限られている。たとえば、「ゲームコントローラの特定のボタン以外の情報は要らない」のが普通だと思われる。その場合、「すべての情報を受け取る仕様」は無駄であるように思われる。しかし、ミニゲームの汎用フレームを作るなら、はじめから受け取る情報を限定することはできない。

この部分の仕様をどうするかはまだ未定であり、今のところ、学生の各自の実装に任せている。

5．まとめ

本小論では、プログラミング教育における「ゲーム制作の活用」を紹介した。著者が主張することは、

・ゲーム制作が学生の興味を引くことが多い。
・ゲーム制作により一通りのプログラミング技法を学ぶことができる。

ということであり、

・教育用として有用なゲームアルゴリズムがある。

ということである。

特に、第3節、第4節で紹介した題材は、近年の学生を対象にするプログラミング教育において、かなり普遍的に役に立つものと思われる。

なお、繰り返しになるが、著者は、学生にゲームを推奨しているわけではない。結果として、ゲーム以外のプログラムに興味を持ってくれることも「良い成果」と考えている。

謝辞

いつもご指導ご鞭撻いただける本学教員のみなさまに感謝致します。

付録

2年次後期実施のシステム分析の授業で、「授業にゲームを使うことに関する感想」を含むアンケートを取った。対象となる学生は、以前に「プログラミング基礎」「システム設計基礎」を受講し、2012年現在に「システム分析」を受講している学生である。

有効総数39名
プログラミングは好きですか？

とても好き	好き	どちらとも	嫌い	とても嫌い
2名	13名	18名	4名	2名

「どちらとも」は、正確には「どちらとも言えない」である。以下同様。
履修の都合上「プログラミングを好きではないが履修している」という学生もいる。

ゲームを題材に使うことをどう思いますか？

とても良い	良い	どちらとも	悪い	とても悪い
5名	21名	12名	1名	0名

「プログラミングは嫌い・とても嫌い」と答えた学生6名のうち、ゲームを題材に使うことを「良い」と答えたのは3名、「どちらとも言えない」と書いたのは3名だった。
「ゲームを題材に使うことは悪い」と答えた学生も1名いた。この学生は、「プログラミングが好き」と回答し、以下に示す「取り上げてよかったサンプルプログラム」では、ゲームを2つ選んでいる（スケジュールは選んでない）。

取り上げてよかったサンプルプログラムを選んでください。（複数回答）

占い	1名	数当て	3名
モンスターゲーム	8名	アドベンチャー	16名
クイズ	6名	迷路ゲーム	13名
動物園ゲーム	8名	スケジュール	8名

スケジュールのみ、ゲームではない。
主なサンプルとして使ったのは、「クラス利用の入門」にモンスターゲーム、「クラス利用のまとめ」にアドベンチャー、「システム設計のまとめ」に動物園ゲームである。あまり時間を割いていない迷路ゲームの人気が高かったのは著者には意外であった。
また、「他に取り上げてほしいサンプル」を聞いたところ、ゲームが3つ、ネットワークプログラムが2つ上がった。

参考文献

[1] B.Stroustrup『プログラミング言語C++』アスキーアジソンウェスレイ
[2] B.W.カーニハン、D.M.リッチー　『プログラミング言語C』共立出版
[3] Unified Modeling Language、http://www.uml.org/
[4] マイクロソフト社のゲーム等のためのライブラリ、
http://msdn.microsoft.com/library/windows/apps/hh452744.aspx
[5] 小林健一郎「オブジェクト指向設計について」静岡産業大学情報学部紀要第1号p87