[バンタン体験授業(2021年01月版)]

シューティングゲームを作ってみよう

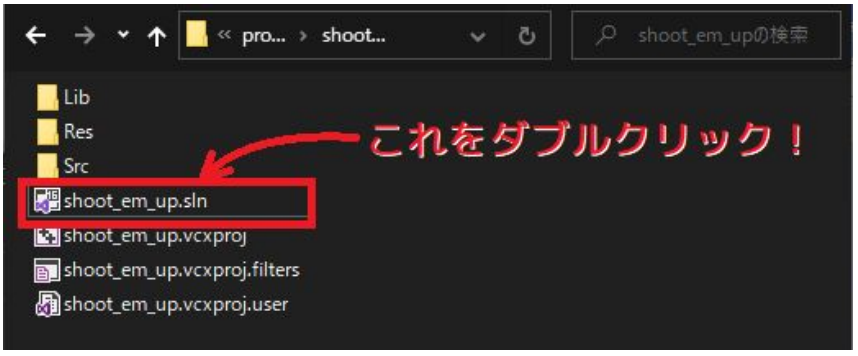
1 Visual Studioを起動する

1.1 SLNファイルを開く

このテキストでは、「C++(シー・プラス・プラス)」というプログラミング言語を使って「隕石を撃ち落とすシューティングゲーム」を作っていきます。また、ゲームを作るために「Visual Studio(ビジュアル・スタジオ)」というツールを使います。

さっそくVisual Studioを起動しましょう。デスクトップの「shoot_em_up(シューテム・アップ)」というフォルダをダブルクリックして開いてください。

その中に「shoot_em_up.sln(シューテム・アップ・エス・エル・エヌ)」というファイルがあります。次はこのファイルをダブルクリックしてください。
するとVisual Studioが起動します。

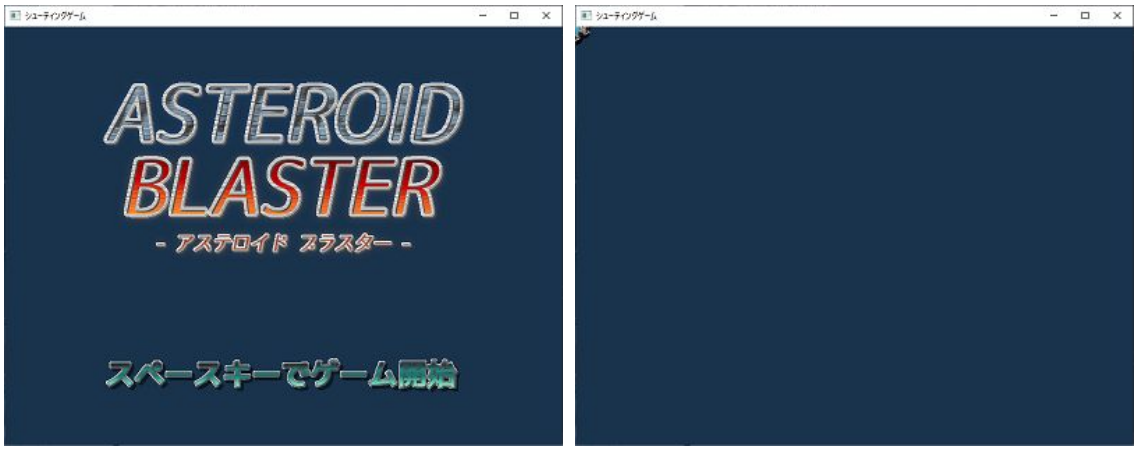


1.2 プログラムを実行する

Visual Studioを起動したら、もうプログラムを実行する準備はできています。
プログラムを実行するには、上の方にある「ローカルWindowsデバッガー(ローカル・ウィンドウズ・デバッガー)」というボタンをクリックします。

しばらく待つと、プログラムが実行されて新しいウィンドウが開き、ゲームのタイトル画面が表示されます。

タイトル画面でスペースキーを押すと、ゲーム画面が表示されます。



左上に何かが表示されているようですね。でも、まだ何もプログラムを書いていないので動かすことはできません。

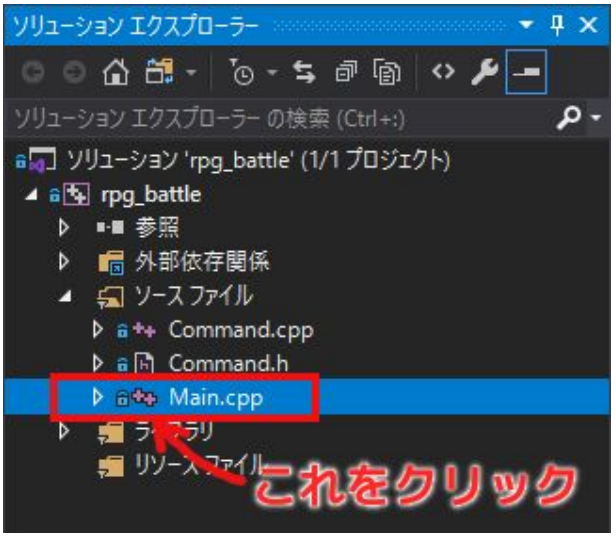
プログラムを終わらせるには、ウィンドウ右上の「×」ボタンをクリックしてください。

2 プログラムを見てみよう

2.1 Main.cppを開く

さきほど実行したプログラムは、どのような仕組みになっているのでしょうか。
それを調べるためには、「**Main.cpp**(メイン・シー・ピー・ピー)」というファイルを開かなくてはなりません。

Visual Studioの右側に「**ソリューション エクスプローラー**」というウィンドウがあると思います。その中に「**ソース ファイル**」というフォルダアイコンがあり、さらにその下に「Main.cpp」と書かれた行があるのが分かるでしょうか。この行をダブルクリックするとMain.cppを開くことができます(**見つからないときは、項目の左側にある「▷」をクリックしてみてください**)。



2.2 C++言語

Main.cppは「**C++(シー・プラス・プラス)**」という言語で書かれています。
コンピューターのプログラムは「プログラミング言語」と呼ばれるもので書かれま

す。C++言語もプログラミング言語のひとつで、家庭用ゲームをはじめ、Windowsなど様々なソフトの開発に使われています。

みなさんは主に日本語を話すと思いますが、世界では、国や地域によってさまざまな言語が使われています。それと同じように、コンピューターの世界にもさまざまな言語が存在します。

2.3 見間違えやすい文字に気をつけよう

これから、Main.cppを見ていくわけですが、その前にいくつか注意点を伝えておきます。プログラムでは**見間違えやすい文字**を使います。以下の文字は特に見間違えやすいので気をつけて読み書きしてください。

- 1(いち)、i(小文字のアイ)、I(大文字のアイ)、l(小文字のエル)、|(垂直線)
- 0(数字のゼロ)、o(小文字のオー)、O(大文字のオー)

他にも、アルファベットの大文字小文字は見間違えやすいものです。注意して読むようにしてください。なお、このテキストで使われているのはほとんどが小文字です。

2.4 プログラムを書くときは「半角モード」を使おう

パソコンで日本語を入力するには、キーボード左上にある「**半角/全角**(はんかく/ぜんかく)」キーを押して「**全角モード**」にします。日本語の入力を漢字に変換するには、キーボード中央下にある細長い「**スペースバー**」を押します。押すたびに変換候補が切り替わります。キーボード右側の「**Enter**(エンター)」キーを押すと、表示された漢字が決定されます。もう一度「半角/全角キー」を押すと「**半角モード**」に戻ります。

C++のプログラムを書くときは、常に**半角モード**を使います。日本語を使えるのは、基本的にはコメントと文章の部分だけです。うっかり全角モードのままプログラムを書いてしまったときは、落ち着いてEnterキーの上の「**backspace**(バックスペース)」キーを押して全角モードの文字を消し、半角モードで書きなおしましょう。

2.5 インクルード指令とユージング指令

```
1 #include "ShootEmUp.h"
2
3 using namespace ShootEmUp;
4 using namespace std;
```

それでは、プログラムを見ていきましょう。プログラムの先頭にある「**#include**(シャープ・インクルード)」は、**インクルード指令**といいます。インクルード指令を使うと、プログラムで使う部品を追加することができます。

実は、現在のC++言語には、画像や音声を扱う機能がありません。そこで、このテキストのために、それらを扱う部品を追加しています。「**ShootEmUp.h**(シューテム・アップ・ドット・エイチ)」というのは、**部品の書かれたファイルの名前**です。

ファイル名は「**"**(ダブル・クォーテーション)」という記号でかこみます。C++でファイル名を書くときは、**ダブル・クォーテーションで囲む**必要があります。

3、4行目の「**using namespace**(ユージング・ネームスペース)」は**ユージング指令**といいます。C++で部品を使うときは「グループ名::部品名」のように書くのですが、毎回グループ名を書くのは面倒です。

そこで、ユージング指令で指定したグループについては、グループ名を省略して部品名だけ書けば動くようになっています。グループ名はファイル名とは違うので「**"**」で囲んではいけません。

2.6 コメントと変数(へんすう)

```
 6| // 自機の変数
 7| double jiki_x;
 8| double jiki_y;
 9|
10| // 敵の変数
11| double teki_x;
12| double teki_y;
13|
14| // 現在の画面番号
15| //   0 = タイトル
16| //   1 = ゲーム
17| //   2 = ゲームオーバー
18| int scene_number = 0;
```

6,10,14～17行目の緑色の部分は「**コメント**」といいます。コメントは人間がプログラムを読みやすくするためのものです。**コンピューターはコメントを無視します。**
コメントは「**//**(ダブル・スラッシュ)」で始まり、改行で終わります。

変数とは、名前のとおり**変更可能な数**のことで、さまざまなデータを記録するために使います。変数は「**型　変数名;**」の形で作成します。「**型**(かた)」は変数の性質を表す名前で、上記のプログラムでは以下の2種類の**型**を使っています。

double(ダブル)	小数を使える。
int(イント)	整数しか使えない。

変数の使い方はのちほど説明します。

2.7 タイトル画面のプログラム

```
23| void title()
24| {
25|   if (get_key(' ') == 1) {
26|     scene_number = 1; // ゲーム画面に切り替える
27|
28|     // ゲーム開始時の設定
```

```
29      teki_x = 0;
30      teki_y = 0;
31      jiki_x = 0;
32      jiki_y = 0;
33    }
34
36    // 画像を表示
36    draw(400, 300, "logo_title.png");
37  }
```

この部分は「**title**(タイトル)」という名前が示すように、タイトル画面を制御するプログラムです。C++では「**関数**(かんすう)」というものを組み合わせてプログラムを作っていきます。関数は

```
型 関数名(パラメータ)
{
  プログラム
}
```

の形で作成します。名前こそ数学の**関数**と同じですが、実際には「**手順書**」と考えてください。プログラムの中に関数名を書くと、波括弧 {} の間に書いてある**プログラム**が上から順番に実行されます。

パラメータは手順書に書けない部分を、手順を実行する段階で指定する機能です。例えば「コンビニにプリンを買いに行く」関数があるとしましょう。プリンを買うにはお金が必要ですが、いくら持っているかは時と場合によりけりです。そこで、所持金はパラメータとして設定するわけです。

title関数の下には「**game**(ゲーム)」関数と「**gameover**(げーむおーばー)」関数があります。これらの3つはスペースキーを押して切り替わる3種類の画面に対応しています。

2.8 メイン関数とゲームループ

```
 3  // ここからプログラムの実行が開始される
 4  int main()
 5  {
```

gameover関数の下には「**main**(メイン)」関数があります。C++プログラムはこの関数から実行が始まることになっています。

ゲームは1/60秒ごとに画面を書き換えることで、画像が動いているように錯覚させています。これを行うのが**ゲームループ**というプログラムです。

```
  // ゲームループ
  for (;;) {
    update();

    // シーン番号に対応する関数を実行
    if (scene_number == 0) {
```

```
      title();
    } else if (scene_number == 1) {
      game();
    } else if (scene_number == 2) {
      gameover();
    }

    render();
  }
```

「for(フォー)」はプログラムを繰り返し実行する命令です。{}の内側のプログラムが何度も繰り返し実行されます。「update(アップデート)」関数で1/60秒たつまで待機し、「render(レンダー)」関数で画面を描きます。

その間にある部分は「scene_number(シーン・ナンバー)」という変数の値によって、異なる関数を実行するプログラムになっています。くわしい説明は省きますが、

- ゲームプログラムは1/60秒単位で関数を繰り返し実行している
- scene_numberに代入する値によって、表示される画面が異なる

の2点だけ覚えておいてください。

3 プログラムを改造しよう

3.1 自機と敵の表示座標を変更する

プログラムの解説だけでは退屈でしょう。ここからは少しずつプログラムを追加して、ゲームらしく変えていきます。さて、ゲームを開始して最初に気づくのは、自機が左上に表示されていることでしょう。

これはtitle関数で「変数」のjiki_xとjiki_yに0が代入されているからです。まずは、自機が画面の中央下付近に表示されるように改造しましょう。次のようにjiki_xとjiki_yに代入する値を変更してください。

```
28|     // ゲーム開始時の設定
29|     teki_x = 0;
30|     teki_y = 0;
31|     jiki_x = 400;
32|     jiki_y = 500;
33|   }
```

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。自機が画面の中央下付近に表示されていたら成功です。

左上にまだ何か表示されていますが、これは破壊対象の「隕石」です。こちらも座標を変更して、中央上に表示されるようにします。次のようにteki_xとteki_yに代入する値を変更してください。

```
28      // ゲーム開始時の設定
29      teki_x = 400;
30      teki_y = 100;
31      jiki_x = 400;
32      jiki_y = 500;
```

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。隕石が画面の中央上付近に表示されていたら成功です。

3.2 隕石を動かす

次は隕石を動かします。game関数に次のプログラムを追加してください。

```
42 void game()
43 {
44   // 隕石の移動
45   teki_y += 3;
46
47   if (get_key(' ') == 1) {
48     scene_number = 2; // ゲームオーバーに切り替える
```

「**teki_y += 3**」は「**teki_y**変数に3を足す」という意味です。
「**+=**(プラス・イコール)」記号は、左辺の変数に右辺の値を加算します。このプログラムは1/60秒ごとに繰り返し実行されるため、1秒後の**teki_y**は180増えて280になっているはずです。

行の最後には「**;**(セミコロン)」という記号があります。C++では、行の終わりにセミコロンを付ける決まりになっています。セミコロンはとても重要です。**行の終わりには忘れずにセミコロンを付けるようにしましょう。**

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。隕石が画面下に向かって移動していったら成功です。

《コラム》計算に使う記号

C++言語の足し算と引き算は、算数や数学と同じく「+」と「-」を使います。しかし、掛け算は「×」のかわりに「＊」(アスタリスク、星印)、割り算は「÷」のかわりに「/」(スラッシュ、斜線)を使います。これは、古いコンピューターは性能が低くて、掛け算や割り算の記号が使えなかったことが原因です。
そのため、当時のプログラミング言語の開発者たちは、掛け算には、数学の乗算記号である「・」(ドット、中点)に見えなくもない「＊」を、割り算には、分数に見える「/」を使うことに決めました。
それがC++言語にも受け継がれているのです。

3.3 if(イフ)はプログラムの分岐点

隕石が画面下に消えたら、上から戻ってくるようにしましょう。

```
44    // 隕石の移動
45    teki_y += 3;
46    if (teki_y > 600) {
47      teki_y = 0; // 隕石を上に戻す
48    }
49
50    if (get_key(' ') == 1) {
51      scene_number = 2; // ゲームオーバーに切り替える
```

追加した3行は「**teki_y**変数の値が600より大きいならば、**teki_y**に0を代入する」という処理を行います。

「**if**(イフ)」は**条件によってプログラムを分岐させる命令**です。**if**は

```
if (条件) {
  プログラム
}
```

の形で作成します。**条件が成立したらAのプログラム**が実行されます。**条件**には「等式」や「不等式」を指定することができます。式に使える記号を以下に示します。

意味	C++での書き方
小なりイコール(≦)	<=
大なりイコール(≧)	>=
小なり(<)	<
大なり(<)	>
等しい(=)	==
等しくない(≠)	!=

「ある数値以下」という条件は、数学では「≦」と書きます。しかし、C++にはこの記号がないので、代わりに「<」と「=」を横につなげて「<=」のように書きます。このとき、**2つの記号の間に空白を入れない**ように注意してください。

それから、C++で**「等しい」を表すときは、イコールを2つ書かなくてはなりません。**なぜなら、**イコール１つは「代入」を表すから**です。これは忘れやすいので、しっかり覚えておきましょう。

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。隕石が画面下に消えたあと、画面上に出現したら成功です。

3.4 自機と隕石の衝突を調べる

「自機と隕石の衝突」は、「自機から隕石までの距離D」が「自機と隕石の半径を足した長さR」より短い場合に発生します。

ゲーム空間にはX軸とY軸があるので、その両方でDがRより短い場合は衝突していることになります。これは二重の**if**を使って調べることができます。

自機画像の大きさは縦32、横32ドット、隕石画像は縦64、横64ドットなので、半径の合計は「16+32=48」です。ただし、隕石も自機も、画像の大きさより少し小さく描かれています。そこで、実際の半径は、合計より少ない32とします。

隕石の移動プログラムの下に、次のプログラムを追加してください。

```
46  if (teki_y > 600) {
47    teki_y = 0; // 隕石を上に戻す
48  }
49
50  // 自機と隕石の衝突
51  if (abs(teki_x - jiki_x) < 32) {
52    if (abs(teki_y - jiki_y) < 32) {
53      scene_number = 2; // ゲームオーバーに切り替える
54    }
55  }
56
57  if (get_key(' ') == 1) {
58    scene_number = 2; // ゲームオーバーに切り替える
```

「**abs**(エービーエス)」は絶対値を計算する関数です。距離を比較する場合、マイナスの値は都合が悪いので、絶対値を使って計算します。

二重のifによって衝突していることが分かったら、ゲームオーバーに切り替えます。「**scene_number**(シーン・ナンバー)」変数に2を代入すれば、ゲームオーバーに切り替わります。

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。隕石が自機に衝突してゲームオーバーになったら成功です。

3.5 隕石を回避する

次は、隕石と衝突しないように自機を移動できるようにします。Aキーで左、Dキーで右に移動させましょう。キーの状態を調べるには「**get_key**(ゲット・キー)」関数を使います。隕石を移動するプログラムの下に、次のプログラムを追加してください。

```
47    teki_y = 0; // 隕石を上に戻す
48  }
49
50  // 自機の移動
51  if (get_key('A') > 0) {
52    jiki_x -= 6;
53  }
54  if (get_key('D') > 0) {
55    jiki_x += 6;
56  }
57
58  // 自機と隕石の衝突
59  if (abs(enemy_x - player_x) < 32) {
```

「get_key(ゲット・キー)」はパラメータで指定した文字キーの状態を調べる関数で、次のように書きます。

```
get_key(調べる文字キー)
```

文字キーの状態は以下の3つです。

get_keyの結果	内容
0	押されていない
1	押されている(押された瞬間)
2	押されている

この表から、結果が0より大きければキーが押されていることが分かります。また、「' '」のように半角空白を使うと、スペースキーの状態を調べることができます。

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。AキーとDキーを押して隕石を回避できたら成功です。

3.6 隕石の出現位置を変更する

同じ位置に飛んでくる隕石を回避するのは簡単すぎます。そこで、毎回違う位置に隕石を出現させましょう。隕石の移動プログラムを次のように変更してください。

```
44|   // 隕石の移動
45|   teki_y += 3;
46|   if (teki_y > 600) {
47|     teki_x = rand() % 800;
48|     teki_y = 0; // 隕石を上に戻す
48|   }
```

「rand(ランド)」関数の結果は0から2147483647(約20億)までのどれかひとつの数値です。結果はrand関数を実行するたびに変わります。数値の範囲が広すぎるため、余りを求めることで範囲を狭くします。

C++言語で余りを計算するには「%(パーセント)」記号を使います。上のプログラムの場合は「800で割った余り」ですから、0から799の範囲の数値が得られます。

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。隕石が上に出現するたびに位置が変わっていたら成功です。

4 隕石を増やそう

4.1 配列で変数を増やす

出現位置がばらけた程度ではまだ簡単すぎるでしょう。そこで隕石の数を増やしましょう。同じタイプの変数を増やすには、C++の「配列」という機能を使います。

```
10 // 敵の変数
11 const int teki_kazu = 30;
12 double teki_x[teki_kazu];
13 double teki_y[teki_kazu];
14
15 // 現在の画面番号
16 //   0 = タイトル
```

配列を作るには、変数名の後ろに「[」と「]」を書き、その中に数を書きます。すると、書いた数の変数が作成されます。個々の変数を参照する場合は「**変数名[番号]**」と書きます。

配列の番号は0から始まることに注意してください。例えば**teki_x**配列の3つめの変数に1を足したい場合「**teki_x[2] += 1;**」と書きます。また、番号には変数や数式を書くことができます。

この「番号」は出席番号のようなものです。上の式は、「出席番号2番の人、右に1歩動いて下さい」と言っているわけです。

4.2 配列とfor(フォー)

配列の数が多い場合、配列の全ての変数に値を代入したり足したりするには「**for**(フォー)」を使います。

```
29     // ゲーム開始時の設定
30     for (int n = 0; n < teki_kazu; n += 1) {
30       teki_x[n] = 400;
31       teki_y[n] = 100;
31     }
32     jiki_x = 400;
33     jiki_y = 500;
```

「**for**(フォー)」は**条件によってプログラムを繰り返す命令**で、次のように書きます。

```
for (変数の作成; 条件; 変数の更新) {
  プログラム
}
```

forの実行順は少し特殊で、以下のようになっています。

変数の作成 →{ 条件判定 → プログラム → 変数の更新 }

繰り返されるのは{}の内側だけで、「変数の作成」は最初の一回しか行われません すこし難しい機能なので、とりあえず30行目のように書けば、teki_kazu変数に代入された回数だけ「**プログラム**」が繰り返されることだけ覚えてください。

敵の座標を「配列」に変えたので、敵の座標に関するプログラムは全てforを使って書き換えなくてはなりません。隕石の移動プログラムを次のように変更してください。

```
47   // 隕石の移動
48   for (int n = 0; n < teki_kazu; n += 1) {
49     teki_y[n] += 3;
50     if (teki_y[n] > 600) {
51       teki_x[n] = rand() % 800;
52       teki_y[n] = 0; // 隕石を上に戻す
53     }
54   }
55
56   // 自機の移動
57   if (get_key('A') > 0) {
```

同様に、自機と隕石の衝突を判定するプログラムもforを使って書き換えます。隕石の衝突プログラムを次のように変更して下さい。

```
61   // 自機と隕石の衝突
62   for (int n = 0; n < teki_kazu; n += 1) {
63     if (abs(teki_x - jiki_x) < 32) {
64       if (abs(teki_y - jiki_y) < 32) {
65         scene_number = 2; // ゲームオーバーに切り替える
66       }
67     }
68   }
69
70   if (get_key(' ') == 1) {
```

【課題１】

for命令を使って、以下の「画像を表示」するプログラムを配列に対応しなさい。

```
// 画像を表示
draw(teki_x, teki_y, "asteroid_a.png");
```

最初の隕石が画面下に消えた後、大量の隕石が出現したら成功です。

4.3 ビームの座標を表す変数を作成する

大量の隕石が降ってくると、どうしても避けられない場合があります。ビーム砲を発射して隕石を壊せるようにしましょう。新しい物体の表示は次の手順で行います。

1. **座標を表す変数を作成する**
2. **game関数で、座標変数の位置に画像を表示する**
3. **title関数で、ゲーム開始時の値を代入する**
4. **game関数で座標変数を制御する**

まず1の「座標を表す変数を作成する」をやりましょう。自機の変数の下に、次のプログラムを追加してください。

```
6 // 自機の変数
7 double jiki_x;
```

```
 8 double jiki_y;
 9 double jiki_beam_x;
10 double jiki_beam_y;
11
12 // 敵の変数
13 const int teki_kazu = 30;
```

《コラム》変数が使える範囲

変数は、その名前をつけた行より下でしか使えません。人間と同じように、コンピューターも、まだ読んでいない部分に書かれていることは分かりません。

4.4 game関数で座標変数の位置にビーム画像を表示する

画像を表示するには「**draw**(ドロー)」関数を使います。game関数の画像を表示するプログラムに、次のプログラムを追加してください。

```
79   // 画像を表示
80   for (int n = 0; n < teki_kazu; n += 1) {
81     draw(teki_x[n], teki_y[n], "asteroid_a.png");
82   }
83   draw(jiki_beam_x, jiki_beam_y, "shot_beam.png”);
84   draw(jiki_x, jiki_y, "space_ship.png");
85 }
```

draw関数は次のように書きます。

draw(X座標, Y座標, 画像ファイルの名前);

ゲーム画面の大きさは横が800、縦が600で、原点は左上になっています。例えば、座標(400, 300)を指定すると、**ウィンドウの中心**に画像が表示されます。

最後の「**"shot_beam.png"**」は、**表示する画像ファイルの名前**です。ファイル名なので「"」で囲んでいます。画像ファイルは「Res/画像」フォルダにあります。

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。左上にちらりと青い何かが表示されていたら成功です。

4.5 title関数で、ゲーム開始時の値を代入する

発射する前のビームを、画面外に配置して見えなくしましょう。ゲーム開始時の自機の座標を代入するプログラムの下に、次のプログラムを追加してください。

```
36     jiki_x = 400;
37     jiki_y = 500;
38     jiki_beam_x = 0;
39     jiki_beam_y = -100;
40   }
```

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。ビームの画像が見えなくなっていたら成功です。

4.6 game関数で、座標変数を制御する

最後に、スペースキーでビームを発射し、ビームを上に動かしましょう。
「スペースキーを押すとゲームオーバーに切り替える」プログラムを「スペースキーを押すとビームを発射する」に書き換えます。

スペースキーを押すとゲームオーバーに切り替えるプログラムを、次のように変更してください。

```
74      }
75    }
76
77    // ビーム発射
78    if (get_key(' ') == 1) {
79      jiki_beam_x = jiki_x;
80      jiki_beam_y = jiki_y;
81    }
82
83    // 画像を表示
84    for (int n = 0; n < teki_kazu; n += 1) {
```

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。スペースキーを押すと自機の位置にビームが表示されたら成功です。

4.7 ビームを動かす

発射したビームを動かします。ビームを発射するプログラムの下に、次のプログラムを追加してください。

```
80      jiki_beam_y = jiki_y;
81    }
82
83    // ビームの移動
84    if (jiki_beam_y > -16) {
85      jiki_beam_y -= 32;
86    }
87
83    // 画像を表示
84    for (int n = 0; n < teki_kazu; n += 1) {
```

ビームは1/60秒ごとに上に32ドット移動します。速さを表現するために、自機や隕石よりもかなり速度を上げています。

また、条件を「jiki_beam_yが-16より大きい」としているのは、ビーム画像が完全に画面外に行くまで移動させたいからです。ビームの画像の大きさは横16、縦32ドットなので、中心が画面上に16ドット移動すると完全に見えなくなります。

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。発射されたビームが上に移動したら成功です。

4.8 ビームと隕石の衝突を検出する

「ビームと隕石の衝突」は「自機と隕石の衝突」とほとんど同じで、「自機の座標」を「ビームの座標」に変えるだけです。

ビーム画像の大きさは横16、縦32ドットです。隕石は縦横64ドットなので、半径の合計は横40、縦48となります。実際に描かれている大きさを考慮して、判定に使うのは横32、縦40とします。

ビームを移動するプログラムの下に、次のプログラムを追加してください。

```
 85|     jiki_beam_y -= 32;
 86|   }
 87|
 88|   // ビームと隕石の衝突
 89|   for (int n = 0; n < teki_kazu; n += 1) {
 90|     if (abs(teki_x[n] - jiki_beam_x) < 32) {
 91|       if (abs(teki_y[n] - jiki_beam_y) < 40) {
 92|         jiki_beam_y = -100; // ビームを消す
 93|         teki_x[n] = rand() % 800;
 94|         teki_y[n] = -32; // 隕石を上に戻す
 95|       }
 96|     }
 97|   }
 98|
 99|   // 画像を表示
100|   for (int n = 0; n < teki_kazu; n += 1) {
```

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。ビームで隕石を破壊できたら成功です。

4.9 背景を表示する

見た目をかっこよくするために、背景を表示しましょう。背景を表示するには**draw**関数を使います。画像を表示するプログラムに、次のプログラムを追加してください。

```
 96|     }
 97|   }
 98|
 99|   // 画像を表示
100|   draw(400, 300, "bg_unknown_planet.png”);
101|   for (int n = 0; n < teki_kazu; n += 1) {
102|     draw(teki_x[n], teki_y[n], "asteroid_a.png");
103|   }
```

プログラムを変更したら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。背景が表示されていたら成功です。

5 音声を鳴らそう

5.1 効果音を鳴らそう

臨場感を出すために、音声を鳴らしましょう。しかし、C++言語に音声を鳴らす機能はありません。そこで、このテキストのために音声再生機能を用意しておきました。

効果音を鳴らすには「**play_sound**(プレイ・サウンド)」を使います。play_soundは、指定された音声を1回だけ再生します。

ビームを発射するプログラムに、次のプログラムを書き加えてください。

```
77|   // ビーム発射
78|   if (get_key(' ') == 1) {
79|     jiki_beam_x = jiki_x;
80|     jiki_beam_y = jiki_y;
81|     play_sound("shot_0.wav");
82|   }
```

書き加えたら「ローカルWindowsデバッガー」ボタンをクリックして実行…の前に、**音量をチェック**しておきましょう。音量は画面右下のスピーカーアイコンをクリックすると変更できます。

音量を調整して、大丈夫そうならプログラムを実行してください。スペースキーでビームを発射したとき、発射音が再生されたら成功です。

5.2 BGMを鳴らそう

BGMを鳴らすには「**play_bgm**(プレイ・ビージーエム)」を使います。play_bgmは**指定された音声を無限に繰り返し再生**します。もう一度play_bgmを実行すると、**再生されていたBGMは停止して、新しいBGMが再生されます。**

ゲーム画面に切り替えるプログラムの下に、次のプログラムを書き加えてください。

```
28|   if (get_key(' ') == 1) {
29|     scene_number = 1; // ゲーム画面に切り替える
30|
31|     // ゲーム開始時の設定
32|     play_bgm("bgm-fly-to-the-space.mp3");
33|     for (int n = 0; n < teki_kazu; n += 1) {
```

書き加えたら「ローカルWindowsデバッガー」ボタンをクリックして実行してください。ゲーム画面でBGMが再生されたら成功です。

【課題６】
play_soundを使って、隕石を壊したときに効果音を鳴らしなさい。
効果音は「Res/音声」フォルダの中にあります。

5.3 BGMを停止する

BGMを停止するには「**stop_bgm**(ストップ・ビージーエム)」を使います。

```
66  // 自機と隕石の衝突
67  for (int n = 0; n < teki_kazu; n += 1) {
68    if (abs(teki_x - jiki_x) < 32) {
69      if (abs(teki_y - jiki_y) < 32) {
70        scene_number = 2; // ゲームオーバーに切り替える
71        stop_bgm();       // BGM停止
72      }
73    }
74  }
```

6 祝🎉　ゲーム完成！

おめでとうございます！
これでシューティングゲームのテキストは終了です。お疲れ様でした。

【課題８】

あなたの好きなようにゲームを改造してください。
いくつか例を挙げておきます。
・敵や背景の画像を変更する。
・音声やBGMを変更する。
・隕石を壊したときに爆発を表示する。
・「**アイデアはあるけれど、やり方が分からない！**」
そんなときは、気軽に質問してください。

(https://github.com/tn-mai/shoot_em_up)