プログラマー勉強会　1回

•basic.h

```
1   //@basic.h
2
3   #include "DxLib.h"          //DxLib使用のため
4   #include <stdio.h>          //基本
5   #include <math.h>           //三角関数をつかうため
6
7   #ifndef _BASIC_H_           //インクルードガードのための条件コンパイル
8   #define _BASIC_H_
9
10  /**マクロ定義**/
11  #define WINDOW_WIDTH 640    //画面横サイズ
12  #define WINDOW_HEIGHT 480   //画面縦サイズ
13  #define FONT_SIZE 32        //フォントサイズ
14  #define GAME_UNDER 360      //プレイヤーが行動できる下の限界
15  #define GAME_LEFT 80        //プレイヤーが行動できる左の限界
16  #define GAME_RIGHT 400      //プレイヤーが行動できる右の限界
17
18  /**グローバル変数宣言**/
19  //円周率。arctan(1) = π/4なので、それを4倍して円周率を表現
20  const double PI = 4 * atan(1);
```

•補足

[修飾子]

const ・・・付けた変数は初期化以外で値を設定することができなくなる。定数宣言に使う。

unsigned ・・・付けた変数は符号がなくなり、正の値しか設定できない。

[条件コンパイル]

#ifdef M・・・ここ以前にMがマクロとして定義されていれば、
ここ以下をコンパイルする。

#ifndef M・・・ここ以前にMというマクロが定義されていなければ、
ここ以下をコンパイルする。

#endif・・・条件コンパイル部分の終了を示す。

•補足②

[インクルードガード]

```
7  #ifndef _BASIC_H_
8  #define _BASIC_H_
```

↑なぜ条件コンパイルを付けたのか。

同じヘッダーファイルを複数回インクルードした場合、
同じ内容のプログラムが複数回コンパイルされ、コンパイルエラーを起こしてしまう。それを防止するために、#ifndefを付け、直後に#ifndefの条件マクロを定義することによって最初の一回だけヘッダーファイルの内容をコンパイルし、エラーを防止する。これを**インクルードガード**と呼び、ヘッダーファイルにはほぼ確実に付けてある。

[一回目のインクルード]
#ifndef _BASIC_H_　←まだ定義されていないため条件を満たしコンパイルされる
#define _BASIC_H_　←条件マクロを定義
・・・

[二回目以降のインクルード]
#ifndef _BASIC_H_　←すでに定義されているため条件を満たさず、コンパイルされない
#define _BASIC_H_
・・・

•basic.h②

```
22  //よく使う色
23  const unsigned int WHITE = GetColor(255, 255, 255);
24  const unsigned int BLACK = GetColor(0, 0, 0);
25  const unsigned int LIGHT_GRAY = GetColor(195, 195, 195);
26  const unsigned int DARK_GRAY = GetColor(128, 128, 128);
27
28  /**関数宣言**/
29  bool InitSet() {
30      ChangeWindowMode(true);
31      SetMainWindowText("Study_2018_Winter");
32      SetWindowSizeChangeEnableFlag(false);
33      SetGraphMode(WINDOW_WIDTH, WINDOW_HEIGHT, 16);
34      SetWindowSizeExtendRate(1);
35      if (DxLib_Init() == -1)
36          return false;       //DxLib_Init()が-1を返したので、失敗の合図を知らせる
37
38      SetOutApplicationLogValidFlag(false);
39      SetDrawScreen(DX_SCREEN_BACK);
40      ChangeFontType(DX_FONTTYPE_ANTIALIASING_4X4);
41      SetFontSize(FONT_SIZE);
42
43      return true;            //正常終了の合図を知らせる
44  }
45
46  #endif      //条件コンパイルの終了
```

- # main.cpp

```
1   //@main.cpp
2
3   #include "basic.h"
4
5   int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
6       LPSTR lpCmdLine, int nCmdShow) {
7       if (!InitSet())
8           return -1;      //初期設定が失敗したのでエラー終了
9
10      /**変数宣言**/
11      int t0;
12
13      while (ProcessMessage() == 0 && CheckHitKey(KEY_INPUT_ESCAPE) == 0) {
14          t0 = GetNowCount();
15          ClearDrawScreen();
16
17
18          while ((GetNowCount() - t0) <= (1000 / 60));
19          ScreenFlip();
20      }
21
22      DxLib_End();
23
24      return 0;
25  }
```

・オブジェクト

[多数のデータの取り扱い]
例(プレイヤーに必要なデータ)

- **座標**(x,y) → int point_X,point_Y;
- **速度**(x,y) → int vellocity_X,vellocity_Y;
- **長さ**(x,y) → int length_X,length_Y;
- **角度** → double angle;
- **ライフ** → int life;

Etc…

通常の変数には一つの値しか保存できないため、変数を多数用意する必要があった。
↓
バラバラでわかりずらい。

• オブジェクト②

[構造体]
構造体の登場で、一つの変数に複数の値を保存できるようになり、必要なデータを一か所にまとめることができるため、多少はプログラムしやすくなった。

```
struct PLAYER{
    int point_X;
    int point_Y;
    int length_X;
    …
};

PLAYER player;
player.point_X += player.vellocity_X;
```

・オブジェクト③

[オブジェクト]
構造体をさらに拡張させ、メンバに関数を加えることで値を保存だけで
なく、計算処理などもできるようになった変数(みたいなやつ)。

[変数による値の保持]
・座標(x,y)
・速度(x,y)
・長さ(x,y)
・角度
・ライフ
[関数による計算処理]
・当たり判定(new)
・移動(new)
・描画(new)
などができるようになった。

・オブジェクト③

難しい感じで言ったけど、大雑把に言うと、要は構造体に関数が加わったくらい。

※イメージ

```
struct PLAYER{
    int point_X;
    int point_Y;
    int vellocity_X;
    …

     void Move(){
       point.X += vellocity.X
    }
};
```

・クラス

クラス・・・オブジェクトの定義をする場所。オブジェクト内で使う変数や関数を宣言したり定義する。

メンバ変数・・・オブジェクト内で使う変数。フィールドとも言う。
メンバ関数・・・オブジェクト内で使う関数。メソッドとも呼ばれる。

[クラス定義テンプレ]

```
class クラス名{
        ・・・
};
```

※クラス名のルールは変数名や関数名と同じ

• クラスの定義

プロジェクト　＞　新しい項目の追加

↓

ヘッダーファイルを選択し、「class_def.h」と名前を付けて追加ボタン

```
1   //@class_def.h
2
3   #include "basic.h"
4
5   #ifndef _CLASS_DEF_H_          //インクルードガード
6   #define _CLASS_DEF_H_
7
8   class PLAYER {                 //PLAYERクラスの定義
9
10  };
11
12  #endif
```

• 便利になるかもしれない

プロジェクト　＞　既存の項目の追加
↓
配布した「SecondVector.h」を選択して追加ボタン
↓
basic.hに移動
↓
#include “SecondVector.h”　を記述
↓
SecondVector.hをプロジェクトのディレクトリに入れる
↓
SDV型(2次元ベクトル型)がつかえるようになる。

• SDV型

SDV型・・・2次元ベクトル型。X成分とY成分の情報を格納できる。

[できること]
SDV A,B;
//一斉代入
A.Set(3,5);
B.Set(-8,2);
//足し算引き算
A + B = (-5,7)
A – B = (11,3)
//足し算代入引き算代入
A += B →　A = (-5,7);
A ー= B → A = (11,3);

あとは*で内積、/で外積とかが出るんだって

• クラスの定義

```
1   //@class_def.h
2
3   #include "basic.h"
4
5   #ifndef _CLASS_DEF_H_          //インクルードガード
6   #define _CLASS_DEF_H_
7
8   class PLAYER {
9   private:
10      //メンバ変数
11      SDV point;        //座標
12      SDV vellocity;    //速度(フレーム毎にpointに加えていく値)
13      SDV length;       //長さ(グラフィックの大きさの半分)
14      double angle;     //角度
15
16  public:
17      //コンストラクタ(後で説明)
18      PLAYER() {
19          point.Set(100, GAME_UNDER / 2);
20          vellocity.Set(0, 0);
21          length.Set(20, 48);
22          angle = 0;
23      }
```

・クラスの定義

```
25      void Move() {
26          if (CheckHitKey(KEY_INPUT_LEFT))
27              vellocity.X = -6;
28              //←キーが押されたので速度のX成分を-6にして左に動かす
29          else if (CheckHitKey(KEY_INPUT_RIGHT))
30              vellocity.X = 6;
31              //→キーが押されたので速度のX成分を6にして右に動かす
32          else
33              vellocity.X = 0;
34              //どちらも押されていないので速度のX成分を0にしてX方向には動かさない
35
36          if (CheckHitKey(KEY_INPUT_UP))
37              vellocity.Y = -6;
38              //↑キーが押されたので速度のY成分を-6にして上に動かす
39          else if (CheckHitKey(KEY_INPUT_DOWN))
40              vellocity.Y = 6;
41              //↓キーが押されたので速度のY成分を6にして下に動かす
42          else
43              vellocity.Y = 0;
44              //どちらも押されていないので速度のY成分を0にしてY方向には動かさない
45
46          point += vellocity;     //速度を座標に加えて動かす
```

• クラスの定義

```
48              /**移動範囲外に行ってしまった時の処理**/
49              if (point.Y - length.Y < 0)
50                  point.Y = length.Y;
51              if (point.Y + length.Y > GAME_UNDER)
52                  point.Y = GAME_UNDER - length.Y;
53              if (point.X < GAME_LEFT)
54                  point.X = GAME_LEFT;
55              if (point.X > GAME_RIGHT)
56                  point.X = GAME_RIGHT;
57          }
58      };
59
60      #endif
```

・コンストラクタ

↓これ

```
PLAYER() {
    point.Set(100, GAME_UNDER / 2);        //初期座標を設定
    vellocity.Set(0, 0);                   //初期速度設定
    length.Set(20, 48);                    //長さの設定
    angle = 0;                             //初期角度の設定
}
```

コンストラクタ・・・オブジェクトを宣言し生成するときに、
最初に呼び出される初期化用関数。主にメンバ変数の初期化の処理を行う。
クラス名と同じで、型は持たない。

・オブジェクトの生成

```
1   //@main.cpp
2
3   #include "basic.h"
4   #include "class_def.h"      //クラスの使用のため
5
6   int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
7       LPSTR lpCmdLine, int nCmdShow) {
8       if (!InitSet())
9           return -1;      //初期設定が失敗したのでエラー終了
10
11      /**変数宣言**/
12      int t0;
13
14      //オブジェクト
15      PLAYER *player = new PLAYER();      //playerオブジェクトの生成
16
```

[オブジェクト生成テンプレ]

オブジェクト型名 *オブジェクト変数名 = new コンストラクタ;

• new演算子

new ・・・　メモリ上から指定されたオブジェクト分の領域を確保する演算子。C++だと確保した領域のアドレスを出すため、格納する変数はポインタ型にしなければならない。

←newを使わないとJavaだとこう言われる。

C++では、newを使わなくてもオブジェクトの生成はできるが、今回は基本的にnewを使って領域を確保し、オブジェクトを生成することにする

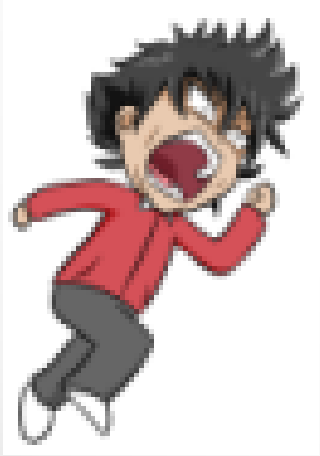
• 描画メンバ関数①

[DrawRectRotaGraph]

グラフィックを指定した部分だけ切り抜いて、回転系描画ができる。公式リファレンスに載っていない隠し関数の一つ。

DrawRectRotaGraph(X座標, Y座標, 切抜の左上X座標,切抜の左上Y座標, 横幅, 縦幅, 拡大率, 角度, グラフィックハンドル, 透過フラグ, 反転フラグ);

[右に動いてるとき]



[左に動いてるとき]

[止まってるとき]

・描画メンバ関数②

```
24      void Move() { ... }
57
58      //グラフィックハンドルは引数で取り込む
59      void Draw(int graph) {
60          int sizeX, sizeY;          //画像のサイズを保存する
61          const int line = 64;       //グラフィックの左側と右側の境界
62
63          GetGraphSize(graph, &sizeX, &sizeY);     //画像サイズの取得
64
65          if (vellocity.X > 0)
66              DrawRectRotaGraphF(point.X, point.Y, 0, 0, line, sizeY,
67                  1.0, angle, graph, TRUE, FALSE);
68              //右に動いているので画像の左側を描画
69          else if (vellocity.X < 0)
70              DrawRectRotaGraphF(point.X, point.Y, 0, 0, line, sizeY,
71                  1.0, angle, graph, TRUE, TRUE);
72              //左に動いているので画像の左側を描画し、反転
73          else
74              DrawRectRotaGraphF(point.X, point.Y, line, 0, sizeX - line, sizeY,
75                  1.0, angle, graph, TRUE, FALSE);
76              //止まっているので画像の右側を描画
77      }
78  };
79
80  #endif
```

• メンバ関数を呼び出す

```
14      //オブジェクト
15      PLAYER *player = new PLAYER();      //playerオブジェクトの生成
16
17      //グラフィックハンドル
18      int player_graph = LoadGraph("DATA/graph/player.png");
19
20      while (ProcessMessage() == 0 && CheckHitKey(KEY_INPUT_ESCAPE) == 0) {
21          t0 = GetNowCount();
22          ClearDrawScreen();
23
24          /*構造体でメンバ変数を呼び出すときと同様*/
25          player->Move();                 //Move()を呼び出して、動作の処理
26          player->Draw(player_graph);     //Draw(int graph)を呼び出して、描画の処理
27          /*↑ポインタ型なのでアロー演算子*/
28
29          while ((GetNowCount() - t0) <= (1000 / 60));
30          ScreenFlip();
31      }
32
33      DxLib_End();
34
35      return 0;
36  }
```

構造体でメンバ変数を呼ぶときと同様に、メンバ演算子「.」、
オブジェクト変数がポインタ型の時はアロー演算子「->」
を使って呼び出す。

- privateとpublic

```
class PLAYER{
private:
 ・・・          ⇕(privateメンバ変数or関数)
public:
 ・・・          ⇕(publicメンバ変数or関数)
};
```

public・・・オブジェト変数があればどこでも呼び出せる変数もしくは関数

private・・・クラス内でしか呼び出せない変数もしくは関数

例

```
class PLAYER{
private:
      int X;
…
```

➡

```
@main関数
PLAYER p;
p.X;
```

➡

• NPCの生成

一応RPGなのでNPCのためのクラスを作り、
PLAYERと同様に動作させる。

```
1   //@class_def.h
2
3   #include "basic.h"
4
5   #ifndef _CLASS_DEF_H_        //インクルードガード
6   #define _CLASS_DEF_H_
7
8   class PLAYER { ... };
79
80  class NPC {
81  private:
82      //メンバ変数
83      SDV point;       //座標
84      SDV vellocity;   //速度
85      SDV length;      //長さ
86      double angle;    //角度
87
88  public:
89      //コンストラクタ
90      NPC() {
91          point.Set(100, GAME_UNDER / 4);
92          vellocity.Set(0, 0);
93          length.Set(20, 48);
94          angle = 0;
95      }
```

使う変数PLAYERと同じじゃん

• クラスの継承

PLAYERとNPCでは使う変数と関数が同じものが多い。
それらを一々書いていては非効率なので、
被っている変数や関数を使うために継承を行う。

継承・・・既に定義されているクラスの変数や関数を
再利用し、プログラミングの効率を高めること。

[継承テンプレ]

```
class クラス名 : public 継承元クラス名{
・・・
};
```

- NPCクラス

```
1   //@class_def.h
2
3   #include "basic.h"
4
5   #ifndef _CLASS_DEF_H_          //インクルードガード
6   #define _CLASS_DEF_H_
7
8   class PLAYER { ... };
79
80  class NPC : public PLAYER {
81  private:
82      int number;        //NPCの番号
83      bool move;         //動いてるかどうか
84
85  public:
86      //コンストラクタ
87      NPC() {
88          point.Set(100, GAME_UNDER / 2);
89          vellocity.Set(0, 0);
90          length.Set(20, 48);
91          angle = 0;
92          move = false;
93      }
```

• NPCクラス②

```
95      //コンストラクタ2
96      NPC(int n) {
97          //引数で取り込んだ数字によって初期配置を変える
98          if (n == 1)
99              point.Set(80, GAME_UNDER / 4);
100         else if (n == 2)
101             point.Set(80, 3 * GAME_UNDER / 4);
102
103         //番号の登録
104         number = n;
105         vellocity.Set(0, 0);
106         length.Set(20, 48);
107         angle = 0;
108         move = false;
109     }
```

• コンストラクタが二つあるが

引数の型や数が異なれば、コンストラクタはいくつでも定義することができる

```
NPC() {
    point.Set(100, GAME_UNDER / 2);
    vellocity.Set(0, 0);
    length.Set(20, 48);
    angle = 0;
    move = false;
}
```

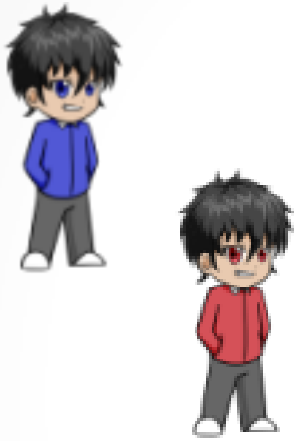
```
NPC(int n) {
    //引数で取り込んだ数字によって初期配置を変える
    if (n == 1)
        point.Set(80, GAME_UNDER / 4);
    else if (n == 2)
        point.Set(80, 3 * GAME_UNDER / 4);

    //番号の登録
    number = n;
    vellocity.Set(0, 0);
    length.Set(20, 48);
    angle = 0;
    move = false;
}
```

また、コンストラクタに限らず関数も同様に、引数の型や個数が異なったり関数の型が異なれば、同じ名前の関数をいくつでも定義できる。
このように同じ名前の関数を複数定義することを**多重定義**や**オーバーロード**と呼ぶ。

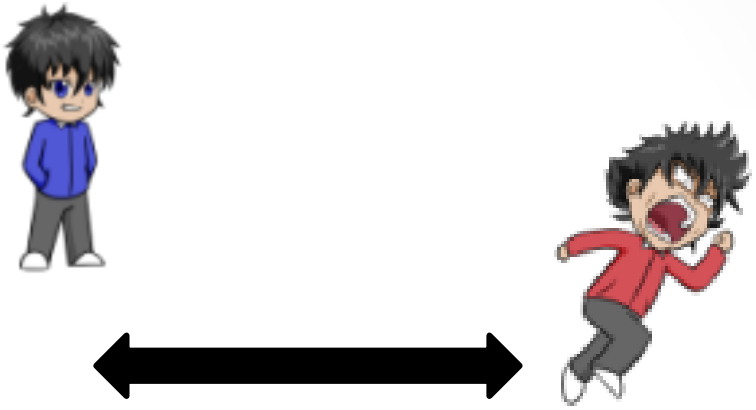
・NPCの動き方

・右に動くとき

①



②



Playerが一定の距離離れる(今回は160pix)

③　Playerを追いかける形で
NPCも動かす

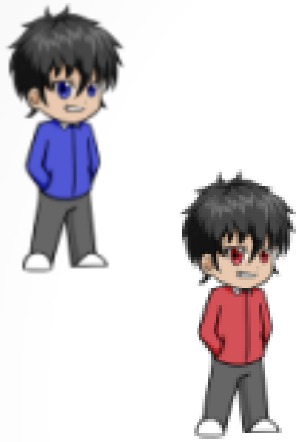
④

Playerが止まっている状態で
NPCが一定の距離(60pix)
近づいたらNPCも止める。

・NPCの動き方②

・右に動くとき

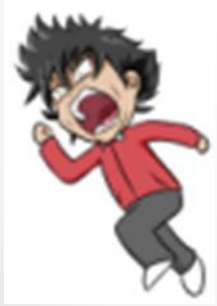
①



②

Playerが一定の距離離れる(今回は120pix)

③　Playerを追いかける形で
NPCも動かす



④

Playerが止まっている状態で
NPCがplayerの後ろ(60pix)に
ついたらNPCも止める。

• NPCクラス③

```
111     void Move(SDV player_point) {
112         //プレイヤーがNPCより160離れたらNPCも動く
113         if (player_point.X > point.X + 160)
114             vellocity.X = 6;
115         //NPCがプレイヤーに60より近づいたら止める
116         if (vellocity.X > 0 && point.X > player_point.X - 60)
117             vellocity.X = 0;
118         //上述の逆版
119         if (player_point.X < point.X - 120)
120             vellocity.X = -6;
121         if (vellocity.X < 0 && point.X < player_point.X - 60)
122             vellocity.X = 0;
123
124         //動いているかのフラグ建て
125         if (vellocity.X > 0 || (player_point.X >= GAME_RIGHT
126             && CheckHitKey(KEY_INPUT_RIGHT) == 1))
127             move = true;
128         else
129             move = false;
130
131         point += vellocity;
```

• NPCクラス④

```
132             /*移動範囲外の処理*/
133             if (point.Y - length.Y < 0)
134                 point.Y = length.Y;
135             if (point.Y + length.Y > GAME_UNDER)
136                 point.Y = GAME_UNDER - length.Y;
137             if (point.X - length.X < 0)
138                 point.X = length.X;
139             if (point.X + length.X > GAME_RIGHT)
140                 point.X = GAME_RIGHT - length.X;
141         }
```

• NPCクラス⑤

```
149     void Draw(int graph) {
150         int sizeX, sizeY;        //画像のサイズを保存する
151         const int line = 64;     //グラフィックの左側と右側の境界
152 
153         GetGraphSize(graph, &sizeX, &sizeY);     //画像サイズの取得
154 
155         if (vellocity.X < 0)
156             DrawRectRotaGraphF(point.X, point.Y, 0, 0, line, sizeY, 1.0,
157                 angle, graph, TRUE, TRUE);
158         //左に移動するので左側を切抜いて反転描画
159         else if (move)
160             DrawRectRotaGraphF(point.X, point.Y, 0, 0, line, sizeY, 1.0,
161                 angle, graph, TRUE, FALSE);
162         //移動フラグが建っているので左側を切抜いて描画
163         else
164             DrawRectRotaGraphF(point.X, point.Y, 64, 0, sizeX - line, sizeY, 1.0,
165                 angle, graph, TRUE, FALSE);
166         //止まっているので右側を切抜いて描画
167     }
168 
169 };
170 
171 #endif
```

• protected

PLAYERクラスを継承したはずなのに、変数にエラーが出る
↓何故？
privateの変数、関数は継承できない。
↓どうする？
PLAYERクラスのprivateをprotectedに変更する。

protected・・・継承されるようになったprivate変数、関数

```
 8  class PLAYER {
 9  private:
10      //メンバ変数
11      SDV point;
12      SDV vellocity;
13      SDV length;
14      double angle;
```

```
 8  class PLAYER {
 9  protected:
10      //メンバ変数
11      SDV point;
12      SDV vellocity;
13      SDV length;
14      double angle;
```

• プレイヤーの座標取得

NPCのMove関数の引数にplayer_point(プレイヤーの座標)があるが、Protectedに置かれているため、取得できない。

↓

座標の値だけ返すメンバ関数をPLAYERクラスに実装

```
 8  class PLAYER {
 9  protected:
10      //メンバ変数
11      SDV point;      //座標
12      SDV vellocity;  //速度(フレーム毎にpointに加えていく値)
13      SDV length;     //長さ(グラフィックの大きさの半分)
14      double angle;   //角度
15
16  public:
17      PLAYER() { ... }
23
24      //座標取得
25      SDV getPoint() {
26          return point;
27      }
28
29      void Move() { ... }
62
63      //グラフィックハンドルは引数で取り込む
64      void Draw(int graph) { ... }
83  };
```

これで、座標の取得が可能

- pointをpublicに置けばええやん

• NPCの生成(出直し)

今回NPCの数は2人で固定

```
1   //@basic.h
2
3   #include ...
7
8   #ifndef _BASIC_H_            //インクルードガードのための条件コンパイル
9   #define _BASIC_H_
10
11  /**マクロ定義**/
12  #define WINDOW_WIDTH 640     //画面横サイズ
13  #define WINDOW_HEIGHT 480    //画面縦サイズ
14  #define FONT_SIZE 32         //フォントサイズ
15  #define GAME_UNDER 360       //プレイヤーが行動できる下の限界
16  #define GAME_LEFT 80         //プレイヤーが行動できる左の限界
17  #define GAME_RIGHT 400       //プレイヤーが行動できる右の限界
18
19  #define NPC_NUM 2            //NPCの人数
```

• NPCの生成②

```
1   //@main.cpp
2
3   #include ...          //クラスの使用のため
5
6   int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
7       LPSTR lpCmdLine, int nCmdShow) {
8       if (!InitSet())
9           return -1;          //初期設定が失敗したのでエラー終了
10
11      /**変数宣言**/
12      int t0;
13
14      //オブジェクト
15      PLAYER *player = new PLAYER();          //playerオブジェクトの生成
16      NPC **npc = new NPC*[NPC_NUM];          //npcオブジェクトの配列宣言
17      for (int i = 0; i < NPC_NUM; i++)
18          npc[i] = new NPC(i + 1);            //npcオブジェクトの生成
```

- **newを使ったオブジェクト配列宣言**

  型名 **配列名 = new 型名[要素数];

  ↑ポインタ型の配列が欲しいのでポインタのポインタ型

• NPCの実装

```
19      //@main.cpp
20      //グラフィックハンドル
21      int player_graph = LoadGraph("DATA/graph/player.png");
22      int npc_graph = LoadGraph("DATA/graph/npc.png");
23
24      while (ProcessMessage() == 0 && CheckHitKey(KEY_INPUT_ESCAPE) == 0) {
25          t0 = GetNowCount();
26          ClearDrawScreen();
27
28          player->Move();                          //Move()を呼び出して、動作の処理
29          for (int i = 0; i < NPC_NUM; i++)
30              npc[i]->Move(player->getPoint());    //Move(SDV)を呼び出して、動作の処理
31
32          player->Draw(player_graph);              //Draw(int)を呼び出して、描画の処理
33          for (int i = 0; i < NPC_NUM; i++)
34              npc[i]->Draw(npc_graph);             //Draw(int)を呼び出して、描画の処理
35
36          while ((GetNowCount() - t0) <= (1000 / 60));
37          ScreenFlip();
38      }
39
40      DxLib_End();
41
42      return 0;
43  }
```

- 第一回終了