ReStudyC++_9

コンポーネント指向プログラミングの基礎

コンポーネントを学ぶために、これからコンソールで遊べるシューティングゲームを作る。

```
/*
C++基本の総復習オマケ1（コンポーネント指向プログラミングの基礎）
*/

//【コンポーネント指向】
//unityとかanrealでは強制的にやらされてるやつ
//ざっくりどんな感じかと言うと、
//どんな機能も追加出来る実体（GameObject的なの）をベースに
//それに機能（コンポーネントorアセット）をどんどん追加して本当に欲しい実体を作成する
//・利点
//機能の流用が簡単（Aには使えてBには使えない、というのが基本的にない）
//汎用性が高いので、効率がよく、別の開発者が後から改造するという事がしやすい
//大規模開発に向いており、現在のゲーム開発に向く

//【今回のコード　ー　非コンポーネント指向（殴り書き）】
//シューティングゲームを通してコンポーネント指向を解説する
//1.殴り書き（とにかく機能を満たしたもの）
//2.オブジェクト指向
//3.コンポーネント指向
//の順にコードを書くことで、コンポーネント指向を実感する。

//【参考サイト】
//https://qiita.com/harayuu10/items/bf6d73353efa45212200

/*
* 以下、なぐり書きのコード
*/



//ヘッダファイルのインクルード
#include "Component.h"//コンソールに分かりやすく表示するために作成
#include <list>
#include <Windows.h>

//画面出力用のバファッファイルの作成
```

```
ScreenBuffer g_ScreenBuffer;
//静的メンバ変数の初期化
char InputData::Buffer = 0;

//エネミークラスの作成
//自身のインスタンスを作成する機能と、作成されたインスタンスを管理する機能を持つ(listの部分)
class Enemy
{
public:
	//静的メンバ変数の作成(全てのインスタンス共通の変数)
	static std::list<Enemy*> List;
	//メンバ変数(全てのインスタンスが個別に持つ変数)
	int x, y;
	//コンストラクタ
	//静的メンバ変数のリストに作成したインスタンスを入れる
	Enemy()
	{
		List.push_back(this);
	}
	//Update(継承先で実装-もしくは何もしない)
	void Update() {}
	//"ReStudyC++_9.h"で作成したバッファ上にエネミーを表示
	void Draw()
	{
		g_ScreenBuffer.buffer2[x][y] = 'E';
	}
};

//静的メンバ変数の初期化(このように外に宣言するだけでも処理化処理はされる)
std::list<Enemy*> Enemy::List;

//弾丸のクラス
class Bullet
{
public:
	int x, y;
	void Update()
	{
		x--;//xが縦(下に行くほど増加)
		auto buff = Enemy::List;
		for (auto e : buff)
		{
```

```
                        //弾丸の位置とエネミーの位置が一致した時に、エネミーを削除
                        if (e->x == x && e->y == y)
                                Enemy::List.remove(e);//removeはリストのメソッド
                }
        }
        //自分のいる位置にbを描画する
        void Draw()
        {
                g_ScreenBuffer.buffer2[x][y] = 'b';
        }
};

//自機のクラス
class Player
{
        int x, y;
        //球のリスト
        std::list<Bullet*> BulletList;
public:

        Player()
        {
                x = 5; y = 8;
        }

        void Draw()
        {
                g_ScreenBuffer.buffer2[x][y] = 'p';

                //球の描画を呼ぶ
                for (auto b : BulletList)
                        b->Draw();
        }

        void Update()
        {
                //移動（押した方向にずっと進む）
                if (InputData::KeyCheck('d') && y < SCREE_NLENGTH - 1)
                        y++;
                if (InputData::KeyCheck('a') && y > 0)
                        y--;
                if (InputData::KeyCheck('s') && x < SCREE_NLENGTH - 1)
```

```
                        x++;
                if (InputData::KeyCheck('w') && x > 0)
                        x--;

                //球のupdateを呼ぶ
                for (auto b : BulletList)
                        b->Update();

                //球発射（他の操作をしない限りは打ちっぱなし）
                if (InputData::KeyCheck(' '))
                {
                        BulletList.push_back(new Bullet());
                        BulletList.back()->y = y;
                        BulletList.back()->x = x - 1;
                }

                //画面から消えた球を消す
                //イテレーション回してるリストの中身変えるとおかしくなるので
                std::list<Bullet*> buff = BulletList;
                for (auto b : buff)
                {
                        if (b->x < 0)
                                BulletList.remove(b);
                }
        }
};



int main()
{
        Player player;
        Enemy enemy[10];
        //横並びに１０機作成
        for (int i = 0; i < 10; i++)
        {
                enemy[i].x = 1;
                enemy[i].y = i;
        }
        //pを押されない限りは実行される
        while (!InputData::KeyCheck('p'))
        {
```

```
                //画面の初期化
                //system() 関数はシステムにコマンド実行を要求する関数
                //システムに cls コマンド実行を要求する(winのみ、コンソール画面のクリア)
                system("cls");
                //作成したバッファのクリア
                g_ScreenBuffer.Clear();
                //入力処理の受け取り
                InputData::Update();

                //実際の処理
                //Update(それぞれのデータの更新)
                player.Update();
                for (auto e : Enemy::List)
                        e->Update();
                //Draw（それぞれの描画）
                player.Draw();
                for (auto e : Enemy::List)
                        e->Draw();

                //Buffer表示（画面に表示）
                printf("%s", g_ScreenBuffer.buffer);
                Sleep(100);//100ms待機（バグるから）
        }
        return 0;
}

/*【上記コードの問題点】
* それぞれのクラス間の依存関係が非常に高く、
* 何か１つ変更するたびに、全部イジらねばならなくなります。
* 下手したら、全体の設計からやり直すことになります。
*/
```

Component.h

```
#pragma once
/*
コンソールに分かりやすく表示するためのソースコードです。
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#define SCREE_NLENGTH 20
```

```
//コンソールに出力するためのバッファを作成するクラス
//バッファ:蓄えておく場所
//コンソールに表示する領域を決め、update毎に新しい画面を作っている。
class ScreenBuffer
{
public:
	//【メンバ変数】
	//SCREE_NLENGTHで画面のサイズを決める
	union {
		//画面表示用のバッファ
		char buffer[SCREE_NLENGTH * (SCREE_NLENGTH + 1) + 1];
		char buffer2[SCREE_NLENGTH][SCREE_NLENGTH + 1];
	};

	//【メンバ関数】
	//コンストラクタ
	ScreenBuffer()
	{
		Clear();
	}
	//Clear()メソッドの作成
	void Clear()
	{
		//Bufferをスペースでクリア
		for (int i = 0; i < SCREE_NLENGTH * (SCREE_NLENGTH + 1); i++)
		{
			buffer[i] = ' ';
		}
		//行の終わりに改行入れる
		for (int i = 0; i < SCREE_NLENGTH; i++)
			buffer2[i][SCREE_NLENGTH] = '¥n';

		//全ての終わりにNULL文字
		buffer[SCREE_NLENGTH * (SCREE_NLENGTH + 1)] = '¥0';
	}
};

//上記で作成したバッファに入力する処理のクラス
class InputData
{
	//一文字分のバッファ
	static char Buffer;
```

```
public:
	//バッファのUpdate
	static void Update()
	{
		//kbhit関数とは、何かキーが押された場合0以外の値を返し、何もキーが押されていない場合は0を
返す関数
		//(2005年に、_kbhit関数に変更されました(VC++専用))
		if (_kbhit()) {
			//文字をエコーせずにコンソールから 1 文字を読み取る。
			//エコー：入力された内容をコンソールに表示すること
			Buffer = _getch();
		}
	}
	//入力Keyのチェック
	static bool KeyCheck(char key)
	{
		if (Buffer == key)
			return true;
		return false;
	}
};

//【extern】
//複数ソースコードをまたいで変数やメソッドを使用するときに時に使います。
//全ファイル中のどれかに定義されている宣言だけを行い定義は行わない宣言方法です。
//つまり、複数ファイルに跨ってグローバル変数を共有する際に、使う。
extern ScreenBuffer g_ScreenBuffer;
```

上記コードとそのコメントを参考に prob9_10_11_12 を解け。

（１２までやらないとたぶん無理）