ReStudyC++_10

コンポーネント指向プログラミングの基礎

ポリモーフィズムを活用し、MainLoop をスッキリさせる

```
/*
C++基本の総復習オマケ2（コンポーネント指向プログラミングの基礎）
*/

//【今回のコード　ー　オブジェクト指向】
//ポリモーフィズムを活用し、MainLoopをスッキリさせる。


#include "Component.h"
#include <list>
#include <Windows.h>

ScreenBuffer g_ScreenBuffer;
char InputData::Buffer = 0;

//Unityでいうgameobject的な物を作りたい
class Object
{
public:
	Object() {}
	virtual ~Object() {}
	int x, y;
	virtual void Update() {}
	virtual void Draw() {}
};

//オブジェクトのリストを定義
std::list<Object*> g_ObjectList;//bufでも弄りたいのでポインタ

class Enemy : public Object
{
public:
	void Update() {}
	void Draw()
	{
```

```
                g_ScreenBuffer.buffer2[x][y] = 'E';
        }
};

class Bullet : public Object
{
public:
        void Update()
        {
                x--;
                //画面外に消えてたら自分を消す
                if (x < 0)
                {
                        g_ObjectList.remove(this);
                        delete this;
                        return;
                }

                //ObjectListからEnemyを検索して当たり判定を行い削除もする。
                auto buff = g_ObjectList;
                for (auto obj : buff)
                {
                        //objがEnemyの場合キャスト出来る。違うと失敗してnullptrが入る
                        if (dynamic_cast<Enemy*>(obj) == nullptr)
                                continue;//位置ループだけ飛ばす（つまり、下のif文を飛ばす）
                        if (obj->x == x && obj->y == y) {
                                g_ObjectList.remove(obj);
                                delete obj;
                        }
                }
        }
        void Draw()
        {
                g_ScreenBuffer.buffer2[x][y] = 'b';
        }
};

class Player : public Object
{
public:
        Player()
        {
```

```
                x = 5; y = 8;
        }
        void Draw()
        {
                g_ScreenBuffer.buffer2[x][y] = 'p';
        }

        void Update()
        {
                //移動
                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--;

                //球発射
                if (InputData::KeyCheck(' '))
                {
                        g_ObjectList.push_back(new Bullet());
                        g_ObjectList.back()->y = y;
                        g_ObjectList.back()->x = x - 1;
                }
        }
};

int main()
{
        //追加
        g_ObjectList.push_back(new Player());
        for (int i = 0; i < 10; i++)
        {
                g_ObjectList.push_back(new Enemy());
                g_ObjectList.back()->x = 1;
                g_ObjectList.back()->y = i;
        }

        while (!InputData::KeyCheck('p'))
        {
```

```
		//画面の初期化
		system("cls");
		g_ScreenBuffer.Clear();
		InputData::Update();

		//実際の処理
		//Update中にObjectListがいじられてイテレーションバグるのを回避
		auto buff = g_ObjectList;
		for (auto obj : buff)
			obj->Update();
		for (auto obj : g_ObjectList)
			obj->Draw();

		//Buffer表示
		printf("%s", g_ScreenBuffer.buffer);
		Sleep(100);
	}

	//追加
	for (auto obj : g_ObjectList)
		delete obj;
	g_ObjectList.clear();

	return 0;
}

/*【上記コードの利点と問題点】
* MainLoopがすっきりする。消失や当たり判定をオブジェクト自身で行える。
* ぶっちゃけ設計が完璧に決まっているなら、オブジェクト指向の方が簡単だし、早い。
* 現実には設計とはどんどん変えていかねばならないので、このままだと変更時のコストが高い。
* 銀行とかならともかく、ゲーム開発のような仕様が二転三転するような現場では向かない。
*/
```

ここを理解しないと次も分からない。