ReStudyC++_11

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にし、頻繁な仕様変更に耐えられるようにする

```
/*
C++基本の総復習オマケ3（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　コンポーネント指向】
//コンポーネント指向にし、頻繁な仕様変更に耐えられるようにする
//オブジェクトにコンポーネントのリストを持たせてコンポーネントを継承して全てのものを定義

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

ScreenBuffer g_ScreenBuffer;
char InputData::Buffer = 0;

class Object;

class Component
{
protected:
public:
	Component() {}
	virtual ~Component() {}
	Object* Parent;
	virtual void Start() {}
	virtual void Update() {}
	virtual void Draw() {}
};

class Object
{
public:
	Object() {}
	~Object() {
		for (auto com : ComponentList)
```

```
                        delete com;
        }

        std::list<Component*> ComponentList;
        void Update()
        {
                auto buff = ComponentList;
                for (auto com : buff)
                        com->Update();
        }
        void Draw()
        {
                for (auto com : ComponentList)
                        com->Draw();
        }

        //オブジェクトが持っているコンポーネントを取得
        template<class T>
        T* GetComponent()
        {
                for (auto com : ComponentList) {
                        T* buff = dynamic_cast<T*>(com);
                        if (buff != nullptr)
                                return buff;
                }
                return nullptr;
        }

        //オブジェクトが持っているコンポーネントを追加
        template<class T>
        T* AddComponent()
        {
                T* buff = new T();
                buff->Parent = this;
                ComponentList.push_back(buff);
                buff->Start();
                return buff;
        }
};

//オブジェクトのリストを定義
std::list<Object*> g_ObjectList;
```

```
//場所を示すコンポーネント
class Position : public Component
{
public:
	int x, y;
};

//敵コンポーネント
class Enemy : public Component
{
	Position* pos = nullptr;
public:
	void Draw()
	{
		if (pos == nullptr)
			pos = Parent->GetComponent<Position>();
		g_ScreenBuffer.buffer2[pos->x][pos->y] = 'E';
	}
};

//弾コンポーネント
class Bullet : public Component
{
	Position* pos = nullptr;
public:
	void Update()
	{
		if (pos == nullptr)
			pos = Parent->GetComponent<Position>();
		pos->x--;
		//画面外に消えてたら自分を消す
		if (pos->x < 0)
		{
			Parent->ComponentList.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)
```

```
                {
                        //全体からEnemyを探す
                        if (obj->GetComponent<Enemy>() == nullptr)
                                continue;
                        if (obj->GetComponent<Position>()->x == pos->x && obj->GetComponent<Position>()-
>y == pos->y) {
                                g_ObjectList.remove(obj);
                                delete obj;
                        }
                }
        }
        void Draw()
        {
                if (pos == nullptr)
                        pos = Parent->GetComponent<Position>();
                g_ScreenBuffer.buffer2[pos->x][pos->y] = 'b';
        }
};

class Player : public Component
{
        Position* pos = nullptr;
public:
        void Start()
        {
                if (pos == nullptr)
                        pos = Parent->GetComponent<Position>();
                pos->x = 5; pos->y = 8;
        }
        void Draw()
        {
                if (pos == nullptr)
                        pos = Parent->GetComponent<Position>();
                g_ScreenBuffer.buffer2[pos->x][pos->y] = 'p';
        }

        void Update()
        {
                if (pos == nullptr)
                        pos = Parent->GetComponent<Position>();
                //移動
                if (InputData::KeyCheck('d') && pos->y < SCREE_NLENGTH - 1)
```

```
                pos->y++;
        if (InputData::KeyCheck('a') && pos->y > 0)
                pos->y--;
        if (InputData::KeyCheck('s') && pos->x < SCREE_NLENGTH - 1)
                pos->x++;
        if (InputData::KeyCheck('w') && pos->x > 0)
                pos->x--;

        //球発射
        if (InputData::KeyCheck(' '))
        {
                Object* obj = new Object;
                Position* posb = obj->AddComponent<Position>();
                obj->AddComponent<Bullet>();
                posb->y = pos->y;
                posb->x = pos->x - 1;
                g_ObjectList.push_back(obj);
        }
    }
};

int main()
{
    //追加
    Object* obj = new Object;
    obj->AddComponent<Position>();
    obj->AddComponent<Player>();
    g_ObjectList.push_back(obj);

    //敵の作成
    for (int i = 0; i < 10; i++)
    {
        obj = new Object;
        Position* pos = obj->AddComponent<Position>();
        pos->x = 1;
        pos->y = i;
        obj->AddComponent<Enemy>();
        g_ObjectList.push_back(obj);
    }

    //ゲームループ処理
    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;
}
/*
* オブジェクトにコンポーネントのリストを持たせてコンポーネントを継承して全てのものを定義
* このようにすることで全てのコンポーネントから別のコンポーネントへのアクセス手段が出来ます(特にpositionが顕著)
* シーンにたくさんのObjectがありそのObjectがたくさんのコンポーネントを持っているという構造。
* コンポーネントをコンポーネント名を指定して取得するためにGetComponentはテンプレートを使用。
* Objectクラスの中にコンポーネントの取得。追加が組み込まれています。
* オブジェクト指向との違いは、間に１階層置くことです。そうすることで、繋がった部分だけ修正せすれば良くなります。
* （感覚としては、電源タップを間に挟むか、直接電源引くかみたいな違いです。
* 間に挟むことで電源の位置が変わっても、タップを延長するだけでOKみたいな感覚です。）
*/
```

ここを理解しないと次も分からない。