プロ研内部向け
超雑C++講座

なにこれ？

簡単なシューティングゲームもどきを作成しながらC++~~とOpenGL~~をいじってみようっていう話

対象のひと

- プロ研の人
- オブジェクト指向知らない
- C言語そこそこ知ってる

対象じゃないひと

- Javaとかバリバリ書ける人
  - たぶんggった方が早い
- 単位が欲しい人
  - 授業内容と全く関係ありません

＃注意！

- 今回紹介するゲームの開発方法はあくまで一例です
- これが最適解とは限らないので，この方法にとらわれ過ぎないでください

なんでC++？

- 恐らく色んなゲーム開発で使われている
- 書き方にもよるけど，動作が速い
- 好み

Keywords

- Class - クラス
- Inheritance - 継承
- Polymorphism - 多様体
- STL - Standard Template Library

Class - クラス

Class - クラス

- クラスは「物」
- クラスはそれぞれ固有の関数や変数を持っている

Class - クラス

例）シューティングゲームの「物」

- プレイヤー
- 敵
- 弾

Class - クラス

例）プレイヤークラスの変数

- 現在位置
- 移動速度
- プレイヤーの画像
- HP

Class - クラス

例）プレイヤークラスの関数

- 現在位置を更新
- プレイヤーを描画

Class - クラス

- .hファイルにクラスの宣言
- .cppファイルにクラスの実際の中身

クラスを書いてみる

クラスを書いてみる

- 配布したプロジェクト内のObject.hとObject.cppにプレイヤークラスを作っていきます

Object.h

```
#pragma once

#include <glm/glm.hpp>
#include "Image.h"

class Player {
public:

     Player();
     ~Player();

     //現在位置を更新
     void Update();

     //プレイヤーを描画
     void Draw();

protected:
     glm::vec2 Position;          //現在位置
     glm::vec2 Velocity;          //移動速度
     Image *pImage;               //プレイヤーの画像
     int HP;
};
```

Object.cpp

```cpp
#include "Object.h"

Player::Player() {
	Position = glm::vec2(0.0f, 0.0f);
	Velocity = glm::vec2(0.0f, 0.0f);
	pImage = new Image("img/player.png");
	HP = 10;
}

Player::~Player() {
	delete pImage;
}

//現在位置を更新
void Player::Update() {
	Position += Velocity;
}

//プレイヤーを描画
void Player::Draw() {
	pImage->Draw(
		Position,
		glm::vec2(pImage->GetWidth(), pImage->GetHeight())
	);
}
```

Playerを表示してみる

クラスの使い方

- Playerクラス≒クラスの金型
- 使うためには，金型に鉄を流し込んで「実体」を作らないといけない

クラスの使い方

- new Player();
  - Playerの金型に鉄を流し込んでPlayerの実体を生成する

クラスの使い方

- Player *pPlayer = new Player();
  - 生成した実体はPlayer型のポインタとして返ってくる
  - これをPlayerのポインタ変数で保存

クラスの使い方

- Player *pPlayer1 = new Player();
  Player *pPlayer2 = new Player();
  - Playerは金型なので，Playerの実体を何個も生成できる

クラスの使い方

- pPlayer->Update();
  - pPlayerのUpdate関数を呼び出す

クラスの使い方

- delete pPlayer;
  - 要らなくなったら，delete演算子で削除する

Playerを表示してみる

- 配布したプロジェクト内のMain.cppで実際にPlayerクラスを使ってみます

Main.cpp

Main.cpp最初らへん

```
#pragma comment(lib, "opengl32.lib")

//プレイヤー
Player *pPlayer;

int main(int argc, char *argv[]) {
```

Main.cpp

Main.cppのmain関数内最後らへん

```
    glfwTerminate();

    delete pPlayer;

    return 0;
```

Main.cpp

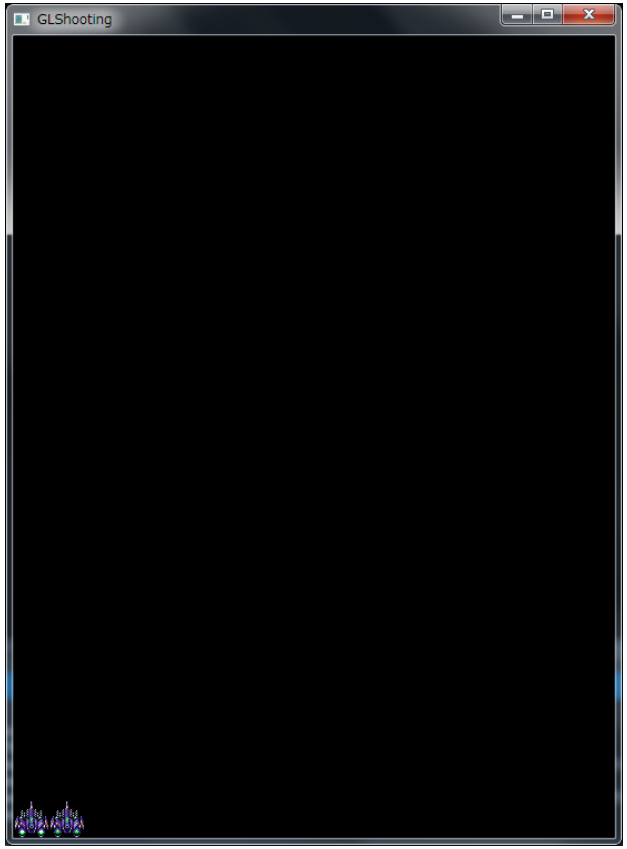
Main.cppの最後らへん

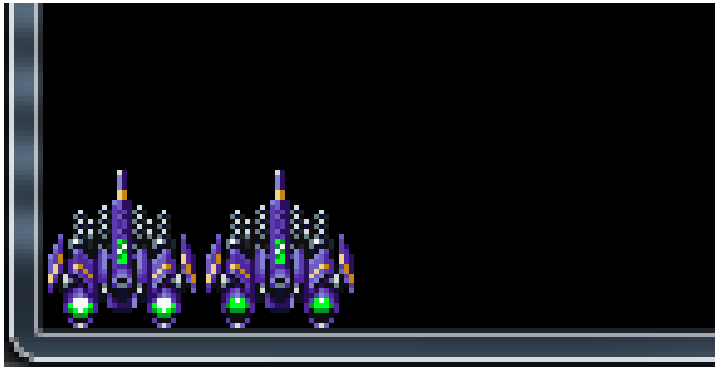
```
void Init() {
    pPlayer = new Player();
}

void Update() {
    pPlayer->Update();
}

void Draw() {
    pPlayer->Draw();
}
```

実行結果





機体が出た！

でもなんか2つ出てない？

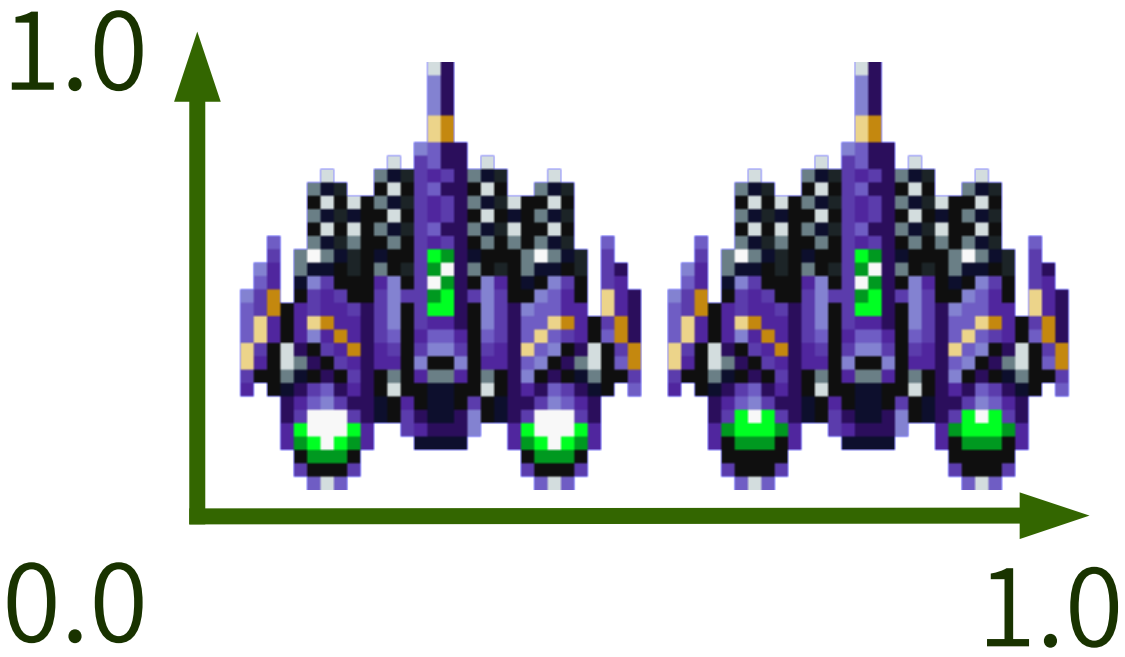
アニメーション

- 1枚の画像に複数のコマ
- 1つずつ切り出してアニメーション

Object.cpp

```
//プレイヤーを描画
void Player::Draw() {
    pImage->Draw(
        Position,                          //位置
        glm::vec2(pImage->GetWidth() * 0.5f,
            pImage->GetHeight()),//表示する大きさ
        false,                             //Y軸を反転
        glm::vec2(0.0f, 0.0f),   //テクスチャ上での開始位置
        glm::vec2(0.5f, 1.0f)    //テクスチャ上での大きさ
    );
}
```

テクスチャの座標系



実行結果

#書くのだるくなってきた

プレイヤーを動かしたい

- 矢印キーでプレイヤーを移動したい！
  - glfwGetKey

GLFWwindow* が要る

- glfwGetKeyは，GLFWwindowのポインタが要る
  - Playerクラスにポインタを渡す

Object.h

```
class Player {
public:

    Player(GLFWwindow *pWindow);
    ~Player();

-----------------------------------------

protected:
    GLFWwindow *pWindow;
    glm::vec2 Position;    //現在位置
```

Object.cpp

```
Player::Player(GLFWwindow *pWindow) {
    this->pWindow = pWindow;
    Position = glm::vec2(0.0f, 0.0f);
    Velocity = glm::vec2(0.0f, 0.0f);
    pImage = new Image("img/player.png");
    HP = 10;
}
```

Main.cpp

```
void Init() {
   pPlayer = new Player(pWindow);
}
```

glfwGetKey

```
int state = glfwGetKey(pWindow, GLFW_KEY_UP);
if( state == GLFW_PRESS ) {
   //上キーが押されている！
}
```

GLFW_KEY_

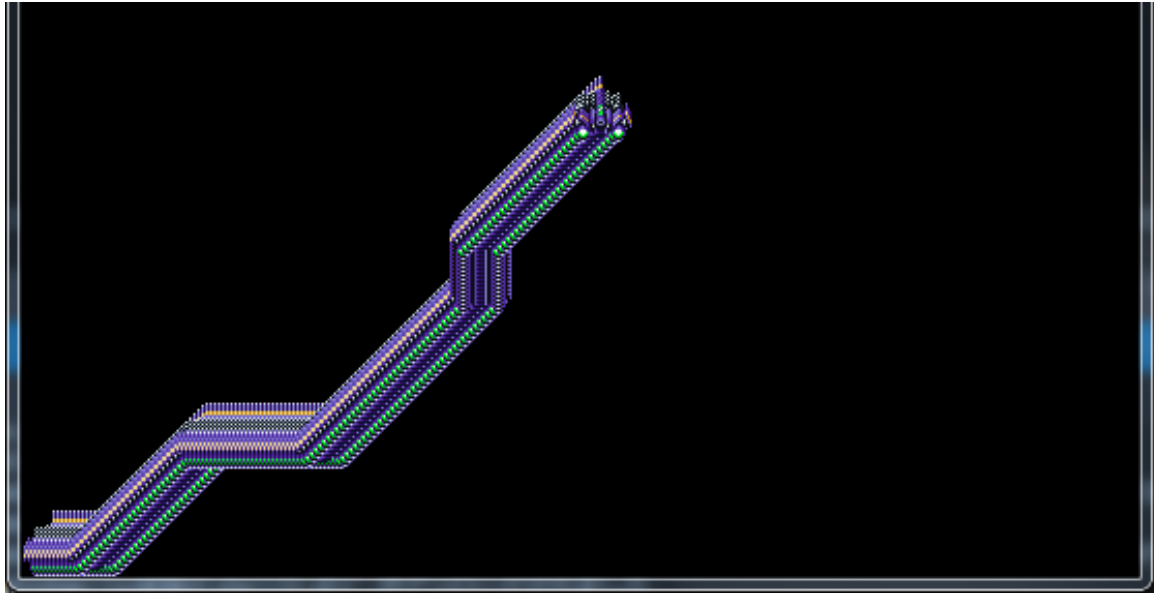
- GLFW_KEY_LEFT : 左キー
- GLFW_KEY_UP : 上キー
- GLFW_KEY_A : Aキー

Object.cpp

Player::Update関数

```
Velocity = glm::vec2(0.0f, 0.0f);
if( glfwGetKey(pWindow, GLFW_KEY_LEFT) == GLFW_PRESS ) {
    //左キーが押されたとき，左に移動する
    Velocity += glm::vec2(-1.0f, 0.0f);
}
if( glfwGetKey(pWindow, GLFW_KEY_RIGHT) == GLFW_PRESS ) {
    //右キーが押されたとき，右に移動する
    Velocity += glm::vec2(1.0f, 0.0f);
}
```

実行結果



動いたけど
分身してる！

Main.cpp

```
void Update() {
    //画面クリア
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    pPlayer->Update();

}
```

実行結果

Inheritance - 継承

敵を出したい

- Playerと同じように敵を表すEnemyクラスを作る

Object.h

```
class Enemy {
public:

    Enemy();
    ~Enemy();

    //現在位置を更新
    void Update();

    //敵を描画
    void Draw();

protected:
    glm::vec2 Position;   //現在位置
    glm::vec2 Velocity;   //移動速度
    Image *pImage;        //敵の画像
    int HP;
};
```

```
class Player {
public:

    Player(GLFWwindow *pWindow);
    ~Player();

    //現在位置を更新
    void Update();

    //プレイヤーを描画
    void Draw();

protected:
    GLFWwindow *pWindow;
    glm::vec2 Position;     //現在位置
    glm::vec2 Velocity;     //移動速度
    Image *pImage;          //プレイヤーの画像
    int HP;
};
```

```
class Enemy {
public:

    Enemy();
    ~Enemy();

    //現在位置を更新
    void Update();

    //敵を描画
    void Draw();

protected:
    glm::vec2 Position;     //現在位置
    glm::vec2 Velocity;     //移動速度
    Image *pImage;          //敵の画像
    int HP;
};
```

Inheritance - 継承

- 継承したクラスが継承されたクラスの関数・変数を引き継ぐ

Object.h

```
class ShipBase {
public:

    ShipBase();
    virtual ~ShipBase();

    //現在位置の更新
    virtual void Update();

    //描画
    virtual void Draw();

protected:
    glm::vec2 Position;      //現在位置
    glm::vec2 Velocity;      //移動速度
    Image *pImage;           //機体の画像
    int HP;
};
```

PlayerとEnemyの共通部分を出したクラスShipBase

Object.cpp

```cpp
ShipBase::ShipBase() {
    Position = glm::vec2(0.0f, 0.0f);
    Velocity = glm::vec2(0.0f, 0.0f);
    pImage = nullptr;
    HP = 0;
}

ShipBase::~ShipBase() { }

//現在位置の更新
void ShipBase::Update() {
    Position += Velocity;
}

//描画
void ShipBase::Draw() { }
```

PlayerとEnemyの共通部分を出したクラスShipBase

C++における継承

クラスPlayerはShipBaseを継承する

```
class Player : public ShipBase {
```

Object.h

```
class Player : public ShipBase {
public:

    Player(GLFWwindow *pWindow);
    virtual ~Player();

    //現在位置を更新
    virtual void Update();

    //プレイヤーを描画
    virtual void Draw();

protected:
    GLFWwindow *pWindow;
};
```

Object.cpp

```
Player::Player(GLFWwindow *pWindow) {
    this->pWindow = pWindow;
    pImage = new Image("img/player.png");
}

Player::~Player() {
    delete pImage;
}
```

Object.cpp

```
//現在位置を更新
void Player::Update() {

    Velocity = glm::vec2(0.0f, 0.0f);
    if( glfwGetKey(pWindow, GLFW_KEY_LEFT) == GLFW_PRESS ) {
        Velocity += glm::vec2(-1.0f, 0.0f);
    }
    if( glfwGetKey(pWindow, GLFW_KEY_RIGHT) == GLFW_PRESS ) {
        Velocity += glm::vec2(1.0f, 0.0f);
    }
    if( glfwGetKey(pWindow, GLFW_KEY_UP) == GLFW_PRESS ) {
        Velocity += glm::vec2(0.0f, 1.0f);
    }
    if( glfwGetKey(pWindow, GLFW_KEY_DOWN) == GLFW_PRESS ) {
        Velocity += glm::vec2(0.0f, -1.0f);
    }

    ShipBase::Update();

}
```

Object.cpp

```
//プレイヤーを描画
void Player::Draw() {
    pImage->Draw(
        Position,
        glm::vec2(pImage->GetWidth() * 0.5f,
            pImage->GetHeight()),
        false,
        glm::vec2(0.0f, 0.0f),
        glm::vec2(0.5f, 1.0f)
    );

    ShipBase::Draw();

}
```

Enemyクラスを作成

クラスEnemyはShipBaseを継承する

```
class Enemy : public ShipBase {
```

Object.h

```
class Enemy : public ShipBase {
public:

    Enemy();
    virtual ~Enemy();

    //現在位置を更新
    virtual void Update();

    //プレイヤーを描画
    virtual void Draw();

};
```

Object.cpp

```
Enemy::Enemy() {
    pImage = new Image("img/enemy1.png");
}
Enemy::~Enemy() {
    delete pImage;
}

//現在位置を更新
void Enemy::Update() {
    ShipBase::Update();
}
```

Object.cpp

```
//プレイヤーを描画
void Enemy::Draw() {
    pImage->Draw(
        Position,
        glm::vec2(pImage->GetWidth() * 0.5f,
            pImage->GetHeight()),
        true,
        glm::vec2(0.0f, 0.0f),
        glm::vec2(0.5f, 1.0f)
    );

    ShipBase::Draw();

}
```

Main.cpp

```
//プレイヤー
Player *pPlayer;

//敵
Enemy *pEnemy;

int main(int argc, char *argv[]) {
```

```
glfwTerminate();

delete pPlayer;
delete pEnemy;

return 0;
```

Main.cpp

```
void Init() {
    pPlayer = new Player(pWindow);
    pEnemy = new Enemy();
}

void Update() {
    //画面クリア
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    pPlayer->Update();
    pEnemy->Update();

}

void Draw() {
    pPlayer->Draw();
    pEnemy->Draw();
}
```

実行結果

敵を動かしたい

- Setter/Getter
  - ShipBaseに位置や速度を設定/取得する関数を作成

Object.h

```
//現在位置を設定
void SetPosition(const glm::vec2 &pos) {
    this->Position = pos;
}

//現在位置を取得
const glm::vec2 &GetPosition() const {
    return Position;
}

//移動速度を設定
void SetVelocity(const glm::vec2 &vel) {
    this->Velocity = vel;
}

//移動速度を取得
const glm::vec2 &GetVelocity() const {
    return Velocity;
}
```

Main.cpp

```
void Init() {
    pPlayer = new Player(pWindow);
    pPlayer->SetPosition(glm::vec2(SCREEN_WIDTH * 0.5f, 20.0f));
    pEnemy = new Enemy();
    pEnemy->SetPosition(glm::vec2(
        SCREEN_WIDTH * 0.5f, SCREEN_HEIGHT
    ));
    pEnemy->SetVelocity(glm::vec2(0.0f, -2.0f));
}
```

ShipBaseにしか追加してない

- PlayerとEnemyはShipBaseを継承しているので，ShipBaseに追加されたものは全て引き継ぐ

弾を発射したい

- Playerに弾を発射させたい
  - PlayerBulletクラスを作成

Object.h

```
class PlayerBullet : public ShipBase {
public:

    PlayerBullet();
    virtual ~PlayerBullet();

    //現在位置を更新
    virtual void Update();

    //弾を描画
    virtual void Draw();

};
```

Object.cpp

```
PlayerBullet::PlayerBullet() {
    pImage = new Image("img/player_bullet.png");
}
PlayerBullet::~PlayerBullet() {
    delete pImage;
}

//現在位置を更新
void PlayerBullet::Update() {
    ShipBase::Update();
}

//弾を描画
void PlayerBullet::Draw() {
    pImage->Draw(
        Position,
        glm::vec2(pImage->GetWidth(), pImage->GetHeight())
    );

    ShipBase::Draw();

}
```

Main.cpp

```
//プレイヤー
Player *pPlayer;

//プレイヤーの弾
PlayerBullet *pBullet;

//敵
Enemy *pEnemy;
```

```
pPlayer->SetPosition(glm::vec2(SCREEN_WIDTH * 0.5f, 20.0f));

pBullet = new PlayerBullet();
pBullet->SetPosition(pPlayer->GetPosition());
pBullet->SetVelocity(glm::vec2(0.0f, 6.0f));

pEnemy = new Enemy();
```

Main.cpp

```
void Update() {
    //画面クリア
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    pPlayer->Update();
    pEnemy->Update();
    pBullet->Update();

}

void Draw() {
    pPlayer->Draw();
    pEnemy->Draw();
    pBullet->Draw();
}
```

実行結果

STL & Polymorphism

Worldクラスの導入

- プレイヤーとか敵とか弾を一括で管理
  - 世界を表すクラス

std::vector

- 可変長配列
  - 配列を必要な分だけ自動的に大きくしたり小さくしたりしてくれるヤツ

std::vector

```
//int型の可変長配列
std::vector<int> intarray;

//配列の末尾に要素を追加
intarray.push_back(1);
intarray.push_back(2);
intarray.push_back(3);

//番号を指定して要素を取得
printf("0番目の要素は: %d", intarray[0]);
printf("2番目の要素は: %d", intarray[2]);

//配列の大きさ
printf("配列の大きさは: %d", intarray.size());

//全部消す
intarray.clear();
```

他のリストとか

- std::list
  - リスト構造 (javaだとArrayList？)
- std::map
  - 連想配列 (javaだとHashMap？)

他のリストとか

- std::stack　　スタック
- std::array　　固定長配列
- std::deque　　効率の良いリスト？
- std::set　　要素が重複しないリスト

iterator

- vectorとかlistとかmapの中での位置を示すのに利用
  - std::vector<int>::iterator

iterator

```
//0〜9の値が入ったvectorを作成
std::vector<int> intarray;
for( int i = 0; i < 10; i++ ) {
    intarray.push_back(i);
}

//intarrayの最初の要素を示すイテレータを取得
std::vector<int>::iterator it = intarray.begin();

printf("first: %d\n", *it);      //最初の要素を取得
it++;                            //イテレータを次に進める
printf("second: %d\n", *it);     //2番目の要素を取得

while( it != intarray.end() ) {
    //intarrayの最後までの要素を取得
    printf("%d\n", *it);
    it++;
}
```

ポリモーフィズム

PlayerがShipBaseを継承しているとき

```
ShipBase *pShip = new Player(pWindow);
```

何がおいしいの

PlayerもEnemyもShipBaseとして
統一的に扱える

```
ShipBase *pPlayer = new Player(pWindow);
ShipBase *pEnemy = new Enemy();
```

何がおいしいの

ShipBaseとして統一できると，
ShipBase型のvectorに全部入れられる

```
std::vector<ShipBase*> objects;
objects.push_back(pPlayer);
objects.push_back(pEnemy);
```

Worldクラスの導入

- Worldクラスを書いてみよう