

# 1. 全体的な共通点

## 1.1. アピールポイントについて

- 作品では「何をやりたいのか」「何が得意なのか」が明確になっていると良いです
  - 会社によって異なると思いますが、例えば弊社のアンケートでは以下の分野（一部抜粋）のどれに興味があるのかを質問します
    - 「フレームワーク設計」「プレイヤーアルゴリズム」「ユーザーインターフェース」「3Dモデル制御」「ポストエフェクト」「ツール開発」「物理演算」
  - **それらの分野の技術が一定の水準を満たしている**と、採用確率が上がると思います
    - →ゲームを動かして "1分" で「技術力が伝わる」ようにする

項目	説明
フレームワーク設計	ゲームフレームワークの自作。グラフィックス、モーション制御、リソース管理、アセットワークフローの構築、ツール開発など
プレイヤーアルゴリズム	移動や攻撃の制御。アクションゲームであれば操作の気持ちよさ、複雑な状態遷移の管理。AI（経路探索、ビヘイビアツリー、自作スクリプトなど）
3Dモデル制御	アニメーションの制御。モーションブレンド。首の回転や視線の制御。物理制御。IK
ポストエフェクト	ライティング。ブルーム、色収差、被写界深度など
物理演算	ベクトル、行列。クォータニオン。衝突判定、OBBやレイキャスト。バネや揺れもの、クロスシミュレーション、水面の物理的な表現

- (※) 上記は一例ですので、実装しないとダメという意味ではありません

## 1.2. コードレビューについて

- コードレビューは「経験則で正しかったこと」なので、**あくまで1つの意見**として捉えてもらえればと思います
- 採用試験では、C++の理解度が問われるので、日頃から言語仕様についての本を読んだり、C++の記事を読んだりして情報を収集しておくの良いです

## 2. 高山桃也 (Rayleigh)



### 2.3. 概要

- ジャンル : 3D横スクロールアクション
- 開発時期 : 2023年10月 1ヶ月間
- 開発人数 : 1人
- 開発環境 : DirectX9、C++
- 使用ツール : Maya、メタセコイア、PhotoShop
- 担当箇所 : プログラミングとモデルは全て自作しました

スピーディーなアクションができる主人公を操作して、ステージを進み、ボスを倒すゲームです。


項目	詳細
アピールポイント	<p>攻撃時や被弾時のリアクションとしてブラーを実装しました。</p> <p>これによって収縮や拡大などでの表現も広がり、よりプレイヤーに対して分かりやすいリアクションを返せています。</p> <p>また、今作はエフェクトの管理やプレイヤーのパラメーターを外部に出すことで、バランス調整やリソースの量産と実装をより効率的に行えるようにしました。</p>
苦労した点	<p>テストプレイをしてもらった上での調整は苦労しました。</p> <p>特にプレイヤーのアクションごとのパラメーターの調整に関しては、上級者が突き詰められ、初心者でもクリアはできるようにするように調整しました</p>
添削してほしいポイント	<ul style="list-style-type: none"><li>• プレイヤーのアクションの手触り</li><li>• リアクションの足りない部分、過剰な部分</li></ul>

## 2.4. 添削

### 2.4.1. 良いところ

- 2段ジャンプやダッシュ時に残像が出るのが動かして気持ちが良い
- コードも丁寧に書かれていて良いです

### 2.4.2. 添削してほしいポイント

添削	内容
プレイヤーのアクションの手触り	<p>総じてアクションの気持ちよさがあるので問題ありませんが、あえて言うなら、硬直時間のある攻撃を用意するなどアクションに緩急があると手触りが良くなるかもしれません。あとは基本の移動速度が高いためダッシュを使う機会があまりない、2段ジャンプ斬りが強いので通常攻撃を使う機会があまりない、ということくらいでしょうか。</p> <p>細かいところでは、ジャンプアクションゲームは飛び移る先（地面）が見えないと、ジャンプするのに躊躇するので、視界に飛び移る先が表示されていると良いと思います。</p>  <ul style="list-style-type: none"><li>• カメラを引く、移動方向に合わせてカメラを先に移動する、水平方向の移動速度を小さくする、など</li></ul>
リアクションの足りない部分、 過剰な部分	<p>細かくエフェクトが入っていてとても良いと思います。あえて言うなら、通常攻撃に斬撃のエフェクトがない、着地時のエフェクトがない、走りエフェクトがない、ということでしょうか。</p>
さらに伸ばすなら	<p>グラフィック周りにこだわりを感じたので、ライティングの表現やポストエフェクト（色収差やディゾルブ、ブルームなど）に挑戦しても良いかもしれません</p>

2.4.3. コードレビュー

項目	説明
クラスのstaticメンバ変数の命名	<p>コーディングルールなので、学校の規則でそうになっているなら問題ないですが、"m_" は "<b>s_</b>" にしたほうがわかりやすく良いです</p>
static関数とそうでない関数はブロックを分けた方が良い	<p>■ 変更前</p> <p>Enlighter: C++</p> <pre>static CBlock *Create(D3DXVECTOR3 pos, D3DXVECTOR3 rot,TYPE type); HRESULT Init(void); void Uninit(void); void Update(void); void Draw(void); static HRESULT Load(char *pPath);          // 読み static void Save(void); // 保存 static void Delete(int nIdx); // 部分削除処理 static void DeleteAll(void); // 全削除処理 static int GetNumAll(void) { return m_nNumAll; } static CBlock **GetBlock(void) { return &amp;m_apBlock[0]; } static float CheckShadow(D3DXVECTOR3 pos); static void LoadModel(void); static void DeleteIdx(void); static int *GetIndex(void) { return m_pIdxObject; }</pre> <p>■ 変更後</p> <p>Enlighter: C++</p> <pre>// ----- public static function(s) static CBlock *Create(D3DXVECTOR3 pos, D3DXVECTOR3 rot,TYPE type); static HRESULT Load(char *pPath); // 読み static void Save(void); // 保存 static void Delete(int nIdx); // 部分削除処理 static void DeleteAll(void); // 全削除処理 static int GetNumAll(void) { return m_nNumAll; } static CBlock **GetBlock(void) { return &amp;m_apBlock[0]; } static float CheckShadow(D3DXVECTOR3 pos); static void LoadModel(void); static void DeleteIdx(void); static int *GetIndex(void) { return m_pIdxObject; }  // ----- public function(s) HRESULT Init(void); void Uninit(void); void Update(void); void Draw(void);め</pre>
変数は宣言と同時に初期化した方がよい。	<p>■ 変更前</p> <p>Enlighter: C++</p> <pre>private:     TYPE m_type;     static int *m_pIdxObject; // モデルのタイプ番号のポインタ     static CBlock *m_apBlock[NUM_OBJECT]; // ブロックの配列     static int m_nNumAll; // 総数     CCollisionCube *m_pCollisionCube; // 立方体の当たり判定     int m_nLife; // 体力</pre>
• コンストラクタで初期化すると宣言と初期化が離れてしまう	

項目	説明
<ul style="list-style-type: none"> <li>初期化子 {} で初期化がお すすめ</li> </ul>	<p>■ 変更後</p> <p>Enlighter: C++</p> <pre>private: // ----- static var(s). static int *m_pIdxObject; // モデルのタイプ番号のポインタ static CBlock *m_apBlock[NUM_OBJECT]; // ブロックの配列 static int m_nNumAll; // 総数 private: // ----- var(s). TYPE m_type = {}; CCollisionCube *m_pCollisionCube = nullptr; // 立方体の当たり判定 int m_nLife = 0; // 体力</pre>
	<p>Enlighter: C++</p> <pre>for (int nCnt = 0; nCnt &lt; NUM_OBJECT; nCnt++) { if (ppCollision[nCnt] != nullptr) { if (ppCollision[nCnt]-&gt;GetType() == TYPE_CUBE &amp;&amp; ppCollision[nCnt] != this) { if (ppCollision[nCnt]-&gt;GetTag() != TAG_BLOCK) {// タグに合わなければ繰り返し continue; } ... } }</pre>
ガード節でネストを減らす	<p>■ 変更後</p> <p>Enlighter: C++</p> <pre>for (int nCnt = 0; nCnt &lt; NUM_OBJECT; nCnt++) { if (ppCollision[nCnt] == nullptr) { continue; // 無効なコリジョン. } if (ppCollision[nCnt]-&gt;GetType() != TYPE_CUBE) { continue; // Cube以外は判定不要. } if (ppCollision[nCnt] == this) { continue; // 自分自身は判定しない. }  if (ppCollision[nCnt]-&gt;GetTag() != TAG_BLOCK) {// タグに合わなければ繰り返し continue; } ... }</pre>

### 3. 金崎朋弥 (斬捨御免)



### 3.5. 概要

- ◇使用言語：C++
- ◇SDK：DirectX9
- ◇開発期間：2023/10/02～2024/02/29 (約6ヵ月)
- ◇ジャンル：横スクロール2Dアクション
- ◇GitHubURL: [https://github.com/TomoyaKanazaki/Act\\_OneMonth.git](https://github.com/TomoyaKanazaki/Act_OneMonth.git)

直感的で気持ちのいいアクションを追求！

項目	詳細
工夫した点	<ul style="list-style-type: none"><li>プレイヤーの攻撃の当たり判定に線分の当たり判定を実装</li><li>初めての人型モデルで初めてのモーションを作成</li><li>明るく派手なエフェクトでも何が起きているのかわかりやすいように調整をしました。</li></ul>
反省点	<ul style="list-style-type: none"><li>初めての試みに時間をかけすぎてしまった。スケジューリングを大切にする。</li><li>度重なる仕様変更を行った。作り始める前の段階でもっとしっかりと仕様を固める。</li></ul>

### 3.6. 添削

#### 3.6.4. 良いところ

- エフェクトにこだわりがあって良い
  - 斬撃やヒットのエフェクトに気持ちよさがある
- ボスの攻撃パターンがいくつかあってこだわりを感じました
- プログラムが全体的に良く書けている印象です

### 3.6.5. アドバイス

項目	説明
コリジョン	線分と円の交差判定はアピールポイントとしては弱いので、もしコリジョンをアピールポイントにしたいのであれば、2Dであれば坂道の判定を作ってみたり、OBBなど3D空間の当たり判定に挑戦したほうが良いかもしれません。ビリヤードのように衝突検知・衝突応答をする物理挙動を自作してみるのも良いと思います。
エフェクト	もしエフェクトをアピールポイントにしたい場合は、シェーダーを使ったポストエフェクト（ブルーム、ディゾルブ、ディストーションなど）に挑戦すると良いと思います。
2Dゲームはアピールが弱くなる	やりたいこと次第ですが、2Dゲームは技術力のアピールが弱くなる傾向があるため、3D空間を使用したゲームに挑戦しても良いかもしれません。

### 3.6.6. コードレビュー

項目	説明
ライブラリの関数を使う	<p>ひょっとしたらZを使っているのかもしれませんが、D3DXVECTOR3でベクトルの距離を求める場合は <b>D3DXVec3Length()</b> が使えます。</p> <p><b>Enlighter: C++</b></p> <pre>// 各ベクトルの大きさを求める float lengthLine = sqrtf((vecLine.x * vecLine.x) + (vecLine.y * vecLine.y)); float lengthToPos = sqrtf((vecToPos.x * vecToPos.x) + (vecToPos.y * vecToPos.y));</pre>
変数は宣言時に初期化する	<p>■ 変更前</p> <p><b>Enlighter: C++</b></p> <pre>//メンバ変数 D3DXVECTOR3 m_CenterPos; //中心座標 D3DXVECTOR3 m_posStart; // 攻撃のスタート地点 float m_fDashAngle;</pre> <p>■ 変更後</p> <p><b>Enlighter: C++</b></p> <pre>//メンバ変数 D3DXVECTOR3 m_CenterPos = {}; //中心座標 D3DXVECTOR3 m_posStart = {}; // 攻撃のスタート地点 float m_fDashAngle = 0.f;</pre>

項目	説明
当たり判定の処理を抽象化する	<p>円は「中心・半径」、線分は「2点 (または始点+方向)」で抽象化できるので、衝突判定を行う関数を作っておくと他に使い回せて良いと思います。</p> <ul style="list-style-type: none"><li>例： <code>bool Intersect_RayVsCircle(const RAY&amp; ray, const CIRCLE&amp; circle);</code></li></ul> <p>もし衝突判定をアピールしたい場合は、「矩形と線分」「線分同士」「三角形と線分」というように様々なケースを作ってみても良いかもしれませんが、2Dの当たり判定をアピールするのは弱いので、できれば3D空間の当たり判定を実装した方が良いです。</p>
ガード節でネストを減らす	<p>■変更前</p> <p><b>Enlighter: C++</b></p> <pre>void CPlayer::Hit() {     //当たり判定の生成     for (int nCntPriority = 0; nCntPriority &lt; PRIORITY_NUM; nCntPriority++)     {         while (pObj != NULL)         {             if (pObj-&gt;GetType() == CObject::TYPE_ENEMY) //敵の場合             {                 if (pObj-&gt;GetObjState() == CObject::NORMAL) // 通常状態の場合                 {                     // 線分の判定                     if (0.0f &lt;= t &amp;&amp; t &lt;= 1.0f)                     {                         // 判定距離の比較                         if (m_fHitLength * m_fHitLength &gt;= (vecToCross.x * vecToCross.x) + (vecToCross.y * vecToCross.y))                         {                             // 当たっていた時の演出系処理                         }                     }                 }             }         }     }      //次のアドレスにずらす     pObj = pObjNext; }</pre> <p>■変更後</p> <p><b>Enlighter: C++</b></p> <pre>void CPlayer::Hit() {     //当たり判定の生成     for (int nCntPriority = 0; nCntPriority &lt; PRIORITY_NUM; nCntPriority++)     {         //先頭のアドレスを取得         CObject* pObj = CObject::GetTop(nCntPriority);          for(; pObj != NULL; pObj = pObj-&gt;GetNext()) {             if (pObj-&gt;GetType() != CObject::TYPE_ENEMY) {                 continue; // 敵でない.             }             if (pObj-&gt;GetObjState() != CObject::NORMAL) {                 continue; // 通常状態でないので判定不要.             }              // 線分の判定.             if (t &lt; 0.0f    1.0f &lt; t) {                 continue; // 線分内に存在しない.             }             // 判定距離の比較             if (m_fHitLength * m_fHitLength &gt;= (vecToCross.x * vecToCross.x) + (vecToCross.y * vecToCross.y))             {                 // 当たっていた時の演出系処理             }         }     } }</pre>



項目	説明
	<pre>     }   } } </pre>
媒介変数tの計算式の書き方が少し気になる	<p><b>Enlighter: C++</b></p> <pre> // 媒介変数tを求める float t = (lengthLine * lengthToPos) / (lengthLine * lengthLine);  // 線分の判定 if (0.0f &lt;= t &amp;&amp; t &lt;= 1.0f) {     // 目標点から直線に垂線を下した時の交点を求める     D3DXVECTOR3 posCross = m_posStart + (t * vecLine); } </pre> <p>「<b>lengthLine * lengthLine</b>」とすることで<b>正の値</b>になるようにしていると思われます（※間違っているかもしれませんが…）が、絶対値が欲しいのであれば <code>abs()</code> を使うとわかりやすくなります。</p>

## 4. 大原怜将 (アキバでリアルファイト)



### 4.7. 概要

- タイトル：アキバでリアルファイト
- プレイ人数：1人
- 開発環境：DirectX9
- 使用した言語：C++
- 開発期間：2024/1/11～2/29

項目	詳細
こだわったポイント	電子レンジを使用したヒートアクション
添削してほしいポイント	<ul style="list-style-type: none"><li>ゲームの良いポイント、悪いポイント</li><li>ソースコードで改善したほうが良いポイント</li></ul>

## 4.8. 添削

### 4.8.7. 良いところ

- 連続技やつかみ技がある
- 演出のセンスが良い

### 4.8.8. アドバイス

破壊オブジェクトや複雑な地形、移動床などのギミックがあるとアピールしやすいかもしれません。

また敵のAIにこだわってみるのも良いかもしれません。

### 4.8.9. コードレビュー

項目	説明
#defineを避けてconstを使う	<div>Enlighter: C++</div> <pre>// マクロ定義 #define MAX_MODEL (64) // モデルの最大数 #define MAX_NAME (128) // テクスチャの最大文字数</pre>

項目	説明
ガード節でネストを減らす	■修正前
定数にはconstをつける	Enlighter: C++
適切なコメントをつける	<pre>//===== // マップにある建物との当たり判定 //===== void CCollision::ItemAttack(CObjectX * pobj) {     int nNum = 0;     float PlayerfRadius = 50.0f;     float fRadius = 75.0f;     CEnemy **ppEnemy = nullptr;      if (CGame::GetEnemyManager() != nullptr)     {         ppEnemy = CGame::GetEnemyManager()-&gt;GetEnemy();         nNum = CGame::GetEnemyManager()-&gt;GetNum();     }      if (pobj != nullptr)     {         for (int nCount = 0; nCount &lt; nNum; nCount++)         {             if (ppEnemy[nCount] != nullptr)             {                 float circleX = ppEnemy[nCount]-&gt;GetPosition().x - (CGame::GetPlayer()-&gt;GetPosition().x + pobj-&gt;GetPosition().x);                 float circleZ = ppEnemy[nCount]-&gt;GetPosition().z - (CGame::GetPlayer()-&gt;GetPosition().z + pobj-&gt;GetPosition().z);                 float c = 0.0f;                  c = (float)sqrt(circleX * circleX + circleZ * circleZ);                  if (c &lt;= fRadius + PlayerfRadius)                 {                     ppEnemy[nCount]-&gt;SetRotition(-CGame::GetPlayer()-&gt;GetRotition());                     ppEnemy[nCount]-&gt;SetMove(D3DXVECTOR3(sinf(CGame::GetPlayer()-&gt;GetRotition().y) * -3.0f, 1.0f, cosf(CGame::GetPlayer()-&gt;GetRotition().y) * -3.0f));                     ppEnemy[nCount]-&gt;SetState(CEnemy::STATE_DAMEGE);                     int nLife = ppEnemy[nCount]-&gt;GetLife();                     nLife -- 1;                     ppEnemy[nCount]-&gt;SetLife(nLife);                 }             }         }     } }</pre>
適切な関数名にする	■修正後
	Enlighter: C++
	<pre>//===== // マップにある建物との当たり判定 //===== void CCollision::ItemAttack(CObjectX * pobj) {     if (pobj == nullptr) {         return; // 無効なオブジェクト.     }      // 敵管理を取得.     auto* pEnemyMgr = CGame::GetEnemyManager();     if (pEnemyMgr == nullptr) {         return // 敵管理が無効.     }      // 定数にはconstをつける.     const float PlayerfRadius = 50.0f;     const float fRadius = 75.0f;      CEnemy **ppEnemy = CGame::GetEnemyManager()-&gt;GetEnemy();     int nNum = CGame::GetEnemyManager()-&gt;GetNum();      for (int nCount = 0; nCount &lt; nNum; nCount++)     {         if (ppEnemy[nCount] == nullptr) {</pre>

項目

説明

```
        continue; // 無効な敵。
    }

    float circleX = ppEnemy[nCount]->GetPosition().x - (CGame::GetPlayer()->GetPosition().x + pObj->GetPosition().x);
    float circleZ = ppEnemy[nCount]->GetPosition().z - (CGame::GetPlayer()->GetPosition().z + pObj->GetPosition().z);
    float c = (float)sqrt(circleX * circleX + circleZ * circleZ);

    if (c > fRadius + PlayerfRadius) {
        continue; // 衝突してない。
    }

    // 敵のダメージ処理。
    ppEnemy[nCount]->SetRotition(-CGame::GetPlayer()->GetRotition());
    ppEnemy[nCount]->SetMove(D3DXVECTOR3(sin(CGame::GetPlayer()->GetRotition().y) * -3.0f, 1.0f, cosf(CGame::GetPlayer()->GetRotition().y) *
-3.0f));
    ppEnemy[nCount]->SetState(CEnemy::STATE_DAMEGE);
    int nLife = ppEnemy[nCount]->GetLife();
    nLife -= 1;
    ppEnemy[nCount]->SetLife(nLife);
}
}
```

■修正前

Enlighter: C++

```
// 取得系
D3DXVECTOR3 GetPosition(void) { return m_Info.pos; } // 位置
D3DXVECTOR3 GetRotition(void) { return m_Info.rot; } // 向き
D3DXVECTOR3 GetMove(void) { return m_Info.move; } // 移動量
STATE GetState(void) { return m_Info.state; } // 状態
int GetLife(void) { return m_Info.nLife; } // 体力
int GetIdxID(void) { return m_Info.nIdxID; }
TYPE GetType(void) { return m_Type; }
CCharacter **GetCharcter(void) { return m_apModel; }
static int GetNumAll(void) { return m_nNumAll; }
CMotion *GetMotion(void) { return m_pMotion; }
MOBILITY GetMobility(void) { return m_Mobility; }
static CEnemy *GetTop(void) { return m_pTop; }
CEnemy *GetNext(void) { return m_pNext; }
```

static関数と非static関数  
を分ける

■修正後

Enlighter: C++

```
// 取得系
// ----- static function(s).
static int GetNumAll(void) { return m_nNumAll; }
static CEnemy *GetTop(void) { return m_pTop; }
// ----- function(s)
D3DXVECTOR3 GetPosition(void) { return m_Info.pos; } // 位置
D3DXVECTOR3 GetRotition(void) { return m_Info.rot; } // 向き
D3DXVECTOR3 GetMove(void) { return m_Info.move; } // 移動量
STATE GetState(void) { return m_Info.state; } // 状態
int GetLife(void) { return m_Info.nLife; } // 体力
int GetIdxID(void) { return m_Info.nIdxID; }
TYPE GetType(void) { return m_Type; }
CCharacter **GetCharcter(void) { return m_apModel; }
CMotion *GetMotion(void) { return m_pMotion; }
MOBILITY GetMobility(void) { return m_Mobility; }
CEnemy *GetNext(void) { return m_pNext; }
```

複雑条件式は関数化する

■問題のダメージ判定部分

項目	説明
	<p><b>Enlighter: C++</b></p> <pre>//===== // ダメージ処理 //===== void CEnemyBoss::Damage(int damage, float blowaway, CPlayer::ATTACKTYPE act) {     if (m_Info.state != STATE_DAMEGE &amp;&amp; m_Info.state != STATE_HEATDAMEGE &amp;&amp; m_Info.state != STATE_PAINFULDAMAGE         &amp;&amp; m_Info.state != STATE_DETH &amp;&amp; (m_Info.state != STATE_ATTACK    CGame::GetPlayer()-&gt;GetState() == CPlayer::STATE_HEAT))     {         m_Info.nLife -= damage;         ...     } }</pre> <p>■ダメージ判定を関数化する</p> <p><b>Enlighter: C++</b></p> <pre>// ダメージ判定を行うかどうか. bool CEnemyBoss::CanDamage() {     switch(m_Info.state) {         case STATE_DAMEGE: // ダメージ中.         case STATE_HEATDAMEGE: // ヒートアクションのダメージ中.         case STATE_PAINFULDAMAGE: // クリティカルダメージ中.         case STATE_DETH: // 死亡中.             return false; // 判定不要.         default:             break;     }      if(m_Info.state == STATE_ATTACK) {         // 攻撃中の場合.         if(CGame::GetPlayer()-&gt;GetState() != CPlayer::STATE_HEAT) {             // プレイヤーがヒートアクションでない場合             return false; // ダメージ判定不要.         }     }      // ダメージ判定を行う.     return true; }</pre>
typoをしない	<p>「STATE_DETH」は「STATE_DEATH」が正しい綴り。</p> <p>些細なことと思うかもしれませんが、よく使われる英単語を間違えるとコードの品質を疑われます…。（調べたら スラングで "DETH" を "DEATH" とするので正しいかもしれませんが、そうだったとしてもスラングを製品コードに入れるのはNGです）</p>

## 5. 高田佳依 (UniqueBall)



## 5.9. 概要

- ◇タイトル : UniqueBall
- ◇制作人数 : 1人
- ◇制作期間 : 2か月(2024/1月~2024/3月)
- ◇担当範囲 : ほぼ全て
- ◇使用言語 : C++
- ◇SDK : DirectX9
- ◇ジャンル : ドッジボールアクション
- ◇プレイ人数 : 2人

項目	詳細
作品概要	<p>ドッジボールをイメージして制作した作品です。ボールを扱うことと複数プレイヤーのゲームを作るのが初めてでした。</p> <p>そのため、そこでバグが出ないように注力しました。プレイは必ずコントローラーでお願いいたします。</p>
実装内容	<ul style="list-style-type: none"><li>画面分割</li><li>各プレイヤー追従カメラ</li><li>プレイヤーがボール所持時投げを行える</li><li>キャッチボールが出来るBOT</li><li>ボールとプレイヤーとの当たり判定</li></ul>

## 5.10. 添削

### 5.10.10. 良いところ

- 多彩なアクション（ジャンプ、ダッシュ、ロックオン、ボール投げる）

### 5.10.11. アドバイス

項目	説明
操作方法をわかりやすくする	多彩なアクションをわかりやすくするためにゲーム中にもボタンがどのアクションに対応しているかを表示した方が良いかもしれません
物理演算やコリジョンに力を入れると良いかもしれません	<p>現状の物理演算はアピールポイントとして弱い気がするので、もう少しボールでの遊びや物理挙動を取り入れると良いかもしれません。</p> <p>以下、ゲームとして面白いかどうかは未検証ですが、物理挙動のアピールポイントを作る例です。</p> <ul style="list-style-type: none"><li>• 狭い空間でボールを投げ合ったり、ボールが飛び跳ねたり、ボール同士の衝突判定をする</li><li>• ジャンプで高いところに飛び移ったり、バネでボールが飛び跳ねたりする</li><li>• 破壊オブジェクトにボールを当てて破壊するなど</li><li>• 斜面や移動床との当たり判定を行う</li></ul>
ゲームを成立させるための要素を増やす	<p>具体的にはどんなことをやりたいのか次第ですが、現状だとシンプルな内容ですので、もう少し要素を増やした方が良さそうです。</p> <p>また見た目が地味な印象を受けたので、地形を明るくしたり、派手な演出を入れてみると良いかもしれません</p>

### 5.10.12. 気になった点

項目	詳細
コリジョン抜けすることがある	タックルを壁に向かって撃ち続けると発生することがあるようです…。

項目	詳細
	
停止が発生することがある	手順は不明ですがハングアップすることがありました

### 5.10.13. コードレビュー

項目	詳細
長い衝突関数がある	<p>具体的な修正には理解を含めて時間がかかるため対応していませんが「ガード節を使う」によってネストを減らしたり、「共通部分を関数化する」といった対応で関数を短くできると思います。</p> <p>また「0.1f」というリテラル値も定数化しておいた方が良いかもしれません</p> <p><b>Enlighter: C++</b></p> <pre>//===== //ブロックの当たり判定(判定で押し戻す) //===== bool CBallGenerator::CollisionRect(void) {     D3DXVECTOR3 pos = GetPos();     D3DXVECTOR3 sizeMin = GetMinVtx();           //最小値     D3DXVECTOR3 sizeMax = GetMaxVtx();           //最大値      D3DXVECTOR3 posC = D3DXVECTOR3(0.0f, 0.0f, 0.0f);     D3DXVECTOR3 posOldC = D3DXVECTOR3(0.0f, 0.0f, 0.0f);     D3DXVECTOR3 sizeC = D3DXVECTOR3(0.0f, 0.0f, 0.0f);     D3DXVECTOR3 moveC = D3DXVECTOR3(0.0f, 0.0f, 0.0f);     D3DXVECTOR3 move = D3DXVECTOR3(0.0f, 0.0f, 0.0f); //押し戻す分      bool bLand = false;           //着地したかどうか     bool bCollision = false;       //当たり判定があったかどうか      //x     for (int nCntPrt = 0; nCntPrt &lt; PRIORITY_MAX; nCntPrt++)     {         CObject *pObject = CObject::GetTop(nCntPrt);         while ((pObject != nullptr))         {             if (pObject != nullptr)             {                 CObject::TYPE type = pObject-&gt;GetType(); //今回のオブジェクトのタイプ                  if (type == CObject::TYPE_PLAYER)                 { //プレイヤーだったら                     posC = pObject-&gt;GetPos();                 }             }         }     } }</pre>



項目	詳細
	<pre> posOldC = pObject-&gt;GetPosOld(); sizeC = pObject-&gt;GetSize(); moveC = pObject-&gt;GetMove();  D3DXVECTOR3 sizeOldMinC = D3DXVECTOR3(posOldC.x - sizeC.x, posOldC.y, posOldC.z - sizeC.z);           //キャラ最小値 D3DXVECTOR3 sizeOldMaxC = D3DXVECTOR3(posOldC.x + sizeC.x, posOldC.y + (sizeC.y * 2), posOldC.z + sizeC.z); //キャラ最大値  D3DXVECTOR3 sizeMinC = D3DXVECTOR3(posC.x - sizeC.x, posC.y, posC.z - sizeC.z);           //キャラ最小値 D3DXVECTOR3 sizeMaxC = D3DXVECTOR3(posC.x + sizeC.x, posC.y + (sizeC.y * 2), posC.z + sizeC.z); //キャラ最大値  if (pObject-&gt;GetJump() == false) {     if (sizeOldMaxC.x &lt;= pos.x + sizeMin.x         &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x         &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z + 0.1f         &amp;&amp; sizeMinC.z &lt; pos.z + sizeMax.z + 0.1f         &amp;&amp; ((sizeMaxC.y &gt;= pos.y + sizeMin.y + 0.1f             &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)                (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))     {         //ブロック西         move.x = (pos.x + sizeMin.x) - (sizeMaxC.x) - 0.1f;         bCollision = true;     }     else if (sizeOldMinC.x &gt;= pos.x + sizeMax.x         &amp;&amp; sizeMinC.x &lt;= pos.x + sizeMax.x         &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z + 0.1f         &amp;&amp; sizeMinC.z &lt; pos.z + sizeMax.z + 0.1f         &amp;&amp; ((sizeMaxC.y &gt;= pos.y + sizeMin.y + 0.1f             &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)                (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))     {         //ブロック左         move.x = (pos.x + sizeMax.x) - (sizeMinC.x) + 0.1f;         bCollision = true;     } } else {     if (sizeOldMaxC.x &lt;= pos.x + sizeMin.x         &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x         &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z + 0.1f         &amp;&amp; sizeMinC.z &lt; pos.z + sizeMax.z + 0.1f         &amp;&amp; ((sizeMaxC.y &gt;= pos.y + sizeMin.y + 0.1f             &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)                (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))     {         //ブロック西         move.x = (pos.x + sizeMin.x) - (sizeMaxC.x) - 0.1f;         bCollision = true;     }     else if (sizeOldMinC.x &gt;= pos.x + sizeMax.x         &amp;&amp; sizeMinC.x &lt;= pos.x + sizeMax.x         &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z + 0.1f         &amp;&amp; sizeMinC.z &lt; pos.z + sizeMax.z + 0.1f         &amp;&amp; ((sizeMaxC.y &gt;= pos.y + sizeMin.y + 0.1f             &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)                (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))     {         //ブロック左         move.x = (pos.x + sizeMax.x) - (sizeMinC.x) + 0.1f;         bCollision = true;     } } }  if (bCollision) {     pObject-&gt;SetPos(posC + move);     bCollision = false;     break; }  pObject = pObject-&gt;GetNext(); } else {     // (pObject == NULL) == Endまで行ったってことでこの優先度は終了     break; } }  }  //y for (int nCntPrt = 0; nCntPrt &lt; PRIORITY_MAX; nCntPrt++) {     CObject *pObject = CObject::GetTop(nCntPrt); </pre>

項目	詳細
	<pre> while ((pObject != nullptr)) {     if (pObject != nullptr)     {         CObject::TYPE type = pObject-&gt;GetType();          //今回のオブジェクトのタイプ          if (type == CObject::TYPE_PLAYER)         {             //プレイヤーだったら             posC = pObject-&gt;GetPos();             posOldC = pObject-&gt;GetPosOld();             sizeC = pObject-&gt;GetSize();             moveC = pObject-&gt;GetMove();              D3DXVECTOR3 sizeOldMinC = D3DXVECTOR3(posOldC.x - sizeC.x, posOldC.y, posOldC.z - sizeC.z);          //キャラ最小値             D3DXVECTOR3 sizeOldMaxC = D3DXVECTOR3(posOldC.x + sizeC.x, posOldC.y + (sizeC.y * 2), posOldC.z + sizeC.z);          //キャラ最大値              D3DXVECTOR3 sizeMinC = D3DXVECTOR3(posC.x - sizeC.x, posC.y, posC.z - sizeC.z);          //キャラ最小値             D3DXVECTOR3 sizeMaxC = D3DXVECTOR3(posC.x + sizeC.x, posC.y + (sizeC.y * 2), posC.z + sizeC.z);          //キャラ最大値              if (pObject-&gt;GetJump() == false)             {                 if (sizeOldMinC.y &gt;= pos.y + sizeMax.y                     &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y                     &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x + 0.1f                     &amp;&amp; sizeMinC.x &lt; pos.x + sizeMax.x + 0.1f                     &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z + 0.1f                     &amp;&amp; sizeMinC.z &lt; pos.z + sizeMax.z + 0.1f)                 {                     //ブロック上                     D3DXVECTOR3 Objmove = pObject-&gt;GetMove();                     pObject-&gt;SetMove(D3DXVECTOR3(Objmove.x, 0.0f, Objmove.z));                      move.y = (pos.y + sizeMax.y) - (sizeMinC.y) + 0.1f;                     pObject-&gt;SetJump(false);                     pObject-&gt;SetBoost(false);                     bCollision = true;                     bLand = true;                 }                 else if (sizeOldMaxC.y &lt;= pos.y + sizeMin.y                     &amp;&amp; sizeMaxC.y &gt;= pos.y + sizeMin.y                     &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x + 0.1f                     &amp;&amp; sizeMinC.x &lt; pos.x + sizeMax.x + 0.1f                     &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z + 0.1f                     &amp;&amp; sizeMinC.z &lt; pos.z + sizeMax.z + 0.1f)                 {                     //ブロック下                     D3DXVECTOR3 Objmove = pObject-&gt;GetMove();                     pObject-&gt;SetMove(D3DXVECTOR3(Objmove.x, 0.0f, Objmove.z));                      move.y = (pos.y + sizeMin.y) - (sizeMaxC.y) - 0.1f;                     bCollision = true;                 }             }         }         else         {             if (sizeOldMinC.y &gt;= pos.y + sizeMax.y                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y                 &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x + 0.1f                 &amp;&amp; sizeMinC.x &lt; pos.x + sizeMax.x + 0.1f                 &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z + 0.1f                 &amp;&amp; sizeMinC.z &lt; pos.z + sizeMax.z + 0.1f)             {                 //ブロック上                 D3DXVECTOR3 Objmove = pObject-&gt;GetMove();                 pObject-&gt;SetMove(D3DXVECTOR3(Objmove.x, 0.0f, Objmove.z));                  move.y = (pos.y + sizeMax.y) - (sizeMinC.y) + 0.1f;                 pObject-&gt;SetJump(false);                 pObject-&gt;SetBoost(false);                 bCollision = true;                 bLand = true;             }             else if (sizeOldMaxC.y &lt;= pos.y + sizeMin.y                 &amp;&amp; sizeMaxC.y &gt;= pos.y + sizeMin.y                 &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x + 0.1f                 &amp;&amp; sizeMinC.x &lt; pos.x + sizeMax.x + 0.1f                 &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z + 0.1f                 &amp;&amp; sizeMinC.z &lt; pos.z + sizeMax.z + 0.1f)             {                 //ブロック下                 D3DXVECTOR3 Objmove = pObject-&gt;GetMove();                 pObject-&gt;SetMove(D3DXVECTOR3(Objmove.x, 0.0f, Objmove.z));                  move.y = (pos.y + sizeMin.y) - (sizeMaxC.y) - 0.1f;                 bCollision = true;             }         }     } } </pre>

項目	詳細
	<pre>         if (bCollision)         {             pObject-&gt;SetPos(posC + move);             bCollision = false;             break;         }          pObject = pObject-&gt;GetNext();     }     else     {         // (pObject == NULL) == Endまで行っただってことでこの優先度は終了         break;     } }  //z for (int nCntPrt = 0; nCntPrt &lt; PRIORITY_MAX; nCntPrt++) {     CObject *pObject = CObject::GetTop(nCntPrt);      while ((pObject != nullptr))     {         if (pObject != nullptr)         {             CObject::TYPE type = pObject-&gt;GetType();          //今回のオブジェクトのタイプ              if (type == CObject::TYPE_PLAYER)             {                 //プレイヤーだったら                 posC = pObject-&gt;GetPos();                 posOldC = pObject-&gt;GetPosOld();                 sizeC = pObject-&gt;GetSize();                 moveC = pObject-&gt;GetMove();                  D3DXVECTOR3 sizeOldMinC = D3DXVECTOR3(posOldC.x - sizeC.x, posOldC.y, posOldC.z - sizeC.z);          //キャラ最小値                 D3DXVECTOR3 sizeOldMaxC = D3DXVECTOR3(posOldC.x + sizeC.x, posOldC.y + (sizeC.y * 2), posOldC.z + sizeC.z);          //キャラ最大値                  D3DXVECTOR3 sizeMinC = D3DXVECTOR3(posC.x - sizeC.x, posC.y, posC.z - sizeC.z);          //キャラ最小値                 D3DXVECTOR3 sizeMaxC = D3DXVECTOR3(posC.x + sizeC.x, posC.y + (sizeC.y * 2), posC.z + sizeC.z);          //キャラ最大値                  if (pObject-&gt;GetJump() == false)                 {                     if (sizeOldMaxC.z &lt;= pos.z + sizeMin.z                         &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z                         &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x + 0.1f                         &amp;&amp; sizeMinC.x &lt; pos.x + sizeMax.x + 0.1f                         &amp;&amp; ((sizeMaxC.y &gt;= pos.y + sizeMin.y + 0.1f                             &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)                                (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f                                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))                     {                         //ブロック北                         move.z = (pos.z + sizeMin.z) - (sizeMaxC.z) - 0.1f;                         bCollision = true;                     }                     else if (sizeOldMinC.z &gt;= pos.z + sizeMax.z                         &amp;&amp; sizeMinC.z &lt;= pos.z + sizeMax.z                         &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x + 0.1f                         &amp;&amp; sizeMinC.x &lt; pos.x + sizeMax.x + 0.1f                         &amp;&amp; ((sizeMaxC.y &gt;= pos.y + sizeMin.y + 0.1f                             &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)                                (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f                                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))                     {                         //ブロック南                         move.z = (pos.z + sizeMax.z) - (sizeMinC.z) + 0.1f;                         bCollision = true;                     }                 }             }             else             {                 {                     if (sizeOldMaxC.z &lt;= pos.z + sizeMin.z                         &amp;&amp; sizeMaxC.z &gt; pos.z + sizeMin.z                         &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x + 0.1f                         &amp;&amp; sizeMinC.x &lt; pos.x + sizeMax.x + 0.1f                         &amp;&amp; ((sizeMaxC.y &gt;= pos.y + sizeMin.y + 0.1f                             &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)                                (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f                                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))                     {                         //ブロック北                         move.z = (pos.z + sizeMin.z) - (sizeMaxC.z) - 0.1f;                         bCollision = true;                     }                     else if (sizeOldMinC.z &gt;= pos.z + sizeMax.z                         &amp;&amp; sizeMinC.z &lt;= pos.z + sizeMax.z                         &amp;&amp; sizeMaxC.x &gt; pos.x + sizeMin.x + 0.1f                         &amp;&amp; sizeMinC.x &lt; pos.x + sizeMax.x + 0.1f                         &amp;&amp; ((sizeMaxC.y &gt;= pos.y + sizeMin.y + 0.1f                             &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)                                (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f                                 &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))                     {                         //ブロック南                         move.z = (pos.z + sizeMax.z) - (sizeMinC.z) + 0.1f;                         bCollision = true;                     }                 }             }         }     } } </pre>

項目	詳細
	<pre>        &amp;&amp; sizeMaxC.y &lt;= pos.y + sizeMax.y - 0.1f)            (sizeMinC.y &gt;= pos.y + sizeMin.y + 0.1f         &amp;&amp; sizeMinC.y &lt;= pos.y + sizeMax.y - 0.1f)))     { //ブロック南         move.z = (pos.z + sizeMax.z) - (sizeMinC.z) + 0.1f;         bCollision = true;     } }  if (bCollision) {     pObject-&gt;SetPos(posC + move);     bCollision = false;     break; }  pObject = pObject-&gt;GetNext(); } else { // (pObject == NULL) == Endまで行ったってことでこの優先度は終了     break; } } }  return bLand; }</pre>
重複したコードがある	<p>ball_ganarator.cppの「<b>bool CBallGenerator::CollisionRect(void)</b>」とblock.cppの「<b>bool CBlock::CollisionRect(void)</b>」が似たような処理となっているので共通化した方が良いと思います。</p> <p>全体的に衝突周りのコードが整理されていない印象を受けました。</p>
newとdeleteが不用意に呼び出されている	<p>以下の処理で new と delete の配置が「バラバラの場所」で使われているためメモリリークを起こしやすい危険なコードとなっています。</p> <p>できれば new と delete の呼び出し回数を減らしたり、オブジェクト化してメモリ解放を安全に行えるようにしておくのが良いです。</p> <p><b>Enlighter: C++</b></p> <pre>//===== // ファイル読み込み (motion) //===== void CEnemy::ReadFile(void) {     char *pComp = new char[READ_PSIZE];     char *pFilepass[MAX_PARTS] = {};     D3DXVECTOR3 pos[MAX_PARTS];     D3DXVECTOR3 rot[MAX_PARTS];     int aParent[MAX_PARTS];     int nNumParts = 0;     D3DXVECTOR3 **ppPos = nullptr;     D3DXVECTOR3 **ppRot = nullptr;     int nNumKey = 0;     int nFrame = 0;     int nLoop = 0;      int nKeyCtr = 0;     int nKeySetCtr = 0;     int nMotionCtr = 0;     CMotion::INFO *info = new CMotion::INFO[16];      for (int nCntNull = 0; nCntNull &lt; MAX_PARTS; nCntNull++)     {         pos[nCntNull] = D3DXVECTOR3(0.0f, 0.0f, 0.0f);         rot[nCntNull] = D3DXVECTOR3(0.0f, 0.0f, 0.0f);         aParent[nCntNull] = -1;     }      FILE *pFile;</pre> <div><div>//ゴミ</div><div>//ファイルパス</div><div>//プリセット位置</div><div>//プリセット向き</div><div>//親モデルの有無</div><div>//パーツ総数</div><div>//位置</div><div>//向き</div><div>//キー数</div><div>//フレーム数</div><div>//ループ [ 0:しない / 1:する ]</div><div>//モーション読込時のキーカウンター</div><div>//モーション読込時のキーセットカウンター</div><div>//モーション数</div><div>//モーション情報</div></div>

項目	詳細
	<pre> pFile = fopen(MOTION_FILE, "r"); if (pFile != nullptr) {     do     {         fscanf(pFile, "%s", pComp);          if (strncmp(pComp, "#", 1) == 0)         {             // このあとコメント             fgets(pComp, READ_PSIZE, pFile);             continue;         }          if (strcmp(pComp, "NUM_MODEL") == 0)         {             //総数取得             fscanf(pFile, "%s %d", pComp, &amp;nNumParts);             m_nNumModel = nNumParts;         }         else if (strcmp(pComp, "MODEL_FILENAME") == 0)         {             //ファイル読み込み             for (int nCntCrt = 0; nCntCrt &lt; nNumParts; nCntCrt++)             {                 if (pFilepass[nCntCrt] == nullptr)                 {                     pFilepass[nCntCrt] = new char[128];                     fscanf(pFile, "%s %s", pComp, pFilepass[nCntCrt]);                     break;                 }             }         }         else if (strcmp(pComp, "CHARACTERSET") == 0)         {             //オフセット情報取得開始             int nCntSet = 0;              do             {                 fscanf(pFile, "%s", pComp);                  if (strcmp(pComp, "PARTSSET") == 0)                 {                     //パーツ情報取得開始                     while (TRUE)                     {                         fscanf(pFile, "%s", pComp);                          if (strcmp(pComp, "END_PARTSSET") == 0)                         {                             break;                         }                         else if (strcmp(pComp, "PARENT") == 0)                         {                             fscanf(pFile, "%s %d", pComp, &amp;aParent[nCntSet]);                             //aParent[nCntSet] = -1;                         }                         else if (strcmp(pComp, "POS") == 0)                         {                             fscanf(pFile, "%s %f %f %f", pComp, &amp;pos[nCntSet].x, &amp;pos[nCntSet].y, &amp;pos[nCntSet].z);                         }                         else if (strcmp(pComp, "ROT") == 0)                         {                             fscanf(pFile, "%s %f %f %f", pComp, &amp;rot[nCntSet].x, &amp;rot[nCntSet].y, &amp;rot[nCntSet].z);                         }                     }                      //取得終了で加算                     nCntSet++;                 }             } while (strcmp(pComp, "END_CHARACTERSET") != 0);         }         else if (strcmp(pComp, "MOTIONSET") == 0)         {             do             {                 //モーション情報を読む                 fscanf(pFile, "%s", pComp);                  if (strncmp(pComp, "#", 1) == 0)                 {                     // このあとコメント                     fgets(pComp, READ_PSIZE, pFile);                     continue;                 }                 else if (strcmp(pComp, "END_MOTIONSET") == 0)                 {                     nMotionCtr++;                 }             } while (strcmp(pComp, "END_MOTIONSET") != 0);         }     } while (strcmp(pComp, "END_MOTIONSET") != 0); } </pre>

項目	詳細
	<pre> nKeyCtr = 0; nKeySetCtr = 0;  if (ppPos != nullptr) {     for (int nCntMotKey = 0; nCntMotKey &lt; nNumKey; nCntMotKey++)     {         delete[] ppPos[nCntMotKey];     }      delete[] ppPos; }  if (ppRot != nullptr) {     for (int nCntMotKey = 0; nCntMotKey &lt; nNumKey; nCntMotKey++)     {         delete[] ppRot[nCntMotKey];     }      delete[] ppRot; }  break; } else if (strcmp(pComp, "LOOP") == 0) {     //ループ設定取得     fscanf(pFile, "%s %d", pComp, &amp;nLoop);      info[nMotionCtr].bLoop = (nLoop == 0 ? false : true); } else if (strcmp(pComp, "NUM_KEY") == 0) {     //キー数取得     fscanf(pFile, "%s %d", pComp, &amp;nNumKey);      info[nMotionCtr].nNumKey = nNumKey;      //モーション時に必要な数だけ位置情報を生成     //生成内容 : ppPos[キー数][パーツ数]     //生成内容 : ppRot[キー数][パーツ数]     ppPos = new D3DXVECTOR3*[nNumKey];     ppRot = new D3DXVECTOR3*[nNumKey];      for (int nCntMotKey = 0; nCntMotKey &lt; nNumKey; nCntMotKey++)     {         ppPos[nCntMotKey] = new D3DXVECTOR3[nNumParts];         ppRot[nCntMotKey] = new D3DXVECTOR3[nNumParts];     } } else if (strcmp(pComp, "KEYSET") == 0) {     //キーセット情報取得開始     do     {         fscanf(pFile, "%s", pComp);          if (strcmp(pComp, "#", 1) == 0)         {             // このあとコメント             fgets(pComp, READ_PSIZE, pFile);             continue;         }         else if (strcmp(pComp, "END_KEYSET") == 0)         {             //取得終了で加算             nKeySetCtr++;             nKeyCtr = 0;              break;         }         else if (strcmp(pComp, "FRAME") == 0)         {             //フレーム数取得             fscanf(pFile, "%s %d", pComp, &amp;nFrame);              info[nMotionCtr].aKeyInfo[nKeySetCtr].nFrame = nFrame;         }         else if (strcmp(pComp, "KEY") == 0)         {             //パーツ情報取得開始             while (TRUE)             {                 fscanf(pFile, "%s", pComp);                  if (strcmp(pComp, "#", 1) == 0)                 {                     // このあとコメント                     fgets(pComp, READ_PSIZE, pFile);                     continue;                 }                 else if (strcmp(pComp, "END_KEY") == 0) </pre>

項目	詳細
	<pre>         {             //取得終了で加算             nKeyCtr++;             break;         }         else if (strcmp(pComp, "POS") == 0)         {             fscanf(pFile, "%s %f %f %f", pComp, &amp;ppPos[nKeySetCtr][nKeyCtr].x, &amp;ppPos[nKeySetCtr][nKeyCtr].y, &amp;ppPos[nKeySetCtr][nKeyCtr].z);              info[nMotionCtr].aKeyInfo[nKeySetCtr].aKey[nKeyCtr].fPosX = ppPos[nKeySetCtr][nKeyCtr].x;             info[nMotionCtr].aKeyInfo[nKeySetCtr].aKey[nKeyCtr].fPosY = ppPos[nKeySetCtr][nKeyCtr].y;             info[nMotionCtr].aKeyInfo[nKeySetCtr].aKey[nKeyCtr].fPosZ = ppPos[nKeySetCtr][nKeyCtr].z;         }         else if (strcmp(pComp, "ROT") == 0)         {             fscanf(pFile, "%s %f %f %f", pComp, &amp;ppRot[nKeySetCtr][nKeyCtr].x, &amp;ppRot[nKeySetCtr][nKeyCtr].y, &amp;ppRot[nKeySetCtr][nKeyCtr].z);              info[nMotionCtr].aKeyInfo[nKeySetCtr].aKey[nKeyCtr].fRotX = ppRot[nKeySetCtr][nKeyCtr].x;             info[nMotionCtr].aKeyInfo[nKeySetCtr].aKey[nKeyCtr].fRotY = ppRot[nKeySetCtr][nKeyCtr].y;             info[nMotionCtr].aKeyInfo[nKeySetCtr].aKey[nKeyCtr].fRotZ = ppRot[nKeySetCtr][nKeyCtr].z;         }     } }      } while (strcmp(pComp, "END_KEYSET") != 0); } while (strcmp(pComp, "END_MOTIONSET") != 0); }      } while (strcmp(pComp, "END_SCRIPT") != 0);  fclose(pFile); } else {     //ファイル読込に失敗     return; }  //モデルの生成(全パーツ分) for (int nCntCrt = 0; nCntCrt &lt; nNumParts; nCntCrt++) {     m_apPart[nCntCrt] = CParts::Create(pFilepass[nCntCrt], pos[nCntCrt], rot[nCntCrt]); }  //親モデルの設定(全パーツ分) for (int nCntPrt = 0; nCntPrt &lt; nNumParts; nCntPrt++) {     if (aParent[nCntPrt] &lt;= -1)     {         m_apPart[nCntPrt]-&gt;SetParent(nullptr);     }     else     {         if (m_apPart[aParent[nCntPrt]] != nullptr)         {             m_apPart[nCntPrt]-&gt;SetParent(m_apPart[aParent[nCntPrt]]);         }     } }  //モーション情報設定 m_pMotion-&gt;SetInfo(info, nMotionCtr);  delete[] pComp;          //ゴミ delete[] info;  for (int nCntPass = 0; nCntPass &lt; 32; nCntPass++) {     if (pFilepass[nCntPass] != nullptr)     {         delete pFilepass[nCntPass];     } } } </pre>

項目	詳細
横に長くないようにする	<div>■修正前</div> <div>Enlighter: C++</div> <pre> for (int nCntInfo = 0; nCntInfo &lt; NUM_MOTION; nCntInfo++) {     m_aInfo[nCntInfo].bLoop = false;     m_aInfo[nCntInfo].nNumKey = 1;      for (int nCntKeyInfo = 0; nCntKeyInfo &lt; NUM_KEY; nCntKeyInfo++)     {         m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].nFrame = 0;          for (int nCntKey = 0; nCntKey &lt; MAX_PARTS; nCntKey++)         {             m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fPosX = 0.0f;             m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fPosY = 0.0f;             m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fPosZ = 0.0f;             m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fRotX = 0.0f;             m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fRotY = 0.0f;             m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fRotZ = 0.0f;         }     } } </pre> <div>■修正後</div> <div>Enlighter: C++</div> <pre> for (int nCntInfo = 0; nCntInfo &lt; NUM_MOTION; nCntInfo++) {     // ローカル変数にイれることでコードの圧迫感を減らす。     auto&amp; info = m_aInfo[nCntInfo];     info.bLoop = false;     info.nNumKey = 1;      for (int nCntKeyInfo = 0; nCntKeyInfo &lt; NUM_KEY; nCntKeyInfo++)     {         auto&amp; keyInfo = info.aKeyInfo[nCntKeyInfo];         keyInfo.nFrame = 0;          for (int nCntKey = 0; nCntKey &lt; MAX_PARTS; nCntKey++)         {             auto&amp; key = keyInfo.aKey[nCntKey];             key.fPosX = 0.0f;             key.fPosY = 0.0f;             key.fPosZ = 0.0f;             key.fRotX = 0.0f;             key.fRotY = 0.0f;             key.fRotZ = 0.0f;         }     } } </pre> <p>静的な配列の場合は、C++11の範囲for文 (Range-based for loops)も使えます</p> <div>Enlighter: C++</div> <pre> for (auto&amp; info : m_aInfo) {     info.bLoop = false;     info.nNumKey = 1;      for (auto&amp; keyInfo : info.aKeyInfo) {         keyInfo.nFrame = 0;          for (auto&amp; key : keyInfo.aKey) {             key.fPosX = 0.0f;             key.fPosY = 0.0f;             key.fPosZ = 0.0f;             key.fRotX = 0.0f; </pre>



項目	詳細
	<pre>        key.fRotY = 0.0f;         key.fRotZ = 0.0f;     } }</pre>
オブジェクト化してコードを減らす	<p>初期化コードが長くなってしまっている。</p> <p><b>Enlighter: generic</b></p> <pre>m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fPosX = 0.0f; m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fPosY = 0.0f; m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fPosZ = 0.0f; m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fRotX = 0.0f; m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fRotY = 0.0f; m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].fRotZ = 0.0f;</pre> <p>これは構造体の初期化関数がないため。（細かい部分ですが、<b>コメントが間違っている</b>のも良くないです…）</p> <p><b>Enlighter: C++</b></p> <pre>//===== // キーの構造体 //===== typedef struct {     float fPosX;           //位置X     float fPosY;           //位置Y     float fPosZ;           //位置Z     float fRotX;           //位置X（※コメントが間違っている）     float fRotY;           //位置Y     float fRotZ;           //位置Z }KEY;</pre> <p>以下のような構造体に置き換えるとコードが短くなります。</p> <p><b>Enlighter: C++</b></p> <pre>// 3次元ベクトル. struct Vec3 {     float x = 0.f;     float y = 0.f;     float z = 0.f;     void Init(float _x=0.f, float _y=0.f, float _z=0.f) {         x = _x;         y = _y;         z = _z;     } };  // キーの構造体. struct KEY {     Vec3 Pos = {}; // 位置.     Vec3 Rot = {}; // 回転. }</pre> <p><b>Enlighter: C++</b></p>

項目	詳細
	<pre>// 初期化. m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].Pos.Init(); // 座標. m_aInfo[nCntInfo].aKeyInfo[nCntKeyInfo].aKey[nCntKey].Rot.Init(); // 回転値.</pre>

## 6. 森川駿弥 (Blade Knight)



### 6.11. 概要

項目	説明
製作期間	<p>2024/1.11～2.16</p> <ul style="list-style-type: none"> <li>1.12作品の計画発表</li> <li>1.19プロト版提出</li> <li>1.26α版提出</li> <li>2.2β版提出</li> <li>2.16日マスター版提出</li> </ul> <p>ブラッシュアップ期間</p> <ul style="list-style-type: none"> <li>2024/4月上旬～6/12</li> </ul>
開発環境	Visual Studio2019、DirectX9
アピールポイント	<ul style="list-style-type: none"> <li>プレイヤーの親子関係とモーションを外部ファイルで管理</li> </ul>

項目	説明
	<ul style="list-style-type: none"> <li>• precompileを実装してビルド時間を短縮</li> <li>• パーティクルを関数ポインタで管理</li> <li>• managerをシングルトン化して処理負荷の軽減</li> </ul>
添削してほしいポイント	<ul style="list-style-type: none"> <li>• 操作性</li> <li>• 関数分けすべきところ</li> </ul>

## 6.12. 添削

### 6.12.14. 良いところ

- 軽快なプレイヤーの操作。複数の攻撃パターン
- 敵の派手な攻撃パターン

### 6.12.15. 気になった点

項目	詳細
ボスを倒すと停止することがあります	<p>自分の環境だけかもしれませんが…</p> 
フィールド外に移動できてしまうのはあまり良くないかもしれません	<p>ひとまず見えないコリジョンを置いて移動できないようにするのが良いと思います。</p>

6.12.16. アドバイス

項目	詳細
操作性（攻撃）	弱・強攻撃の違いを表現するときに、モーションに溜めや硬直時間を用意するなど。（単にクールタイムを用意しても良いかも）  攻撃の重みをうまく表現できると印象が良くなるかもしれません。
ボタン連打で勝ててしまう	強攻撃ボタン連打で突進すると勝ててしまうので、何らかの対処が必要です <ul style="list-style-type: none"><li>敵の無敵時間を作る</li><li>敵の攻撃パターンを増やす<ul style="list-style-type: none"><li>ミニオン（雑魚敵）を生成して攻撃パターンを増やす</li></ul></li></ul>
モーションの読み込み周りについて	モーションの管理方法をアピールポイントにするのであれば、モーション情報をオブジェクト化するなど技術力をアピールできる書き方にしたほうが良いかもしれません（※コードレビューで後述）
ゲームを成立させるための要素を増やす	具体的にはどんなことをやりたいのか次第ですが、現状だとシンプルな内容ですので、もう少し要素を増やした方が良さそうです

6.12.17. コードレビュー

項目	詳細
定数値であれば#defineよりもconstを使う	<div>Enlighter: C++</div> <pre>//===== //マクロ定義 //===== #define CAMERA_SPEED (1.5f) //カメラの移動速度 #define MOVEFAST (0.2f) //カメラの慣性 #define CAMR (25.0f) //視点の距離 #define CAMV_MOVE (0.03f) //視点の移動速度 #define CAM_DISTANCE (400.0f) //カメラとプレイヤーの距離 #define CAM_R_INERTIA (0.2f) //視点の慣性 #define CAM_V_INERTIA (0.2f) //視点の慣性</pre>
D3DXVECTOR3 はオペレーター演算子が使える（はず）	<div>■ 修正前</div> <div>Enlighter: C++</div> <pre>//位置を更新 m_posV.x += m_move.x; m_posV.y += m_move.y; m_posV.z += m_move.z;</pre>

項目	詳細
	<pre>m_posR.x += m_move.x; m_posR.y += m_move.y; m_posR.z += m_move.z;</pre> <p>■ 修正後</p> <p>Enlighter: C++</p> <pre>//位置を更新 m_posV += m_move; m_posR += m_move;</pre>
読み取り用のバッファの変数名とコメントが気になりました	<p>Enlighter: C++</p> <pre>char garbage[640];           // ゴミ格納用</pre> <p>fscanf() で読み取るためのテンポラリのバッファなので、もう少し適切な変数名とコメントにした方が良いです（tmpやbufferなど）</p>
fscanfの書き方が無限ループする可能性がある	<p>void CMotion::Load(std::string pfile)で、fscanf() でファイルを1行ずつ読み込みしていますが、終了条件を満たさなかったときに無限ループする可能性があります。そのため <b>EOF</b> マクロで終了判定したほうが良いです。</p> <p>Enlighter: C++</p> <pre>#define _CRT_SECURE_NO_WARNINGS #include &lt;stdio.h&gt;  int main() {     // ファイル読み込み.     FILE* pFile = fopen("hoge.txt", "r");      while(1) {         char buf[128] = {}; // バッファ.         int ret = fscanf(pFile, "%s", buf); // 1行ずつ読み込む.         printf("%s ret:%d\n", buf, ret);         if(ret == EOF) {             printf("[WARNING] 終了条件を乱さずにファイルの終端に達しました");             break; // 終端.         }         if(strcmp(buf, "end") == 0) {             break; // 終了条件.         }     }     // ファイルを閉じる.     fclose(pFile); }</pre> <p>またできれば、while(1) は終了条件に一致しない場合に無限ループする可能性があるので、避けたほうが良いです。</p>

項目	詳細
モーション読み込みをオブジェクトにした方が良くかもしれない	<p>CMotion::Load(std::string pfile) が長くなってしまっています。原因は「<b>モーションデータの読み込み</b>」と「<b>モーションの生成</b>」を<b>同時に行っているため</b>です。もしこの関数を短くしたいのであれば、「モーションデータの読み込み」を別モジュールに分離できるようにします。</p> <pre>Enlighter: C++  const MAX_MOTION_FILENAME = 128; // ファイルパスは128文字まで。 const MAX_MOTION_PARTS    = 128; // パーツの最大数 const MAX_MOTION_DATA      = 32; // 1ファイルあたりのモーション最大数。  // パーツ情報。 struct MOTION_PARTS {     int      Index; // 自身のインデックス。     int      ParentIdx; // 親のインデックス。     D3DXVECTOR3 Pos; // 座標。     D3DXVECTOR3 Rot; // 回転。 } // モーション全体のデータ。 struct MOTION_DATA {     char      File[MAX_MOTION_FILENAME] = {}; // 読み込むファイル名。     MOTION_PARTS PartsList[MAX_MOTION_PARTS] = {}; // パーツ情報。 };  // モーション読み込みクラス。 class MotionParser { public:     // モーションの読み込み。     void Parse(const char* pPath);     const MOTION_DATA&amp; GetResult();  private:     MOTION_DATA m_DataList[MAX_MOTION_DATA] = {}; // 読み込んだモーションデータ。 };</pre> <p>読み込み専用のクラス「<b>MotionParser</b>」を作成し、読み込みはそのクラスで行います。その結果を「<b>MOTION_DATA</b>」としてモデル側が取得してモデルの構築を行います。</p>
アクションに対して入力判定の関数を作成することで処理をシンプルにする	<p>■修正前</p> <pre>Enlighter: C++  //===== // 行動 //===== void CPlayer::Act(float fSpeed) {     ...      if (pInputKeyboard-&gt;GetPress(DIK_A) == true            pInputPad-&gt;GetLStickXPress(CInputPad::BUTTON_L_STICK, 0) &lt; 0)     { //Aが押された         if (pInputKeyboard-&gt;GetPress(DIK_W) == true                pInputPad-&gt;GetLStickYPress(CInputPad::BUTTON_L_STICK, 0) &gt; 0)         { //左上             ...         }     } }</pre> <p>■修正後</p> <pre>Enlighter: C++  // キーボードとパッドの移動ベクトルを取得する。 D3DXVECTOR2 Input::GetMoveDir() {</pre>

項目	詳細
	<pre> // キーボードの情報取得 CInputKeyboard* pInputKeyboard = CManager::GetInstance()-&gt;GetInputKeyboard();  // コントローラーの情報取得 CInputPad* pInputPad = CManager::GetInstance()-&gt;GetInputPad();  D3DXVECTOR2 ret = {};  if (pInputKeyboard-&gt;GetPress(DIK_A) == true        pInputPad-&gt;GetLStickXPress(CInputPad::BUTTON_L_STICK, 0) &lt; 0) {     ret.x = -1.f; // 左. } else if (pInputKeyboard-&gt;GetPress(DIK_D) == true        pInputPad-&gt;GetLStickXPress(CInputPad::BUTTON_L_STICK, 0) &gt; 0) {     ret.x = 1.f; // 右. }  if (pInputKeyboard-&gt;GetPress(DIK_W) == true        pInputPad-&gt;GetLStickYPress(CInputPad::BUTTON_L_STICK, 0) &gt; 0) {     ret.y = -1.f; // 上. } else if (pInputKeyboard-&gt;GetPress(DIK_S) == true        pInputPad-&gt;GetLStickYPress(CInputPad::BUTTON_L_STICK, 0) &lt; 0) {     ret.y = 1.f; // 下. }  // 正規化する. D3DXVec2Normalize(&amp;ret, &amp;ret);  return ret; } </pre>

## 7. 佐藤根詩音 (METEOR)



### 7.13. 概要

- 開発環境 ; DirectX9
- 使用言語 ; C++
- 制作期間 ; 2023年10月上旬～下旬(1か月)
- 制作人数 : 1人

- ジャンル：全方向スクロールアクションゲーム

項目	説明	
ゲームの説明	自分の分身である敵から逃げつつ、できるだけ死なずに星を集め、最高スコアを目指すゲームです。スコアの隠しボーナスがあるので、ぜひ探してみてください。	
ツールの説明	マップツールです。F1キーを押すと操作方法が表示されます。オブジェクトを選択して配置したり、削除したいオブジェクトにカーソルを合わせると、対象のオブジェクトを削除できます。保存すると、data内のTEXTフォルダにmap.txtとして保存されます。	
アピールポイント	プレイヤーの行動を真似する敵	敵に数秒前のプレイヤーの動きをさせて、まるで敵がプレイヤーの行動を真似して追いかけているかのようにしました。この処理を作るときにどのように作るか色々な方法を試したのですが、最終的にはプレイヤーの動きを保存しておき、それを数秒後に敵に反映させることにしました。 最初はどのように追いかけさせれば想像通りになるのか凄く悩みましたが、処理自体は簡単になり、想像通りプレイヤーの真似をしながら追いかけているように見せることが出来ました。
	マップエディタの作成	1か月の期間の中で一番時間がかかるのはマップ制作だと考え、マップエディタを制作しました。ツールの説明にも書いてあるのですが、設置だけでなく削除もでき、さらには外部ファイルに保存してそのデータをゲームの方に持ってくるとそのまま使用することができます。 これによりマップの修正が物凄く楽になり時間も短縮されるので、長いマップでもすぐに完成させることができました。

## 7.14. 添削

### 7.14.18. 良いところ

- しっかりとゲームとしての面白さが成立していました（Celesteの影響を受けている…？）
  - 多彩なギミックがあって楽しめました

### 7.14.19. 気になったところ

項目	説明
マップエディターツールが見つからない	ゲームの実行ファイルはありますが、ツールが見つけれませんでした…



項目

不要なファイルを提出データに含めない

説明

… 20240620\_吉田学園 > 06\_佐藤根詩音 > 実行ファイル > data > TEXT

🔍 ↺ 🗑

⬇ 並べ替え ▾ ≡ 表示 ▾ ⋮

名前	更新日時	種類	サ
<div>📄 map - コピー.txt</div>	2024/06/13 18:00	テキスト文書	
<div>📄 map.txt</div>	2024/06/13 18:00	テキスト文書	
<div>📄 motion.txt</div>	2024/06/13 18:00	テキスト文書	
<div>📄 motion_enemy.txt</div>	2024/06/13 18:00	テキスト文書	
<div>📄 motion_frog.txt</div>	2024/06/13 18:00	テキスト文書	

map - コピー.txt はおそらく不要なデータです

### 7.14.20. アドバイス

項目	説明
2Dゲームはアピールが弱くなりやすい	可能であれば3D空間を使用したゲームを作ってみると、技術力をアピールしやすくなります

### 7.14.21. コードレビュー

項目	詳細
定数は#defineではなくconstを使う	<div>Enlighter: C++</div> <pre>//マクロ定義 #define PRIORITY (3) //優先順位 #define POS (10.0f) //pos初期値 #define CURVE_RL (0.5f) //左右の角度 #define CURVE_UP (0.0f) //上の角度 #define CURVE_DOWN (1.0f) //下の角度 #define MOVE_Y (0.7f) //移動量 #define ADD_MOVE_Y (1.5f) //移動量+加算する数 #define JUMP_HEIGHT (10.0f) //ジャンプの高さ #define MAX_STR (128) //文字の最大数 #define FRONT_MOVE (0.6f) //手前の時の移動量 #define FRONT_DASH_MOVE (15.0f) //手前のダッシュ時の移動量 #define MAX_DASH (2) //ダッシュの最大数 #define STOP_MOVE (0.8f) //止まる判定の移動量 #define FILE_ENEMY "data\\TEXT\\motion_player.txt" //敵モデルのテキスト</pre> <div>Enlighter: C++</div> <pre>// enemy.h</pre>

項目	詳細
ファイル名のテーブルはC++ヘッダに変数宣言を書かなくても良い	<pre>//敵クラスの定義 class CEnemy : public CObject { private:     static char *m_apFileName[PARTS_MAX];           //ファイル名  // enemy.cpp  //静的メンバ変数宣言 int CEnemy::m_nNumAll = 0;                          //敵の総数 char *CEnemy::m_apFileName[PARTS_MAX] = {     "data\\MODEL\\enemy\\00_body.x",     "data\\MODEL\\enemy\\01_head.x",     "data\\MODEL\\enemy\\02_hair.x",     "data\\MODEL\\enemy\\03_LU_arm.x",     "data\\MODEL\\enemy\\04_LD_arm.x",     "data\\MODEL\\enemy\\05_L_hand.x",     "data\\MODEL\\enemy\\06_RU_arm.x",     "data\\MODEL\\enemy\\07_RD_arm.x",     "data\\MODEL\\enemy\\08_R_arm.x",     "data\\MODEL\\enemy\\09_waist.x",     "data\\MODEL\\enemy\\10_LU_leg.x",     "data\\MODEL\\enemy\\11_LD_leg.x",     "data\\MODEL\\enemy\\12_L_shoe.x",     "data\\MODEL\\enemy\\13_RU_leg.x",     "data\\MODEL\\enemy\\14_RD_leg.x",     "data\\MODEL\\enemy\\15_R_shoe.x", };</pre> <p>通常は CPPファイルのみに static 変数として記述します (もしくは無名名前空間)。それと const も忘れずにつけておきます。</p>
正しいコメントをつける	<p>以下のコードはおそらくSEの再生です</p> <p><b>Enlighter: generic</b></p> <pre>//BGM再生 pSound-&gt;Play(pSound-&gt;SOUND_LABEL_SE_DASH_AUTO);</pre>
プレイヤーの移動情報のコピーについて	<p><b>Enlighter: C++</b></p> <pre>//プレイヤーの情報代入 m_aSaveAction[m_nFrameCounter].pos = pPlayer-&gt;GetPosition();           //位置 m_aSaveAction[m_nFrameCounter].rot = pPlayer-&gt;GetRotation();           //向き m_aSaveAction[m_nFrameCounter].bMove = pPlayer-&gt;GetIsMove();           //移動判定 m_aSaveAction[m_nFrameCounter].bDash = pPlayer-&gt;GetIsDash();           //ダッシュ判定 m_aSaveAction[m_nFrameCounter].bDashAuto = pPlayer-&gt;GetIsDashAuto();   //自動ダッシュ判定 m_aSaveAction[m_nFrameCounter].bJump = pPlayer-&gt;GetIsJump();           //ジャンプ判定 m_aSaveAction[m_nFrameCounter].bLand = pPlayer-&gt;GetIsLand();           //着地判定</pre> <p>1つずつ情報をコピーしていますが、PlayerReplayDataのようなクラスを作成してプレイヤー情報をそこに保存し、敵はそれを調べてコピーする、といったように受け渡しを抽象化した方が良いです。</p>

## 8. 堀川萩大 (BAD REFLECT)



### 8.15. 概要

制作途中のため落ちてしまう場合があります。申し訳ございません。

### 8.16. 添削

#### 8.16.22. アドバイス

項目	説明
どんな要素を足していくか	<p>おそらくまだ要素が入り切っていない印象を受けたので、色々試してみるのが良いと思います。以下、要素の例です。</p> <ul style="list-style-type: none"><li>プレイヤーのアクション<ul style="list-style-type: none"><li>クールダウン（ダッシュ移動や遠距離攻撃に制限があるなど）</li><li>敵弾を跳ね返すと必殺技ゲージが溜まるなど</li></ul></li><li>敵の種類<ul style="list-style-type: none"><li>近接タイプ：プレイヤーに接近してくる、近接攻撃攻撃力が高い、ガードを使うなど</li><li>遠距離タイプ：プレイヤーと距離を取る（逃げる）、遠距離攻撃する</li><li>バフ系：他の敵の能力を強化する</li><li>味方を巻き込む：プレイヤー・敵問わず攻撃する</li><li>中ボス：耐久力が高い。他の敵を従えて行動する</li></ul></li><li>アイテム<ul style="list-style-type: none"><li>体力回復</li><li>敵を倒すと武器をドロップして、その武器が使える</li><li>火炎瓶で炎を発生させ、その地点を移動不可にする</li></ul></li><li>ギミック</li></ul>

項目	説明
	<ul style="list-style-type: none"><li>壁</li><li>破壊オブジェクト</li><li>投擲オブジェクト（投げられるなど）</li></ul> <p>似たジャンルの色々なゲームを真似して作ってみても良いと思います。</p>

### 8.16.23. コードレビュー

項目	説明								
プレイヤーと敵の共通化	3DPlayerと3DEnemyでいくつか似たような処理があったので、処理を共通化させると良いかもしれません								
ColiissionModBull() という関数名について	<p>弾との衝突を行う <b>ColiissionModBull()</b> ですが、命名にいくつか問題があります</p> <table><tr><th>問題点</th><th>説明</th></tr><tr><td>typoしている</td><td>Coliission は正確な綴りは "<b>Collision</b>" です。英語を使用する場合、Google検索などして綴りは間違えないようにしてください</td></tr><tr><td>関数は動詞始まりにする</td><td>Collision は名詞なので、"<b>Collide</b>" が適切です</td></tr><tr><td>不用意に省略語を使用しない</td><td>ModBull はおそらく ModelBullet の省略語と思われますが、「Mod」は "Modification" や 剰余を求める関数名でもあるので適切ではありません。また Bull は「牛」という意味でありこちらも適切ではありません</td></tr></table> <p>個人的には "<b>CollideBullet</b>" が良いかなと思います。</p>	問題点	説明	typoしている	Coliission は正確な綴りは " <b>Collision</b> " です。英語を使用する場合、Google検索などして綴りは間違えないようにしてください	関数は動詞始まりにする	Collision は名詞なので、" <b>Collide</b> " が適切です	不用意に省略語を使用しない	ModBull はおそらく ModelBullet の省略語と思われますが、「Mod」は "Modification" や 剰余を求める関数名でもあるので適切ではありません。また Bull は「牛」という意味でありこちらも適切ではありません
問題点	説明								
typoしている	Coliission は正確な綴りは " <b>Collision</b> " です。英語を使用する場合、Google検索などして綴りは間違えないようにしてください								
関数は動詞始まりにする	Collision は名詞なので、" <b>Collide</b> " が適切です								
不用意に省略語を使用しない	ModBull はおそらく ModelBullet の省略語と思われますが、「Mod」は "Modification" や 剰余を求める関数名でもあるので適切ではありません。また Bull は「牛」という意味でありこちらも適切ではありません								
ガード節を使う	■修正前								
適切にコメントを入れる	<p>Enlighter: C++</p> <pre>//敵の当たり判定 for (int nCntPri = 0; nCntPri &lt; PRIORITY_MAX; nCntPri++) {     for (int nCntObj = 0; nCntObj &lt; NUM_POLYGON; nCntObj++)     {         pObj = CObject::GetObject(nCntPri, nCntObj);         if (pObj != NULL)         {             CObject::TYPE type;             type = pObj-&gt;GetType();             if (type == CObject::TYPE_ENEMY)             {                 D3DXVECTOR3 E_Pos = pObj-&gt;GetPos();                 if (CEntity3D::ColiissionModBull(E_Pos, m_pos) == true)                 {                     pObj-&gt;nLife--;                 }             }         }     } }</pre>								

項目	説明
	<div><pre>if (pObj-&gt;nLife &lt;= 0) { }</pre></div> <div>■修正後</div> <div>Enlighter: C++</div> <div><pre>//敵の当たり判定 for (int nCntPri = 0; nCntPri &lt; PRIORITY_MAX; nCntPri++) {     for (int nCntObj = 0; nCntObj &lt; NUM_POLYGON; nCntObj++)     {         pObj = CObject::GetObject(nCntPri, nCntObj);         if (pObj == NULL) {             continue; // 無効なオブジェクト.         }         CObject::TYPE type = pObj-&gt;GetType();         if (type != CObject::TYPE_ENEMY) {             continue; // 敵でない.         }         D3DXVECTOR3 E_Pos = pObj-&gt;GetPos();         if (CEEnemy3D::ColissionModBull(E_Pos, m_pos) == false) {             continue; // 衝突していない.         }         pObj-&gt;nLife--; // HPを減らす         if (pObj-&gt;nLife &lt;= 0) {             // 死亡処理.         }     } }</pre></div>