改良案

===== ☒ Character 分割 実行順おすすめ（3コミットで見栄えが出る） =====

目標： 1ファイル1000行超の Character.cpp を段階的に分割し、責務を明確化。
Visual Studio プロジェクト（CMake 不使用）で、見た目を変えずにコードの「整理力」「設計力」をアピールできる構成にします。

☒ コミット1：VFX / Blink 抽出

- ☒ **CharacterVfx** に VFX 関数群を移設。
  - InitCSVEffect / SetChargingEffect / SetFullChargeEffect / SetAttackLocusEffect / SetHitEffect / SetFenceHitEffect
- ☒ **CharacterModelBlink** に DrawCharacterModel を移設。
  - 無敵中の点滅描画などの「表現」責務をまとめる。
- ☒ まずビジュアル責務を独立させることで、レビュー映え＆最初のインパクトが大きい！

☒ **Character（変更後）**

```
Character::Draw() → CharacterModelBlink::Draw()
Character::Update() → CharacterVfx::Update()
```

☒ 効果：視覚表現が明確に分離。300行以上軽量化。

☒ コミット2：Shadow / Air / Forward 抽出

- ● **CharacterShadow** に InitShadow / ShadowSet / ShadowDraw を移設。
- ☒ **CharacterAir** に CharacterGravity / SetJump を移設。
- ☒ **CharacterForward** に RotateVecFront / FrontVectorConfirm を移設。
- ☒ Character::Update / Draw は新モジュール呼び出しのみの薄い制御層へ。

☒ **Character（変更後）**

```
Update() → shadow.tick() → air.tick() → forward.tick()
Draw() → shadow.draw()
```

☒ 効果：ランタイムの共通処理が整理され、Facade 化が進行。

☒ コミット3：Movement / Rotate / Charge / Hit / Fence / Csv / Observer の順次抽出

- ☒ **CharacterMovement / Rotate / Charge**
  - 移動・回転・チャージ処理を担当。
- ☒ **CharacterHit / Fence**
  - 被弾／ノックバック／柵反射をまとめる。
- ☒ **CharacterCsvLoader**
  - CSV パラメータ読み込みを専用化。
- ☒ **CharacterObserver**
  - Add / RemoveObserver など監視者管理を独立。

☒ **Character（変更後）**

```
Update() → movement.tick() → rotate.tick() → charge.tick() → hit.tick() → fence.tick()
```

☒ 効果：Player / Enemy 側の共通処理が削ぎ落とされ、状態遷移と入力処理に集中できる。

☒ ファイル移行マップ（Visual Studio での構成）

> 各 `.cpp` を Visual Studio の「ソースファイル」フォルダに新規作成し、既存関数を下表のファイルに移動します。

移行元（Character.cpp内の関数）	移行先（新ファイル）
DrawCharacterModel	CharacterModelBlink.cpp / .h
DrawCharacterImGui	CharacterDebugPanel.cpp / .h
InitShadow / ShadowSet / ShadowDraw	CharacterShadow.cpp / .h
CharacterGravity / SetJump	CharacterAir.cpp / .h
RotateVecFront / FrontVectorConfirm	CharacterForward.cpp / .h
CharacterMove / CreateMoveVector / MoveConfirm / IsOutsideStage / 減速系関数群	CharacterMovement.cpp / .h
RotateDirectionVector / MoveRotateX / FastRotateX / RotateXStop	CharacterRotate.cpp / .h
Charging / ChargeRelease / ChargeReset / SetArrow / DrawArrow	CharacterCharge.cpp / .h
Reflect / KnockBack / InvincibilityTimeCalculation / etc.	CharacterHit.cpp / .h
GetWireNormal / FenceReflect / NotifyFenceHit	CharacterFence.cpp / .h
SetCSVStatus	CharacterCsvLoader.cpp / .h
AddObserver / RemoveObserver	CharacterObserver.cpp / .h

☒ 補足：

- `CharacterParams.h` を作成し、MoveParam / JumpParam / ShadowParam などの構造体を移動。
- `Character.h` では各モジュールを前方宣言して include を最小化。
- Visual Studio の「ソリューションエクスプローラ」で `Character` フォルダを右クリック → 「追加」→「新しい項目」で上記を順次作成。

☒ 仕上げ（README / BookStack反映時）

☒ Before / After の依存図

```
Before: Character.cpp（1000行超）
  ├── 表示（VFX/点滅/影）
  ├── 挙動（移動/回転/ジャンプ/チャージ）
  ├── 当たり（被弾/ノックバック/柵反射）
  ├── データ（CSV読み込み）
  └── 管理（Observer/ImGui）

After: Character Facade + 8モジュール
  ├── CharacterVfx
  ├── CharacterModelBlink
  ├── CharacterShadow
  ├── CharacterAir
  ├── CharacterMovement / Rotate / Charge
  ├── CharacterHit / Fence
  ├── CharacterCsvLoader
  └── CharacterObserver
```

☒ 責務表

モジュール	主な役割	代表関数
CharacterVfx	VFX 発火・制御	SetHitEffect() 他
CharacterModelBlink	無敵中点滅描画	Draw()
CharacterShadow	影生成・描画	InitShadow(), ShadowDraw()
CharacterAir	重力・ジャンプ	CharacterGravity(), SetJump()
CharacterMovement	地上移動・減速処理	CharacterMove(), MoveConfirm()
CharacterRotate	回転ユーティリティ	RotateDirectionVector() 他
CharacterCharge	チャージ・矢印描画	Charging(), ChargeRelease()
CharacterHit	被弾・ノックバック	Reflect(), KnockBack()
CharacterFence	柵反射処理	GetWireNormal(), FenceReflect()
CharacterCsvLoader	パラメータ読込	SetCSVStatus()
CharacterObserver	監視者管理	AddObserver(), RemoveObserver()

☒ 動作不変性の説明

- Update の呼び順を保持（`shadow → air → forward → movement → ...`）
- API 仕様を変更せず、Player / Enemy 側の呼び元はそのまま利用可能。
- 表現系（VFX / SE）は Assets / Vfx 経由に統一。

🐵 将来拡張例

- Enemy も CharacterMovement / Hit / Fence を共通化 → AI 部分だけ差分に。
- Player 側の OnCollision → CollisionCharacter → Reflect は CharacterHit に委譲し、**入力とステート遷移だけ**に集中可能。

☒ TIP:

- まず「VFX/Blink → Shadow/Air/Forward」から着手するのが最も安全。
  見た目を変えずに **300〜400行削減**、レビューにも強い第一歩！

✔仕上げ（READMEに載せると採点が伸びる）

- **Before/After の依存図**（`Character` 1000行 → Facade + 8モジュール）
- **責務表**（1行ずつ「点滅描画＝Blink」「重力/ジャンプ＝Air」…）
- **動作不変性の説明**（`Update` の呼び順を保持。VFX/SE は Assets/Vfx 経由に）
- **将来拡張例：** `Enemy` も `CharacterMovement/Hit/Fence` を共有 → AI 部分だけの差分で成立。
  ※ Player 側の `OnCollision → CollisionCharacter → Reflect` は `CharacterHit` の窓口に寄せ、Player 独自は**入力とステート遷移だけ**に整理可能。
  Player

☒ 新規 .h / .cpp 最小テンプレート一式（Visual Studio 追加用）

> そのまま貼り付けて新規ファイルとして保存できます。必要に応じて型名（`Transform` / `Vec3` など）を既存プロジェクトのものに合わせてください。

> 例）保存先：`Game/Actors/Character/` 配下

CharacterParams.h

```
#pragma once
// 既存の型に合わせて置換してください
struct Vec3 { float x{}, y{}, z{}; };

struct InitializeParam { /* TODO: 初期姿勢/速度など */ };
struct MoveParam       { float maxSpeed{6.0f}; float accel{40.0f}; float decel{50.0f}; };
struct RotateParam     { float yawSpeed{360.0f}; float pitchSpeed{0.0f}; };
struct JumpParam       { float power{8.5f}; float coyoteTime{0.08f}; float bufferTime{0.12f}; };
struct HitParam        { float iFrames{0.25f}; float knockback{6.0f}; };
struct FenceHitParam   { float reflectScale{1.0f}; float cooldown{0.15f}; };
struct ShadowParam     { float size{1.0f}; float offsetY{0.05f}; };
```

CharacterModelBlink.h

```
#pragma once
struct Transform; // 前方宣言（既存のTransform型に合わせてください）
class CharacterModelBlink {
public:
  void Draw(int modelHandle, const Transform& t) const; // 無敵点滅込みの描画
};
```

CharacterModelBlink.cpp

```
#include "CharacterModelBlink.h"
// #include 必要に応じて: マテリアル/タイマー/無敵状態参照 など
```

```
void CharacterModelBlink::Draw(int modelHandle, const Transform& t) const {
  // TODO: 旧 DrawCharacterModel() の中身を移設
  (void)modelHandle; (void)t;
}
```

CharacterVfx.h

```
#pragma once
class CharacterVfx {
public:
  void InitFromCsv(const char* path);
  void SetChargingEffect();
  void SetFullChargeEffect();
  void SetAttackLocusEffect();
  void SetHitEffect();
  void SetFenceHitEffect();
  void Tick(float dt);
};
```

CharacterVfx.cpp

```
#include "CharacterVfx.h"
void CharacterVfx::InitFromCsv(const char* ){}
void CharacterVfx::SetChargingEffect(){}
void CharacterVfx::SetFullChargeEffect(){}
void CharacterVfx::SetAttackLocusEffect(){}
void CharacterVfx::SetHitEffect(){}
void CharacterVfx::SetFenceHitEffect(){}
void CharacterVfx::Tick(float){ }
```

CharacterShadow.h

```
#pragma once
struct Transform;
class CharacterShadow {
public:
  void Init();
  void Tick(const Transform& t);
  void Draw() const;
};
```

CharacterShadow.cpp

```
#include "CharacterShadow.h"
void CharacterShadow::Init(){}
void CharacterShadow::Tick(const Transform&){ }
void CharacterShadow::Draw() const { }
```

CharacterAir.h

```
#pragma once
class CharacterAir {
public:
  void TickGravity(float dt);
  void RequestJump();
  bool IsJumpBuffered() const { return false; }
};
```

CharacterAir.cpp

```
#include "CharacterAir.h"
```

```
void CharacterAir::TickGravity(float){ }
void CharacterAir::RequestJump(){ }
```

CharacterForward.h

```
#pragma once
struct Vec3; struct Transform;
class CharacterForward {
public:
  void Tick(const Transform& t);
  const Vec3& Front() const { return front_; }
private:
  Vec3 front_{}; // 右手系/左手系は既存に合わせる
};
```

CharacterForward.cpp

```
#include "CharacterForward.h"
void CharacterForward::Tick(const Transform&){ /* TODO: 回転→前方更新 */ }
```

CharacterMovement.h

```
#pragma once
struct Vec3; struct Transform; struct MoveParam;
class CharacterMovement {
public:
  void Tick(float dt, const MoveParam& prm, Transform& t);
  void MoveConfirm(Transform& t);
  bool IsOutsideStage(const Transform& t) const; // 必要ならステージ参照を注入
  // 減速/加速系
  static void Deceleration(float& v, float amount, float dt);
  static void FrictionDeceleration(float& v, float amount, float dt);
  static void AccelerationStop(float& v, float threshold);
  static bool IsDashStop(float v, float threshold);
};
```

CharacterMovement.cpp

```
#include "CharacterMovement.h"
void CharacterMovement::Tick(float, const MoveParam&, Transform&){ }
void CharacterMovement::MoveConfirm(Transform&){ }
bool CharacterMovement::IsOutsideStage(const Transform&) const { return false; }
void CharacterMovement::Deceleration(float& v, float a, float dt){ v -= a*dt; }
void CharacterMovement::FrictionDeceleration(float& v, float a, float dt){ v -= a*dt; }
void CharacterMovement::AccelerationStop(float& v, float th){ if(fabsf(v)<th) v=0.0f; }
bool CharacterMovement::IsDashStop(float v, float th){ return fabsf(v)<=th; }
```

CharacterRotate.h

```
#pragma once
struct Transform; struct RotateParam;
class CharacterRotate {
public:
  void Tick(float dt, const RotateParam& prm, Transform& t);
  void RotateDirectionVector(Transform& t);
  void MoveRotateX(Transform& t);
  void MoveRotateXReverse(Transform& t);
  void FastRotateX(Transform& t);
  void FastRotateXReverse(Transform& t);
  void RotateXStop(Transform& t);
};
```

CharacterRotate.cpp

```
#include "CharacterRotate.h"
void CharacterRotate::Tick(float, const RotateParam&, Transform&){ }
void CharacterRotate::RotateDirectionVector(Transform&){ }
void CharacterRotate::MoveRotateX(Transform&){ }
void CharacterRotate::MoveRotateXReverse(Transform&){ }
void CharacterRotate::FastRotateX(Transform&){ }
void CharacterRotate::FastRotateXReverse(Transform&){ }
void CharacterRotate::RotateXStop(Transform&){ }
```

CharacterCharge.h

```
#pragma once
struct Transform;
class CharacterCharge {
public:
  void Charging(float dt);
  void ChargeRelease();
  void ChargeReset();
  void SetArrow();
  void DrawArrow(const Transform& t) const;
};
```

CharacterCharge.cpp

```
#include "CharacterCharge.h"
void CharacterCharge::Charging(float){ }
void CharacterCharge::ChargeRelease(){ }
void CharacterCharge::ChargeReset(){ }
void CharacterCharge::SetArrow(){ }
void CharacterCharge::DrawArrow(const Transform&) const { }
```

CharacterHit.h

```
#pragma once
struct Transform; struct HitParam;
class CharacterHit {
public:
  void Reflect(const Transform& hitNormal);
  void KnockBack(float power);
  bool IsKnockBackEnd() const { return true; }
  void KnockBackVelocityReset();
  void InvincibilityTimeCalculation(float dt, const HitParam& prm);
};
```

CharacterHit.cpp

```
#include "CharacterHit.h"
void CharacterHit::Reflect(const Transform&){ }
void CharacterHit::KnockBack(float){ }
void CharacterHit::KnockBackVelocityReset(){ }
void CharacterHit::InvincibilityTimeCalculation(float, const HitParam&){ }
```

CharacterFence.h

```
#pragma once
struct Transform; struct FenceHitParam;
class CharacterFence {
public:
  void GetWireNormal(/* args: 環境からの法線取得に必要な情報 */);
```

```
  void FenceReflect(const FenceHitParam& prm);
  void NotifyFenceHit();
};
```

CharacterFence.cpp

```
#include "CharacterFence.h"
void CharacterFence::GetWireNormal(){ }
void CharacterFence::FenceReflect(const FenceHitParam&){ }
void CharacterFence::NotifyFenceHit(){ }
```

CharacterCsvLoader.h

```
#pragma once
#include "CharacterParams.h"
class CharacterCsvLoader {
public:
  bool Load(const char* csvPath, InitializeParam& init, MoveParam& move,
            RotateParam& rot, JumpParam& jump, HitParam& hit,
            FenceHitParam& fence, ShadowParam& shadow);
};
```

CharacterCsvLoader.cpp

```
#include "CharacterCsvLoader.h"
bool CharacterCsvLoader::Load(const char*, InitializeParam&, MoveParam&, RotateParam&, JumpParam&, HitParam&, FenceHitParam&,
ShadowParam&){
  // TODO: 旧 SetCSVStatus() を分解して移設
  return true;
}
```

CharacterObserver.h

```
#pragma once
#include <vector>
class ICharacterObserver { public: virtual ~ICharacterObserver() = default; /* OnHit/OnDie など */ };
class CharacterObserver {
public:
  void AddObserver(ICharacterObserver* o);
  void RemoveObserver(ICharacterObserver* o);
private:
  std::vector<ICharacterObserver*> obs_;
};
```

CharacterObserver.cpp

```
#include "CharacterObserver.h"
#include <algorithm>
void CharacterObserver::AddObserver(ICharacterObserver* o){ obs_.push_back(o); }
void CharacterObserver::RemoveObserver(ICharacterObserver* o){
  obs_.erase(std::remove(obs_.begin(), obs_.end(), o), obs_.end());
}
```

CharacterAssets.h

```
#pragma once
class CharacterAssets {
public:
  void Init();            // SE / モデル / マテリアルのロード
  int  ArrowModel() const { return arrowModel_; }
private:
```

```
 int arrowModel_ = -1;    // 例：DxLib/目作Engineのハンドル
};
```

CharacterAssets.cpp

```
#include "CharacterAssets.h"
void CharacterAssets::Init(){ /* TODO: 旧 InitArrow()/SEロード等を移設 */ }
```

> 必要なら **各クラス間の依存を注入する小さな Facade（Character 本体）** の見本も追加できます。既存 `Character.h/.cpp` にメンバとしてこれらを持ち、`Update/Draw` で順に呼ぶだけの薄い制御層にしてください。

☒ Facade 見本：`Character` 本体（依存注入・薄い制御層）

> 既存のエンジンフック（Init/Update/FixedUpdate/Render/OnCollision 等）を想定。
> 依存は **コンストラクタ注入** or **Setter 注入**で受け取り、各モジュールへ渡します。

`Character.h`（Facade）

```
#pragma once
#include <memory>
#include <string>
#include "CharacterParams.h"
#include "CharacterModelBlink.h"
#include "CharacterVfx.h"
#include "CharacterShadow.h"
#include "CharacterAir.h"
#include "CharacterForward.h"
#include "CharacterMovement.h"
#include "CharacterRotate.h"
#include "CharacterCharge.h"
#include "CharacterHit.h"
#include "CharacterFence.h"
#include "CharacterCsvLoader.h"
#include "CharacterObserver.h"

// ---- Engine 側（前方宣言にして依存を最小化） ----
struct Transform;          // 既存の Transform 型
class IPhysics;            // レイ/衝突問合せ（エンジン提供）
class IStageQuery;         // 地形/柵/足場の問合せ
class IAudio;              // SE 再生命令
class ITime;               // deltaTime 提供（必要なら）

class Character /* : public GameObject 等 */ {
public:
  struct Deps { IPhysics* physics{}; IStageQuery* stage{}; IAudio* audio{}; ITime* time{}; };

  explicit Character(const Deps& deps);
  ~Character();

  // --- ライフサイクル（エンジンのフックから呼ばれる想定） ---
  bool Init(const char* csvPath);          // パラメータ読み込み + アセット初期化
  void Update();                           // 1フレーム更新（順序制御のみ）
  void FixedUpdate(float fixedDt);         // 物理刻みが分かれている場合に使用
  void Render() const;                     // 影 + モデル点滅描画など

  // --- 衝突・イベント ---
  void OnCollision(/* Engine の接触情報を薄く受ける */);
  void OnFenceHit();
  void OnTakeDamage(int dmg);              // 被弾イベント入口
```

```
  // --- 状態アクセス（UI 等から参照） ---
  const Transform& GetTransform() const { return *transform_; }
  int HP() const { return hp_; }

  // --- Optional: Setter 注入（コンストラクタ後に差し替えたい依存がある場合） ---
  void SetDeps(const Deps& d) { deps_ = d; }

private:
  // ---- 依存／データ ----
  Deps deps_{};
  Transform* transform_{};     // 位置/回転/スケール：エンジン所有を生ポインタで参照（または保持）

  InitializeParam init_{}; MoveParam move_{}; RotateParam rot_{}; JumpParam jump_{};
  HitParam hitp_{}; FenceHitParam fencep_{}; ShadowParam shadowp_{};
  int hp_{100};

  // ---- モジュール（Composition） ----
  CharacterModelBlink blink_;
  CharacterVfx        vfx_;
  CharacterShadow     shadow_;
  CharacterAir        air_;
  CharacterForward    forward_;
  CharacterMovement   movement_;
  CharacterRotate     rotate_;
  CharacterCharge     charge_;
  CharacterHit        hit_;
  CharacterFence      fence_;
  CharacterCsvLoader  csv_;
  CharacterObserver   observer_;

  // ---- 内部ユーティリティ ----
  void TickRuntime_();
  void DrawModel_() const;
};
```

Character.cpp（Facade）

```
#include "Character.h"
// 最小限の Engine ヘッダのみを include（Physics/Stage/Audio/Time/Transform 等）

Character::Character(const Deps& deps) : deps_(deps) {}
Character::~Character() = default;

bool Character::Init(const char* csvPath) {
  // 1) CSV/データローディング
  if (!csv_.Load(csvPath, init_, move_, rot_, jump_, hitp_, fencep_, shadowp_)) {
    return false;
  }

  // 2) Transform はエンジンから取得（生成/アタッチ方法は既存に合わせる）
  // transform_ = Engine::CreateTransform(init_.pos, init_.rot, ...);

  // 3) アセット/VFX/影などの初期化
  shadow_.Init();
  vfx_.InitFromCsv(csvPath); // 必要に応じて別CSV
  // Audio 初期化が必要なら deps_.audio を使う

  return true;
}

void Character::Update() {
  // 依存の取得（必要なら）：float dt = deps_.time ? deps_.time->Delta() : 1.0f/60.0f;
  TickRuntime_();
}

void Character::FixedUpdate(float fixedDt) {
  // 物理刻みが別のときに Movement/Hit などの一部をここで進める設計も可
```

```
  (void)fixedDt;
}

void Character::Render() const {
  // 影→モデル点滅描画の順
  if (transform_) {
    shadow_.Draw();
    DrawModel_();
  }
}

void Character::OnCollision() {
  // エンジンの接触情報から法線・相手属性などを得て、Hit/Fence へ振り分け
  // 例：fence_.FenceReflect(fencep_);  hit_.InvincibilityTimeCalculation(dt, hitp_);
}

void Character::OnFenceHit() {
  fence_.FenceReflect(fencep_);
  vfx_.SetFenceHitEffect();
}

void Character::OnTakeDamage(int dmg) {
  (void)dmg; // 必要なら HP 減算・ノックバック決定
  hit_.KnockBack(hitp_.knockback);
  vfx_.SetHitEffect();
}

void Character::TickRuntime_() {
  // ※ 呼び順は従来どおりに維持（動作不変性の鍵）
  // shadow → air → forward → movement → rotate → charge → hit → fence → vfx
  if (!transform_) return;

  // 例: air_ は重力/ジャンプの内部タイマを進行
  air_.TickGravity(/*dt*/ 0.016f);

  forward_.Tick(*transform_);
  movement_.Tick(/*dt*/ 0.016f, move_, *transform_);
  rotate_.Tick(/*dt*/ 0.016f, rot_, *transform_);
  charge_.Charging(/*dt*/ 0.016f);
  hit_.InvincibilityTimeCalculation(/*dt*/ 0.016f, hitp_);
  vfx_.Tick(/*dt*/ 0.016f);

  // 影の位置更新：必要なら Transform を渡す
  shadow_.Tick(*transform_);
}

void Character::DrawModel_() const {
  // モデルハンドルなどは Engine/Assets 管理の取得方式に合わせる
  const int modelHandle = /* Engine::GetModelHandle(...) */ 0;
  blink_.Draw(modelHandle, *transform_);
}
```

> 使い方（例）

```
Character::Deps deps{ physicsSys, stageQuery, audioSys, timeSys };
Character player(deps);
player.Init("Data/Character.csv");
// game loop:
player.Update();
player.Render();
```

ポイント

- **依存注入（Deps）**で Engine 側と疎結合化。単体テスト時にはモックを注入可能。
- **Update の呼び順を固定**して動作不変性を担保。差分は各モジュール内で閉じる。

- **Facade は“薄い制御のみ”**。ロジックや I/O をここに戻さない（肥大化の再発防止）。

Revision #2
★Created 10 October 2025 01:59:30 by youe2
Updated 13 June 2026 21:00:27 by youe2