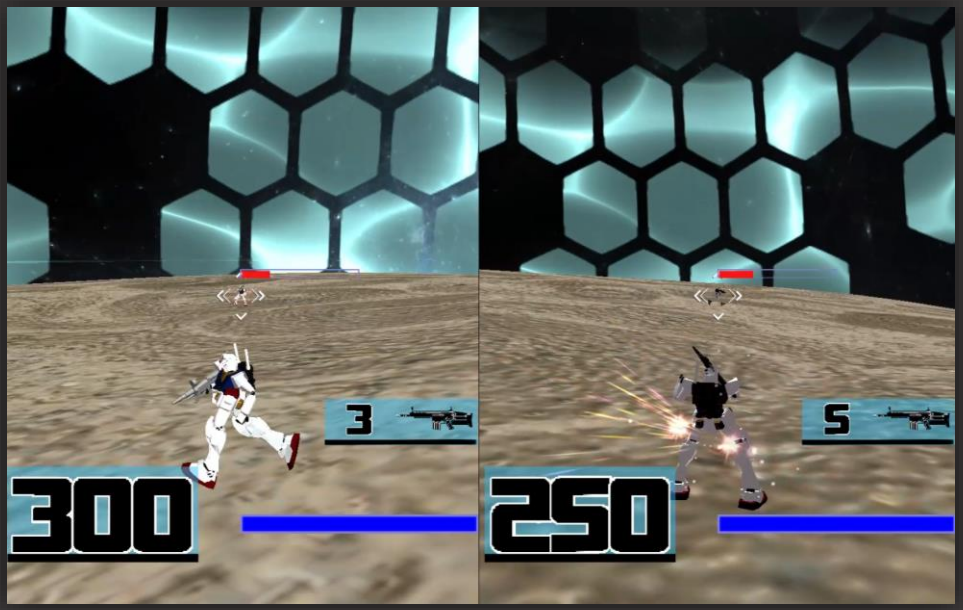
EXTREME_BOOST

個人製作

制作時期　　：4年前期
制作期間　　：４か月
使用言語　　：C++
使用ライブラリ：DXライブラリ
使用ツール　　：Visual Studio

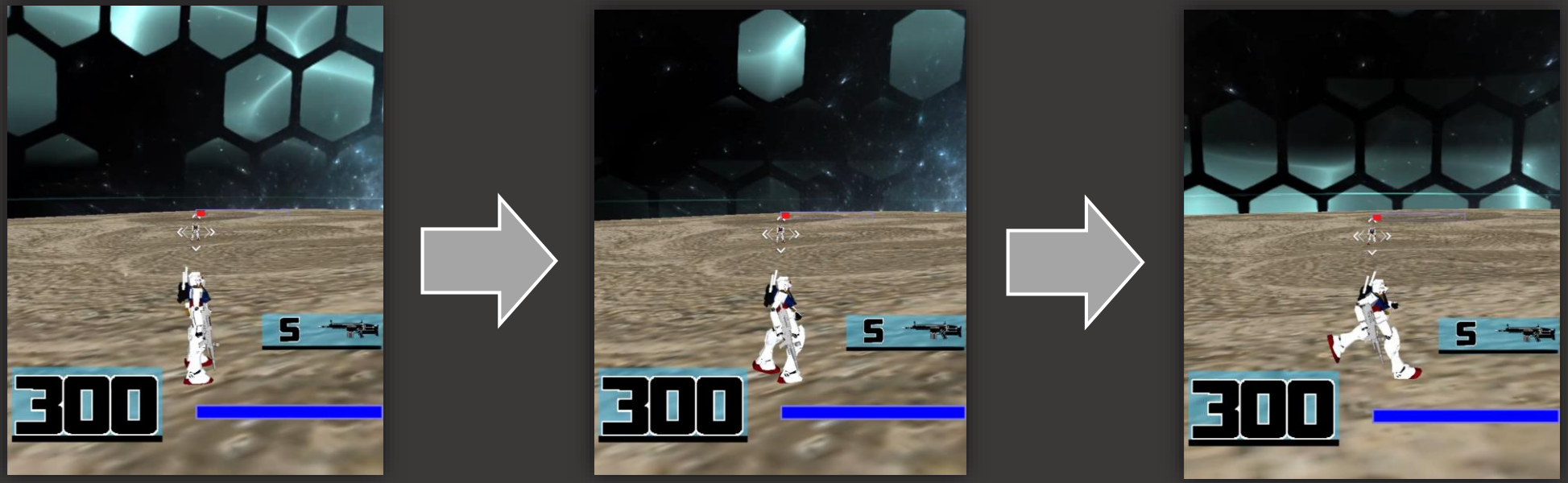
ずっと制作してみたいと思っていた、
3D対戦ゲームです。
ガンダムエクストリームバーサスシリーズを目標に、一対一のゲームしました。

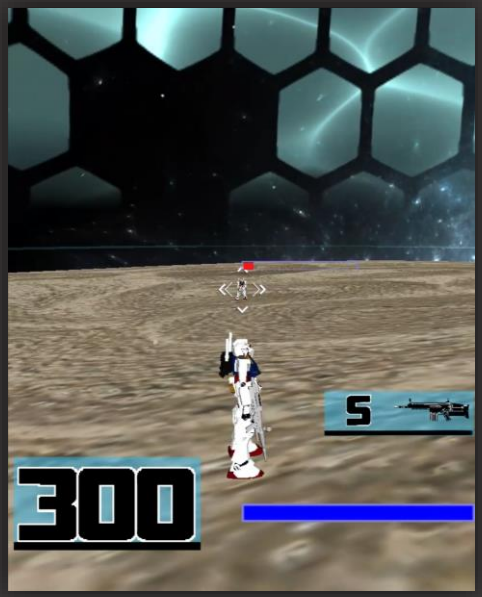


https://youtu.be/Ng4OAS0jCNE

アニメーションブレンド

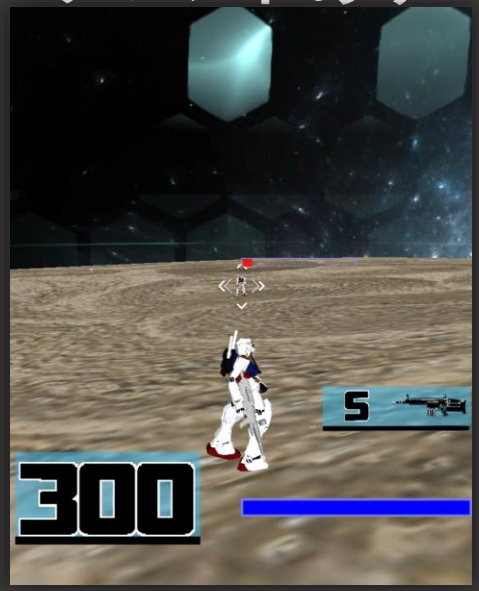
立ちモーションから、歩きアニメーションなどに移行するときに、
いきなり切り替わるのではなく、徐々に切り替えることで、
継ぎ接ぎ感のない滑らかなアニメーションにしました。





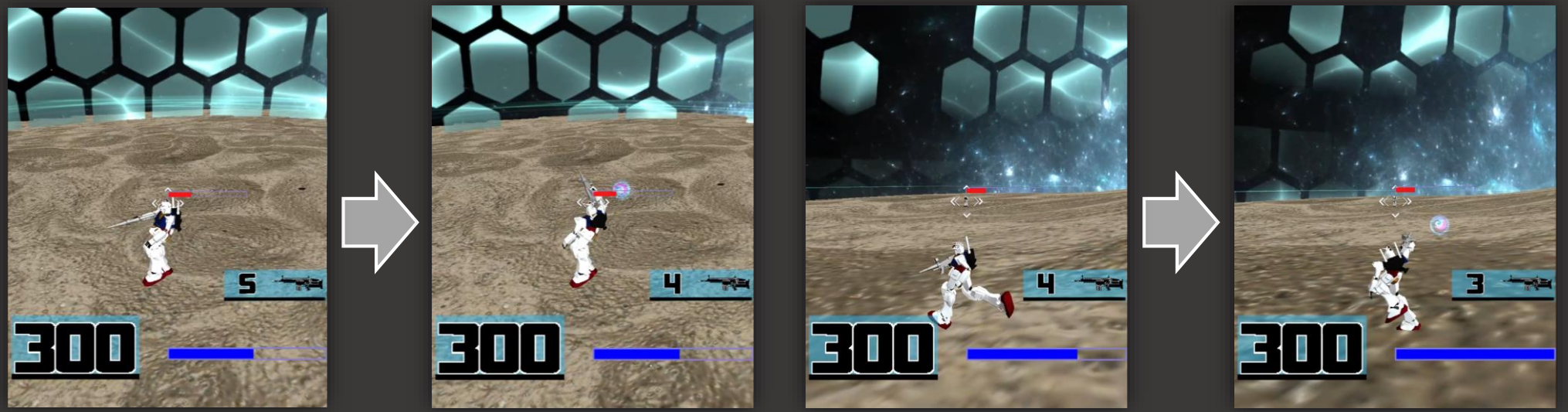
ブレンドなし

ブレンドあり



上下半身を分けたアニメーション

上半身と下半身のアニメーションを分けているので、射撃中は上半身のみ射撃アニメーションをして、下半身は歩きや立ち、ジャンプなどその時の移動アニメーションを行うことができます。



アニメーションの制御

上半身プレイ、下半身プレイ関数を使ってアニメーションを
切り替えます。そのとき、ブレンドフラグをtrueにして、
切り替え先アニメーションと現在アニメーションの情報を保存します。

上半身

```
void RobotAnimeController::UpperBodyPlay(int type, bool priority, bool isLoop ,
    bool isStop, float endStep , float startStep , bool isForce )
{
    //現在再生中のアニメーションとタイプが違い、優先再生でもなく、かつアニメーションタイプが-1でなければアニメーションを入れ替える
    if ( !(playAnim_[Body::UP].priority_) && playType_[Body::UP] != static_cast<STATE>(type) &&  type != -1)
    {
        auto newType = static_cast<STATE>(type);
        if (!blend_[Body::UP].blendFlag_)
        {
            //現在のアニメーションナンバーをブレンドアニメーションナンバーに格納
            blend_[Body::UP].attachNo_ = playAnim_[Body::UP].attachNo_;

            //現在のプレイタイプをブレンドアニメーションタイプに格納
            blend_[Body::UP].type_ = playType_[Body::UP];

            //新しいアニメーション情報を再生アニメーション情報に格納
            playAnim_[Body::UP] = animations_[type];

            //現在再生中タイプを新しいアニメーションに更新
            playType_[Body::UP] = newType;

            //新しいアニメーションがまだアタッチされていなければ
            if (!isAttach_[newType])
            {
                //アニメーションをアタッチして現在再生中アニメーションナンバーに格納
                playAnim_[Body::UP].attachNo_ = MV1AttachAnim(modelId_, 0, playAnim_[Body::UP].model_);
                MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::UP].attachNo_, 60, 0.0f, true);
                isAttach_[newType] = true;
                attachedTypeNum_[newType] = playAnim_[Body::UP].attachNo_;

                playAnim_[Body::UP].step_ = startStep;
            }
            else
            {
                //アタッチされていない場合、アタッチする
                playAnim_[Body::UP].attachNo_ = attachedTypeNum_[newType];
```

下半身

```
void RobotAnimeController::LowerBodyPlay(int type, bool priority , bool isLoop ,
    bool isStop , float endStep, float startStep, bool isForce )
{
    //アニメーションタイプが違い、かつアニメーションタイプが入っていたら
    if (!(playAnim_[Body::LOW].priority_) && playType_[Body::LOW] != static_cast<STATE>(type) && type != -1)
    {
        auto newType = static_cast<STATE>(type);
        if (!blend_[Body::LOW].blendFlag_)
        {
            blend_[Body::LOW].attachNo_ = playAnim_[Body::LOW].attachNo_;
            blend_[Body::LOW].type_ = playType_[Body::LOW];
            playAnim_[Body::LOW] = animations_[type];
            playType_[Body::LOW] = newType;

            //切り替わり先のアニメーションがアタッチ済みかどうかを判定
            if (!isAttach_[newType])
            {
                //アタッチされていない場合、アタッチする
                playAnim_[Body::LOW].attachNo_ = MV1AttachAnim(modelId_, 0, playAnim_[Body::LOW].model_);
                MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::LOW].attachNo_, 60, 0.0f, true);
                isAttach_[newType] = true;
                attachedTypeNum_[newType] = playAnim_[Body::LOW].attachNo_;

                playAnim_[Body::LOW].step_ = startStep;
            }
            else
            {
                //アタッチ済みの場合すでにあるアタッチ番号をそのまま使用
                playAnim_[Body::LOW].attachNo_ = attachedTypeNum_[newType] ;
                if (!startStep == 0.0f)
                {
                    playAnim_[Body::LOW].step_ = startStep;
                }
                else
                {
                    playAnim_[Body::LOW].step_ = MV1GetAttachAnimTime(modelId_, playAnim_[Body::LOW].attachNo
```

アニメーションの制御

ブレンドフラグがtrueの時は、アニメーション通常再生を止めて、
切り替え先アニメーションと、現在再生中のアニメーションの影響率を０～１
の間で上下させます。切り替え先のアニメーションを０から１に上げ、
現在のアニメーションを０になるまでに徐々に減らします。

ブレンドが完了したら、ブレンドフラグをfalseにして通常再生に戻します。

上半身

```
void RobotAnimeController::UpperBodyUpdate(void)
{
    //ブレンド中フラグが立っていれば
    if (blend_[Body::UP].blendFlag_)
    {
        //rateが1.0fでブレンド終了
        if (blend_[Body::UP].rate_ < 1.0f)
        {
            //ROOTと背骨0の、次アニメレートを増やす
            MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::UP].attachNo_, 60, blend_[Body::UP].rate_, false);
            MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::UP].attachNo_, 61, blend_[Body::UP].rate_, false);
            //背骨0の前アニメレートを徐々に減らす
            MV1SetAttachAnimBlendRateToFrame(modelId_, blend_[Body::UP].attachNo_, 61, 1.0f - blend_[Body::UP].rate_, false);
            //背骨1以下はすべて連動、前を減らし、次を増やす
            MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::UP].attachNo_, 62, blend_[Body::UP].rate_, true);
            MV1SetAttachAnimBlendRateToFrame(modelId_, blend_[Body::UP].attachNo_, 62, 1.0f - blend_[Body::UP].rate_, true);
            //ROOTは不動のため、現アニメの影響を受けない
            MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::UP].attachNo_, 60, 0.0f, false);

            blend_[Body::UP].rate_ = blend_[Body::UP].stepBlend_ / blendTime_;
            blend_[Body::UP].stepBlend_ += deltaTime_;
        }
        //ブレンドが終了したら
        else
        {
            //に関するパラメータを初期化
            blend_[Body::UP].blendFlag_ = false;
            blend_[Body::UP].stepBlend_ = 0.0f;
            blend_[Body::UP].rate_ = 1.0f;
        }
    }
    //ブレンド中じゃなければ通常再生
    else
    {
        //ストップフラグがfalse
        if (!playAnim_[Body::UP].isStop_)
```

下半身

```
void RobotAnimeController::LowerBodyUpdate(void)
{
    if (blend_[Body::LOW].blendFlag_)
    {
        if (blend_[Body::LOW].rate_ < 1.0f)
        {
            //ROOTの現アニメのrateを減らし、新しいアニメのrateを増やす
            MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::LOW].attachNo_, 60, blend_[Body::LOW].rate_, false);
            MV1SetAttachAnimBlendRateToFrame(modelId_, blend_[Body::LOW].attachNo_, 60, 1.0f - blend_[Body::LOW].rate_, false);
            //下半身 右足以下と左足以下連動
            MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::LOW].attachNo_, 115, blend_[Body::LOW].rate_, true);
            MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::LOW].attachNo_, 120, blend_[Body::LOW].rate_, true);
            MV1SetAttachAnimBlendRateToFrame(modelId_, blend_[Body::LOW].attachNo_, 115, 1.0f - blend_[Body::LOW].rate_, true);
            MV1SetAttachAnimBlendRateToFrame(modelId_, blend_[Body::LOW].attachNo_, 120, 1.0f - blend_[Body::LOW].rate_, true);

            blend_[Body::LOW].rate_ = blend_[Body::LOW].stepBlend_ / blendTime_;
            blend_[Body::LOW].stepBlend_ += deltaTime_;
        }
        else
        {
            blend_[Body::LOW].blendFlag_ = false;
            blend_[Body::LOW].stepBlend_ = 0.0f;
            blend_[Body::LOW].rate_ = 1.0f;
        }
    }
    else
    {
        if (playType_[Body::UP] == playType_[Body::LOW])
        {
            MV1SetAttachAnimBlendRateToFrame(modelId_, playAnim_[Body::LOW].attachNo_, 60, 1.0f, false);
        }
        //ストップフラグがfalse
        if (!playAnim_[Body::LOW].isStop_)
        {
            // 再生
            playAnim_[Body::LOW].step_ += (deltaTime_ * playAnim_[Body::LOW].speed_);
```

ステートパターンを使って状態遷移

ステートベースクラスを継承した、ステートクラスを複数作成して、
そこでそのステート固有の動きを、更新していきます。

```
void Player::UpdateBattleMode(void)
{
    //カメラに敵の座標を渡す
    camera_->SetTargetPos(*enemyPos_);

    //ブーストゲージ回復
    RecoverBoostGauge();

    //現在の敵の状態を調べる
    EnemyState();

    //現在のステートのアップデート
    state_->Update();

    //無敵時間があればそれを減らしていく
    if (!IsSafeTimeSufficient())
    {
        CountSafeTime(deltaTime_ * 50.0f);
    }

    //行動不能時間計測
    CountCombatStanTime();

    //敵との距離に応じてホーミングの有無を決める
    Range();
```

プレイヤーのアップデート関数では、
ステートのアップデートを呼ぶ

ステートパターンを使って状態遷移

新しい動きを実装したいときもステートクラスを増やすだけで、
プレイヤー側のアップデートなどに触れずに実装ができるので、
不具合が出たときの問題個所の特定などもしやすくなり、
より管理をしやすくなりました。

ダメージステート


```
DamageState::DamageState(Player& player) :player_(player)
{
    //ダメージアニメーションを再生
    player_.PlayAnim(static_cast<int>(Player::STATE::DAMAGE), true, true);
    //現在のステート情報を保存
    player_.pState_ = Player::STATE::DAMAGE;
    //前後左右移動をストップ
    player_.MoveStop();
    //上下の移動をストップ
    player_.JumpStop();
    count_ = 0.0f;
    //無敵時間を設定
    player_.SetSafeTime(Player::SMALL_SAFE_TIME);
    //ビームサーベルの当たり判定を非アクティブにする
    player_.GetBeamSaber().GetSaber().InActivate();
    //ビームサーベルの描画を非アクティブ状態にする
    player_.GetBeamSaber().InActivate();
    //ビームライフルをアクティブ状態にする
    player_.GetBeamRifle().Activate();
}

void DamageState::Update()
{
    //現在再生中アニメーションがループ再生完了したら
    if (player_.IsAnimEnded())
    {
        player_.pState_ = Player::STATE::IDLE;
        //アイドルアニメーションを再生
        player_.PlayAnim(static_cast<int>(Player::STATE::IDLE), false, true);
        //アイドル状態に移行
        player_.ChangeState(std::make_unique<IdleState>(player_));
        return;
    }
```

ブーストステート


```
void BoostState::Update()
{
    //ブースト時間が終わっているか判定
    if (time_> MAX_BOOST_TIME)
    {
        //ジャンプボタンを押していたらジャンプ状態に移行
        if (player_.GetInput().IsPrassed("jump"))
        {
            player_.ChangeState(std::make_unique<BoostDashState>(player_));
            return;
        }
        //スティックを触っていたらムーブ状態に移行
        if (!(player_.GetInput().isStickTilted(Input::STICK_LR::L)))
        {
            player_.ChangeState(std::make_unique<MoveState>(player_));
            return;
        }
        //スティックを触ってなければアイドル状態に移行
        else
        {
            player_.ChangeState(std::make_unique<IdleState>(player_));
            return;
        }
    }
    //ブースト行動開数
    player_.Boost();
    //上下の移動を止める
    player_.JumpStop();
    //射撃
    player_.Shot();
    //ブースト時間加算
    time_+=(player_.GetDeltaTime()*60.0f);
}
```