

ゲームプログラミング改

○評価要件

- ☒ ステートの切り替え
- ☒ アニメーション制御

○概要

今回はプレイヤーの行動制御を実装します。

前回は 3D モデルに対してアニメーションを再生できるようになりました。

プレイヤーはゲームパッドの入力にあわせて「移動」「ジャンプ」などの行動をしますが、見た目に説得力を持たせるため、状態に合わせてアニメーションを切り替えます。

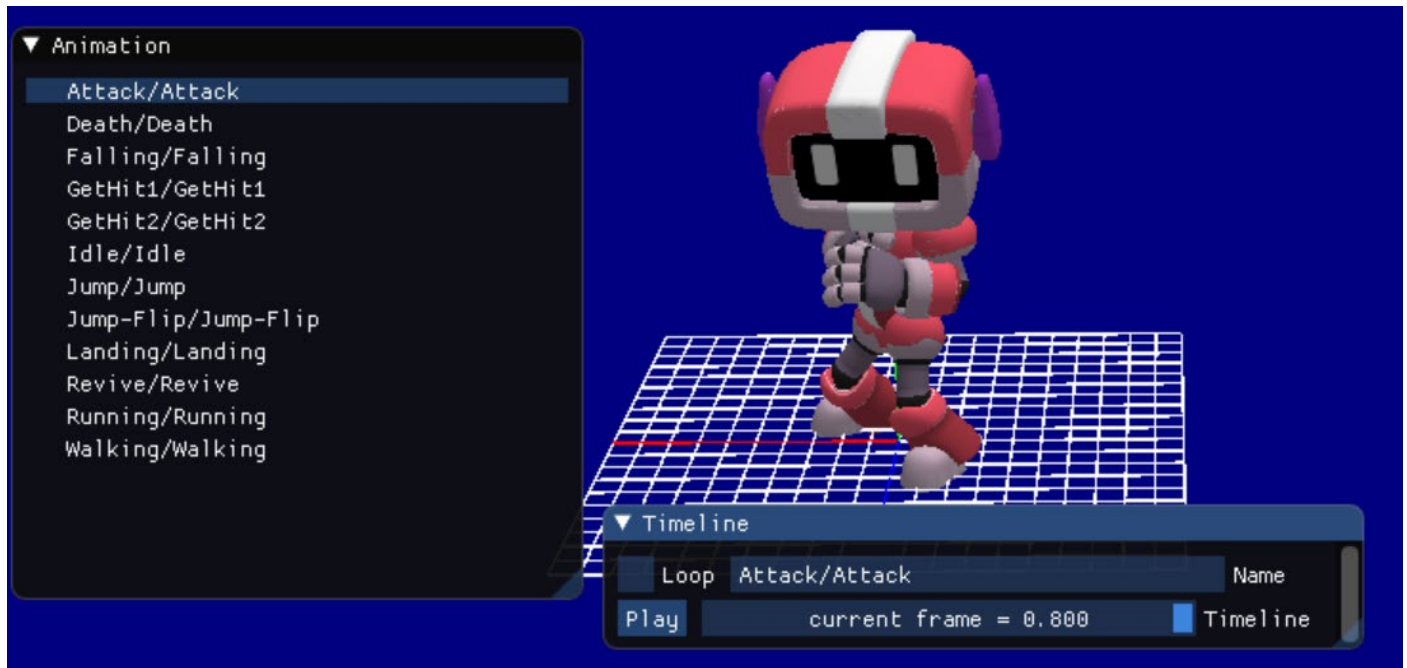
今回はプレイヤーに状態（ステート）情報を持たせ、状態（ステート）が遷移する時にアニメーションを切り替えるプログラムを実装していきます。

ゲームプログラミング改

○アニメーションの確認

これから状態毎にアニメーションを切り替える制御をするわけですが、モデルデータに入っているアニメーションデータの確認をしておきましょう。

モデルエディタを開き、プレイヤーのモデルファイルを開き、アニメーションを確認しましょう。



アニメーションデータは上図の「Animation」ウインドウに列挙されているアニメーションが上から順に並んで入っています。

このアニメーションデータをアクセスしやすいようにしましょう。

Player.h

```
---省略---  
  
// プレイヤー  
class Player : public Character  
{  
    ---省略---  
  
private:  
    // アニメーション  
    enum Animation  
    {  
        Anim_Attack,  
        Anim_Death,  
        Anim_Falling,  
        Anim_GetHit1,  
        Anim_GetHit2,  
        Anim_Idle,  
        Anim_Jump,  
        Anim_Jump_Flip,
```

ゲームプログラミング改

```
    Anim_Landing,  
    Anim_Revive,  
    Anim_Running,  
    Anim_Walking  
};  
  
---省略---  
};
```

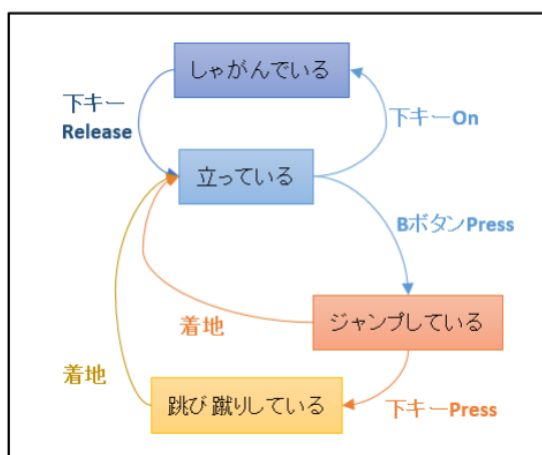
Player.cpp

```
---省略---  
  
// 更新処理  
void Player::Update(float elapsedTime)  
{  
    // Bボタン押下でアニメーション再生  
    GamePad& gamePad = Input::Instance().GetGamePad();  
    if (gamePad.GetButtonDown() & GamePad::BTN_B)  
    {  
        model->PlayAnimation(0, false);  
        model->PlayAnimation(Anim_Running, true);  
    }  
    ---省略---  
}  
---省略---
```

Enum のアニメーション番号を設定し、正しいアニメーションが再生されているか確認しましょう。

○状態（ステート）について

状態（ステート）とは「待機」や「移動」などプログラムの動作状態を表すもので、この状態を切り替えるような構造をステートマシンと呼びます。



ステート毎にどのステートに遷移できるかの条件が存在する。

今回は switch 文での簡易的なステートマシンを実装し、アニメーションを制御します。「待機」「移動」「ジャンプ」「着地」の4つのステートを制御できるようにしましょう。「ステート毎の更新処理」「指定のステートへの遷移処理」の2つを実装していきます。

ゲームプログラミング改

○待機ステート

待機ステートは様々な行動に遷移する基礎となるステートです。

まずは今までに実装した「移動」「ジャンプ」「弾丸射出」をできるように実装しましょう。

Player.h

```
---省略---

// プレイヤー
class Player : public Character
{
public:
    ---省略---

private:
    ---省略---

    // 待機ステートへ遷移
    void TransitionIdleState();

    // 待機ステート更新処理
    void UpdateIdleState(float elapsedTime);

private:
    // ステート
    enum class State
    {
        Idle
    };

private:
    ---省略---
    State state = State::Idle;
};
```

Player.cpp

```
---省略---

// コンストラクタ
Player::Player()
{
    ---省略---

    // 待機ステートへ遷移
    TransitionIdleState();
}

// 更新処理
void Player::Update(float elapsedTime)
{
    // Bボタン押下でワンショットアニメーション再生
    GamePad& gamePad = Input::Instance().GetGamePad();
    if (gamePad.GetButtonDown() & GamePad::BTN_B)
```

ゲームプログラミング改

```
{
    model->PlayAnimation(0, false);
}
// ワンショットアニメーション再生が終わったらループアニメーション再生
if (!model->IsPlayAnimation())
{
    model->PlayAnimation(1, true);
}

// 移動入力処理
InputMove(elapsedTime);

// ジャンプ入力処理
InputJump();

// 弾丸入力処理
InputProjectile();

// ステート毎の処理
switch (state)
{
{
case State::Idle:
    UpdateIdleState(elapsedTime);
    break;
}
}

// 速力更新処理
---省略---
}

---省略---

// 待機ステートへ遷移
void Player::TransitionIdleState()
{
    state = State::Idle;

    // 待機アニメーション再生
    model->PlayAnimation(Anim_Idle, true);
}

// 待機ステート更新処理
void Player::UpdateIdleState(float elapsedTime)
{
    // 移動入力処理
    InputMove(elapsedTime);

    // ジャンプ入力処理
    InputJump();

    // 弾丸入力処理
    InputProjectile();
}
```

ゲームプログラミング改

この時点で実行確認をしてみましょう。

待機アニメーションが再生され、今までと同じ操作ができていれば OK です。

ただ、今のままでは待機アニメーションのまま移動したり、ジャンプしたりします。

これから、ステートが遷移する際にアニメーションを切り替え、ステート毎にできる行動を実装していきます。

○移動ステート

移動ステートは移動中の状態での行動処理を実装します。

まずは待機ステートから移動ステートへ切り替える処理を実装しましょう。

Player.h

```
---省略---

// プレイヤー
class Player : public Character
{
public:
    ---省略---

private:
    ---省略---

    // 移動入力処理
    void InputMove(float elapsedTime);
    bool InputMove(float elapsedTime);

    ---省略---

    // 移動ステートへ遷移
    void TransitionMoveState();

    // 移動ステート更新処理
    void UpdateMoveState(float elapsedTime);

private:
    ---省略---

    // ステート
    enum class State
    {
        ---省略---
        Move
    };

    ---省略---
};
```

入力されたときに true を返すようにする。

ゲームプログラミング改

Player.cpp

---省略---

// 更新処理

void Player::Update(float elapsedTime)

{

// ステート毎の処理

switch (state)

{

---省略---

case State::Move:

UpdateMoveState(elapsedTime);

break;

}

---省略---

}

---省略---

// 移動入力処理

bool Player::InputMove(float elapsedTime)

{

---省略---

// 進行ベクトルがゼロベクトルでない場合は入力された

return

}

---省略---

// 待機ステート更新処理

void Player::UpdateIdleState(float elapsedTime)

{

// 移動入力処理

移動入力されたら移動ステートへ遷移

---省略---

}

// 移動ステートへ遷移

void Player::TransitionMoveState()

{

state = State::Move;

// 走りアニメーション再生

model->PlayAnimation(Anim_Running, true);

}



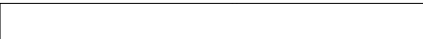
// 移動ステート更新処理

void Player::UpdateMoveState(float elapsedTime)

{

}

ゲームプログラミング改

```
{  
    // 移動入力処理  
      
    // ジャンプ入力処理  
      
    // 弾丸入力処理  
      
}
```

移動入力がない場合は待機状態へ遷移

実行確認をしてみましょう。

移動入力によってアニメーションが切り替わり、状態処理が正常にできていれば OK です。
このような感じでジャンプ状態や着地状態を実装していきましょう。

○ジャンプ&着地状態

ジャンプ状態はジャンプ中の状態での行動処理を実装します。

ジャンプ中もできることは今までと同じですがアニメーションの制御が異なります。

ジャンプに関するアニメーションは「ジャンプ開始」「落下中」「フリップ」「着地」の4つのアニメーションが用意されています。

モデルエディタで「Jump」「Falling」「Jump-Flip」「Landing」のアニメーションを確認し、自然なアニメーション制御を実装しましょう。

Player.h

```
---省略---  
  
// プレイヤー  
class Player : public Character  
{  
public:  
    ---省略---  
  
private:  
    ---省略---  
  
    // ジャンプ入力処理  
    void InputJump();  
    bool InputJump();  
  
    ---省略---  
  
    // ジャンプ状態へ遷移  
    void TransitionJumpState();  
}
```


ゲームプログラミング改

```
// ジャンプステート更新処理
void UpdateJumpState(float elapsedTime);

// 着地ステートへ遷移
void TransitionLandState();

// 着地ステート更新処理
void UpdateLandState(float elapsedTime);

---省略---

private:
    ---省略---

    // ステート
    enum class State
    {
        ---省略---
        Jump,
        Land
    };

    ---省略---
};
```

Player.cpp

```
---省略---

// 更新処理
void Player::Update(float elapsedTime)
{
    // ステート毎の処理
    switch (state)
    {
        ---省略---

        case State::Jump:
            UpdateJumpState(elapsedTime);
            break;

        case State::Land:
            UpdateLandState(elapsedTime);
            break;
    }

    ---省略---
}

---省略---

// 着地した時に呼ばれる
void Player::OnLanding()
{
    ---省略---
```

ゲームプログラミング改

```
// 下方方向の速力が一定以上なら着地ステートへ

}

---省略---

// ジャンプ入力処理
bool Player::InputJump()
{
    ---省略---
    if (gamePad.GetButtonDown() & GamePad::BTN_A)
    {
        // ジャンプ回数制限
        if (---省略---)
        {
            // ジャンプ
            ---省略---

            // ジャンプ入力した
            return true;
        }
    }

    return false;
}

---省略---

// 待機ステート更新処理
void Player::UpdateIdleState(float elapsedTime)
{
    ---省略---

    // ジャンプ入力処理
    

    ---省略---
}

---省略---

// 移動ステート更新処理
void Player::UpdateMoveState(float elapsedTime)
{
    ---省略---
```

ゲームプログラミング改

```
// ジャンプ入力処理
```



```
---省略---
```

```
}
```

```
// ジャンプステートへ遷移
```

```
void Player::TransitionJumpState()
```

```
{
```

```
    state = State::Jump;
```

```
    // ジャンプアニメーション再生
```

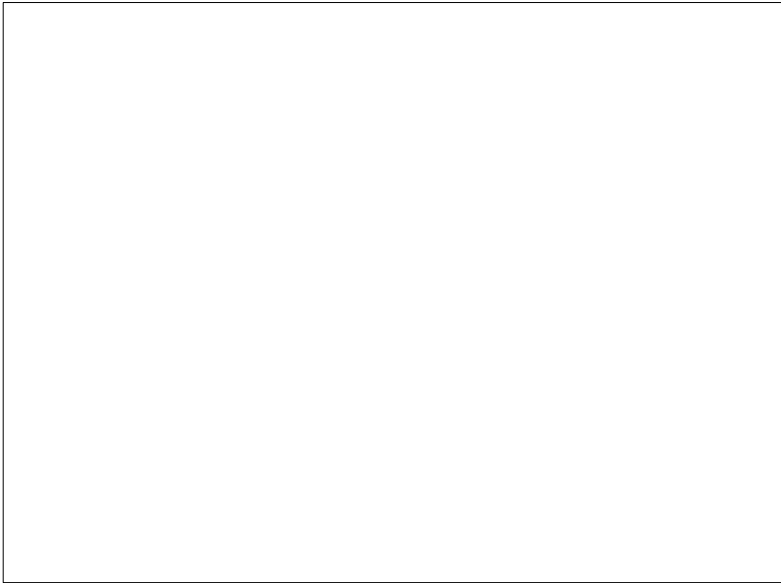
```
    model->PlayAnimation(Anim_Jump, false);
```

```
}
```

```
// ジャンプステート更新処理
```

```
void Player::UpdateJumpState(float elapsedTime)
```

```
{
```



```
}
```

```
// 着地ステートへ遷移
```

```
void Player::TransitionLandState()
```

```
{
```

```
    state = State::Land;
```

```
    // 着地アニメーション再生
```

```
    model->PlayAnimation(Anim_Landing, false);
```

```
}
```

```
// 着地ステート更新処理
```

```
void Player::UpdateLandState(float elapsedTime)
```

```
{
```



ゲームプログラミング改

```
}
```

実装が完了したら実行確認してみましょう。

プレイヤーのステート毎に正しい処理やアニメーションが実行されていれば OK です。

お疲れさまでした。