3D アクションゲームプログラミング

○評価要件

☑衝撃による吹き飛ばし処理
☑慣性によるキャラクター移動処理
☑空中でのキャラクター移動制御処理

○概要

今回は速力に関する処理を実装します。
速力とは運動する物体の進む速さです。
今までに実装したジャンプでは、上方向に力を加えますが、重力によって上方向に進む力が弱まっていきます。このときの進む力が速力です。

この速力の計算によってオブジェクトを吹き飛ばす処理や慣性移動などが実現できます。
今回は、前回の続きとして弾丸を敵に当てた際に衝撃を与えて吹き飛ばす処理を実装します。

また、現状ではゲームパッドの左スティックを入力するとキャラクターが移動しますが、左スティックの入力をニュートラルに戻すとピタッと移動が止まります。
慣性移動を実装するとスティックの入力をしていない状態でも一定時間減速しながら移動します。
わかりやすいゲームはスーパーマリオシリーズですが、スーパーマリオに限らず、だいたいのゲームでは操作感の向上のために慣性移動処理をしているので、本課題でも実装していきましょう。

○リファクタリング

今回、速力関連の処理を実装していきます。
UpdateVelocity()関数に処理を追加していくわけですが、今後のことも考えると処理がかなり増えていくことになるので、処理の内容を関数化します。

現在、縦方向の速力処理を実装しているので関数化しておきましょう。

Character.h

```
---省略---

// キャラクター
class Character
{
public:
    ---省略---

private:
    // 垂直速力更新処理
    void UpdateVerticalVelocity(float elapsedFrame);

    // 垂直移動更新処理
    void UpdateVerticalMove(float elapsedTime);

    ---省略---
};
```

```
---省略---

// 速力処理更新
void Character::UpdateVelocity(float elapsedTime)
{
    // 経過フレーム
    float elapsedFrame = 60.0f * elapsedTime;

    // 垂直速力更新処理
    UpdateVerticalVelocity(elapsedFrame);

    // 垂直移動更新処理
    UpdateVerticalMove(elapsedTime);

    // 重力処理
    velocity.y += gravity * elapsedFrame;

    // 移動処理
    position.y += velocity.y * elapsedTime;

    // 地面判定
    if (position.y < 0.0f)
    {
        position.y = 0.0f;
```

```
        // 着地した
        if (!isGround)
        {
            OnLanding();
        }
        isGround = true;
        velocity.y = 0.0f;
    }
    else
    {
        isGround = false;
    }
}

---省略---

// 垂直速力更新処理
void Character::UpdateVerticalVelocity(float elapsedFrame)
{
    // 重力処理
    
}

// 垂直移動更新処理
void Character::UpdateVerticalMove(float elapsedTime)
{
    // 移動処理
    
    // 地面判定
    
}
```

実行確認して以前と同じ挙動になっているか確認しておきましょう。
リファクタリングの際に引数を間違えてしまうことなどよくあるので小まめに確認しましょう。

○吹き飛ばし処理

オブジェクトを吹き飛ばすにはどうすれば良いでしょうか。
例えばテーブルの上に消しゴムがあったとして、消しゴムを指で弾いた（デコピンした）ら消しゴムが吹き飛びます。
吹き飛んだ消しゴムは壁との衝突や空気抵抗や地面の摩擦でいずれ動かなくなります。
この一連の処理をプログラムで実装していきます。

実装方法はジャンプ処理と似ています。
ジャンプ処理の時は上方向に力を与え、毎フレーム重力による下方向の力によって縦方向への加速と減速を実現していました。
今回は横方向も加味した減速処理を実装しましょう。

まずは、キャラクタークラスに衝撃を与える関数と減速用の変数を用意します。

Character.h

```
---省略---

// キャラクター
class Character
{
public:
    ---省略---

    // 衝撃を与える
    void AddImpulse(const DirectX::XMFLOAT3& impulse);

    ---省略---

private:
    ---省略---

    // 水平速力更新処理
    void UpdateHorizontalVelocity(float elapsedFrame);

    // 水平移動更新処理
    void UpdateHorizontalMove(float elapsedTime);

protected:
    ---省略---
    float                   friction = 0.5f;
};
```

Character.cpp

```
---省略---

// 衝撃を与える
void Character::AddImpulse(const DirectX::XMFLOAT3& impulse)
```

```
{
    // 速力に力を加える
    
}

---省略---

// 速力処理更新
void Character::UpdateVelocity(float elapsedTime)
{
    // 経過フレーム
    ---省略---

    // 垂直速力更新処理
    ---省略---

    // 水平速力更新処理
    

    // 垂直移動更新処理
    ---省略---

    // 水平移動更新処理
    
}

---省略---

// 水平速力更新処理
void Character::UpdateHorizontalVelocity(float elapsedFrame)
{
    // XZ平面の速力を減速する
    float length = 
    if (length > 0.0f)
    {
        // 摩擦力
        float friction = this->friction * elapsedFrame;

        // 摩擦による横方向の減速処理
        if (length > friction)
        {
            
        }
        // 横方向の速力が摩擦力以下になったので速力を無効化
        else
        {
            
        }
    }
```

```
}

// 水平移動更新処理
void Character::UpdateHorizontalMove(float elapsedTime)
{
    // 移動処理
    
}
```

弾丸と敵との衝突判定に吹き飛ばし処理を実装してみましょう。

Player.cpp

```
---省略---

// 弾丸と敵の衝突処理
void Player::CollisionProjectilesVsEnemies()
{
    ---省略---
    for (int i = 0; i < projectileCount; ++i)
    {
        ---省略---
        for (int j = 0; j < enemyCount; ++j)
        {
            ---省略---
            // 衝突処理
            DirectX::XMFLOAT3 outPosition;
            if (Collision::IntersectSphereVsCylinder(---省略---))
            {
                // ダメージを与える
                if (enemy->ApplyDamage(1, 0.5f))
                {
                    // 吹き飛ばす
                    {
                        DirectX::XMFLOAT3 impulse;
                        
                        enemy->AddImpulse(impulse);
                    }

                    // 弾丸破棄
                    ---省略---
```

吹き飛ばす方向と力を計算しよう

```
                }
            }
        }
    }
}
```

○慣性移動

吹き飛ばしの実装により、速力の減速処理が実装出来ました。
次は速力処理を拡張して慣性移動をできるようにしましょう。

ジャンプや吹っ飛ばし処理の時はボタンを押した瞬間に力を加えるという考えでしたが、今回はスティックを入力し続けている間は力を加え続けるということをする必要があります。

車で例えるとスティックを入力し続けている間はアクセルを踏み続けていて、スティックを離すとアクセルを踏むのを外すというイメージです。
車の場合、アクセルを踏み続けていると速度が速くなっていき、最高速度まで到達するとそれ以上のスピードにはなりません。逆にアクセルを踏むのを外すと減速していきいずれ止まります。

この内容をプログラムで実装していきましょう。

Character.h

```
---省略---

// キャラクター
class Character
{
    ---省略---
protected:
    // 移動処理
    void Move(float elapsedTime, float vx, float vz, float speed);
    void Move(float vx, float vz, float speed);

protected:
    ---省略---
    float                   acceleration = 1.0f;
    float                   maxMoveSpeed = 5.0f;
    float                   moveVecX = 0.0f;
    float                   moveVecZ = 0.0f;
};
```

Character.cpp

```
---省略---

// 移動処理
void Character::Move(float elapsedTime, float vx, float vz, float speed)
void Character::Move(float vx, float vz, float speed)
```

```
{
    speed *= elapsedTime;
    position.x += vx * speed;
    position.z += vz * speed;

    // 移動方向ベクトルを設定
    moveVecX = vx;
    moveVecZ = vz;

    // 最大速度設定
    maxMoveSpeed = speed;
}

---省略---

// 水平速力更新処理
void Character::UpdateHorizontalVelocity(float elapsedFrame)
{
    // XZ平面の速力を減速する
    ---省略---

    // XZ平面の速力を加速する
    if (length <= maxMoveSpeed)
    {
        // 移動ベクトルがゼロベクトルでないなら加速する
        float moveVecLength =
        if (moveVecLength > 0.0f)
        {
            // 加速力
            float acceleration = this->acceleration * elapsedFrame;
            // 移動ベクトルによる加速処理


            // 最大速度制限
            float length =
            if (length > maxMoveSpeed)
            {


            }
        }
    }
    // 移動ベクトルをリセット

}
```

今まで、この speed は elapsedTime を乗算した値だったが、乗算しない値とする

Player.cpp

```
---省略---
```

```
// 移動入力処理
void Player::InputMove(float elapsedTime)
{
    ---省略---

    // 移動処理
    Move(elapsedTime, moveVec.x, moveVec.z, moveSpeed);
    Move(moveVec.x, moveVec.z, moveSpeed);

    ---省略---
}

---省略---
```

実装出来たら実行確認しましょう。
ゲームパッドの左スティック操作で移動してみてスティックを離すと減速しながら停止できていれば OK です。

○空中制御

ジャンプ操作するゲームでは空中での移動制御をすることがあります。
地上にいる時は自由に移動操作ができますが、ジャンプなどで空中にいる時は移動操作が鈍くなるように制御する方が操作感が良くなると思います。

既に地面判定をすることはできるはずなので空中にいるときの移動操作が鈍くなるように実装しましょう。

Character.h

```
---省略---

// キャラクター
class Character
{
    ---省略---
protected:
    ---省略---
    float                   airControl = 0.3f;
};
```

Character.cpp

```
---省略---
// 水平速力更新処理
void Character::UpdateHorizontalVelocity(float elapsedFrame)
{
    // XZ平面の速力を減速する
    ---省略---
    if (length > 0.0f)
    {
```

```
        // 摩擦力
        float friction = this->friction * elapsedFrame;

        // 空中にいるときは摩擦力を減らす
        
        // 摩擦による横方向の減速処理
        ---省略---
    }

    // XZ平面の速力を加速する
    if (length <= maxMoveSpeed)
    {
        // 移動ベクトルがゼロベクトルでないなら加速する
        ---省略---
        if (moveVecLength > 0.0f)
        {
            // 加速力
            float acceleration = this->acceleration * elapsedFrame;

            // 空中にいるときは加速力を減らす
            
            // 移動ベクトルによる加速処理
            ---省略---
        }
    }
    ---省略---
}
```

実行確認してみましょう。
空中時での移動操作がいい感じになるように調整していきましょう。

お疲れさまでした。