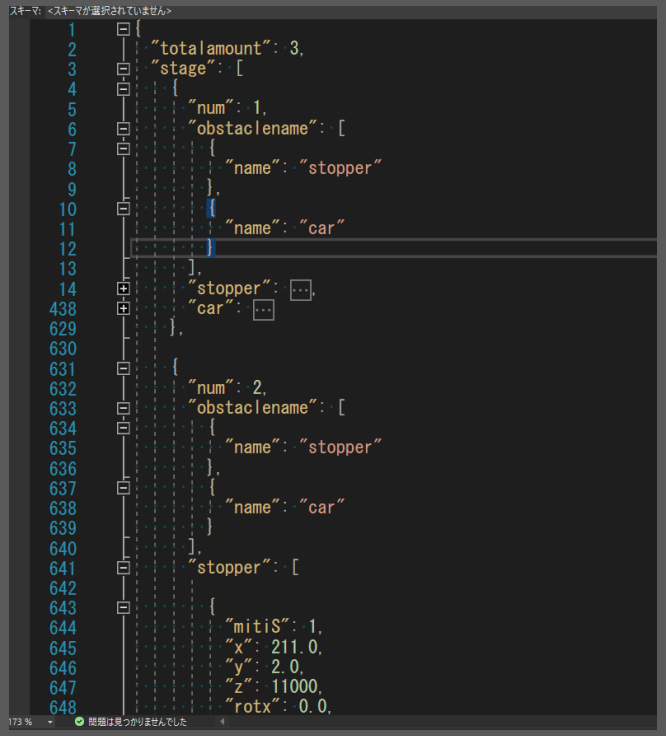
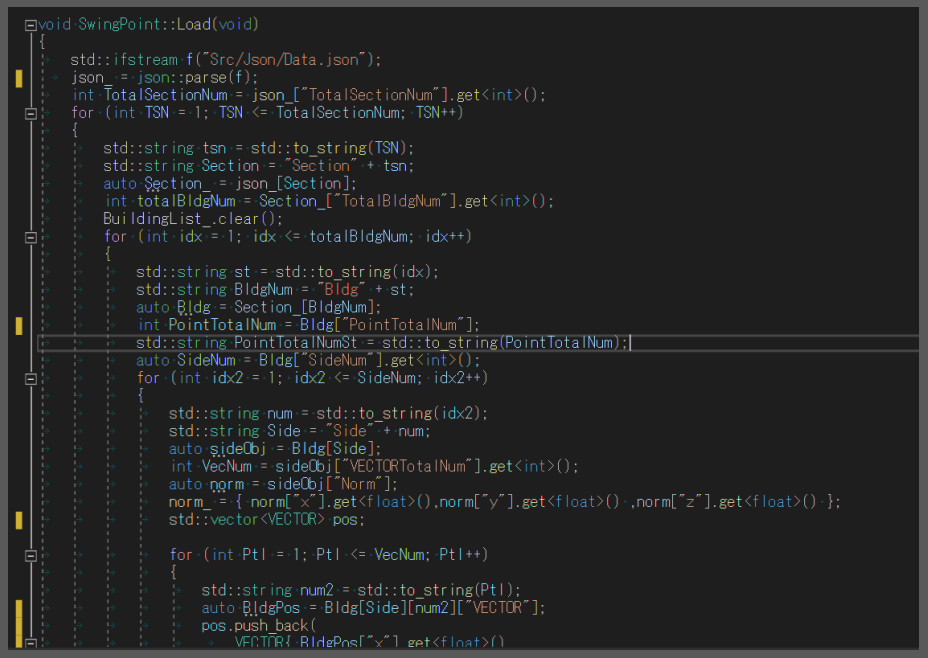
スウィングアクションを
実現させるために
こだわったポイント

フックポイントの読みこみ

フックポイントの座標をJsonで管理できるようにして、作業効率を向上させました。



```
スキーマ: <スキーマが選択されていません>
{
  "totalamount": 3,
  "stage": [
    {
      "num": 1,
      "obstaclename": [
        {
          "name": "stopper"
        },
        {
          "name": "car"
        }
      ],
      "stopper": ...,
      "car": ...
    },

    {
      "num": 2,
      "obstaclename": [
        {
          "name": "stopper"
        },
        {
          "name": "car"
        }
      ],
      "stopper": [

        {
          "mitiS": 1,
          "x": 211.0,
          "y": 2.0,
          "z": 11000,
          "rotx": 0.0,
173 %  問題は見つかりませんでした
```



```
void SwingPoint::Load(void)
{
    std::ifstream f("Src/Json/Data.json");
    json_ = json::parse(f);
    int TotalSectionNum = json_["TotalSectionNum"].get<int>();
    for (int TSN = 1; TSN <= TotalSectionNum; TSN++)
    {
        std::string tsn = std::to_string(TSN);
        std::string Section = "Section" + tsn;
        auto Section_ = json_[Section];
        int totalBldgNum = Section_["TotalBldgNum"].get<int>();
        BuildingList_.clear();
        for (int idx = 1; idx <= totalBldgNum; idx++)
        {
            std::string st = std::to_string(idx);
            std::string BldgNum = "Bldg" + st;
            auto Bldg = Section_[BldgNum];
            int PointTotalNum = Bldg["PointTotalNum"];
            std::string PointTotalNumSt = std::to_string(PointTotalNum);
            auto SideNum = Bldg["SideNum"].get<int>();
            for (int idx2 = 1; idx2 <= SideNum; idx2++)
            {
                std::string num = std::to_string(idx2);
                std::string Side = "Side" + num;
                auto sideObj = Bldg[Side];
                int VecNum = sideObj["VECTORTotalNum"].get<int>();
                auto norm = sideObj["Norm"];
                norm_ = { norm["x"].get<float>(),norm["y"].get<float>() ,norm["z"].get<float>() };
                std::vector<VECTOR> pos;

                for (int Ptl = 1; Ptl <= VecNum; Ptl++)
                {
                    std::string num2 = std::to_string(Ptl);
                    auto BldgPos = Bldg[Side][num2]["VECTOR"];
                    pos.push_back(
                        VECTOR{ BldgPos["x"].get<float>()
```

フックポイントの決定

```
const VECTOR SwingPoint::SetSwingPoint(VECTOR pos, int section, VECTOR Angle)
{
    min = 9999999.0f;
    auto pop = pos;
    auto angle= Angle;
    distance_.clear();
    comparison_.clear();
    //自分から近いビルから
    for (int idx =0; idx< BillPpoint_.size();idx++)
    {
        auto tesP = BillPpoint_[idx];
        tesP.y = 0.0f;
        auto p =VSub(pop, tesP);
        float pp = abs(p.x) +abs( p.z);
        distance_.push_back(pp);
    }
    BillNum_ =0;
    for (int idx = 0; idx < distance_.size(); idx++)
    {
        if (distance_[idx] <= min)
        {
            min = distance_[idx];
            BillNum_ = idx;
        }
    }
    distance_.clear();
    min = 9999999.0f;
    for (int idx = 0; idx < swingList3_[BillNum_].size(); idx++)
    {
        auto Point = swingList3_[BillNum_];
        auto p = VSub(pop, Point[idx]);
        float pp = abs(p.x) + abs(p.z);
        distance_.push_back(pp);
    }
    swingNum_ = 0;
    for (int idx = 0; idx < distance_.size(); idx++)
    {
        if (distance_[idx] <= min)
        {
            min = distance_[idx];
            swingNum_ = idx;
        }
    }
    return swingList3_[BillNum_][swingNum_];
}
```

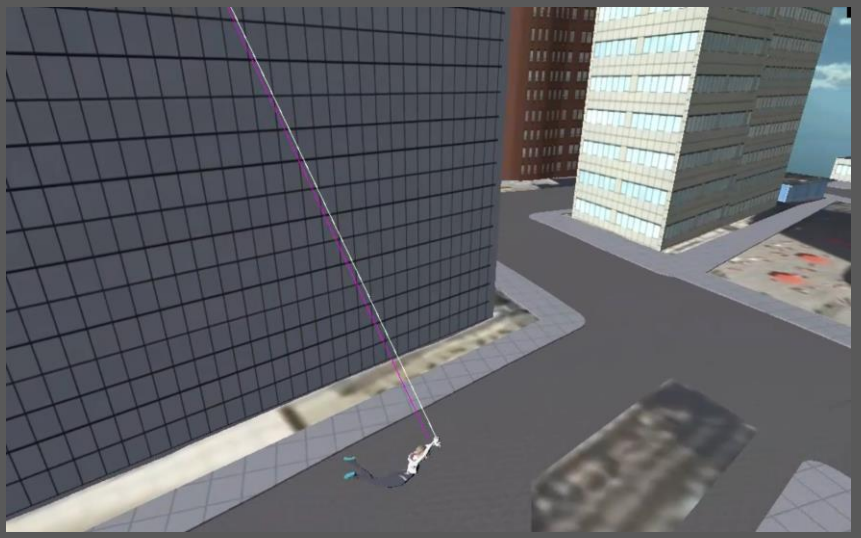
自分から一番近いビルまず決定して、そののかからさらに一番近いポイントをフックポイントにしています。

ウェブスイング

```
bool Player::SetSwingParam(VECTOR pos ,VECTOR end)
{
    stringV_ = VSub(transform_.pos,endPos_);//支点から錘へののベクトル
    VECTOR GroundV = VSub(VECTOR{ 0.0f,100.0f,0.0f }, endPos_);
    auto stleng = Magnitude(VECTOR{ 0.0f,stringV_.y,0.0f });
    auto GroundLength = Magnitude(GroundV);
    if (abs(stleng) >= abs(GroundLength))
    {
        return false;
    }
    length_ = Magnitude(stringV_);//ヒモの長さ
    swingGravityNorm_ = VNorm(swingGravity_);
    float x = Dot(swingGravityNorm_, stringV_);//重力ベクトルと錘から重力軸までのベクトルが交わる点と支点の大きさ
    swingYnorm_ = stringV_ - swingGravityNorm_ * x;//錘から重力軸までのベクトル
    float y = Magnitude(swingYnorm_);                //錘から重力軸までの正規化済みベクトル
    swingYnorm_=Normalized(swingYnorm_);

    theta_ = atan2f(y, x);                //支点の近接二辺のそれぞれの大きさを使用して角度を出す
    VECTOR gravity_ = VScale(swingGravityNorm_, 3500.0f);
    gMag_ = Magnitude(gravity_);
    omega_ = -0.5f;                                  //角速度は0で初期化
    transform_.Update();
    isSwingFlag_ = true;
    phase_ = &Player::UpdatePendulum;
    return true;
}
```

```
auto coe = -gMag_ / length_;                //kの係数

//ルンゲ=クッタ法の計算
auto k1a = delta * coe * sin(theta_);
auto m1a = delta * omega_;

auto k2a = delta * coe * sin(theta_ + m1a / 2.0f);
auto m2a = delta * (omega_ + k1a / 2.0f);

auto k3a = delta * coe * sin(theta_ + m2a / 2.0f);
auto m3a = delta * (omega_ + k2a / 2.0f);

auto k4a = delta * coe * sin(theta_ + m3a);
auto m4a = delta * (omega_ + k3a);

omega_ += (k1a + 2.0f * k2a + 2.0f * k3a + k4a) /5.0f;      //角速度の加算
theta_ += (m1a + 2.0f * m2a + 2.0f * m3a + m4a) / 5.0f;     //ｚ角度の加算

auto stv = Normalized(stringV_);
if (isSwingFlag_==true)
{
    auto l = endPos_ + length_ * cos(theta_) * Normalized(swingGravityNorm_) + length_ * sin(theta_) * swingYnorm_;
    movePow_ = VSub(l, transform_.pos);
}
```

スイングを始める前に、計算に必要な
各種パラメータを最初に設定します。
そのパラメータを使いスイングをさせています。



垂直ではなく斜めに

垂直ではビルにぶつかってしまって、
何も面白くありません。
「動かすだけで楽しい良い操作性」
のために斜めにしています。