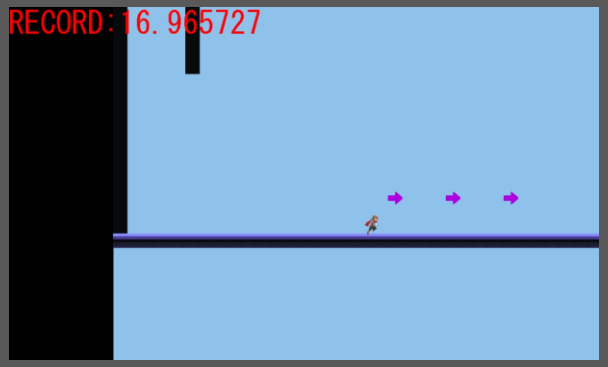
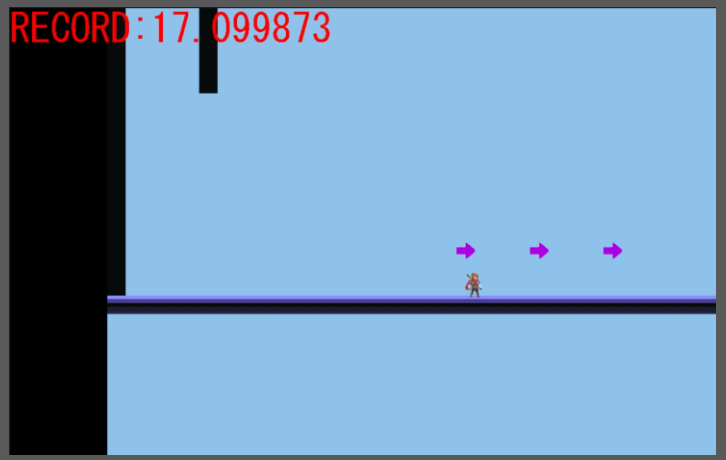
「動かすだけで楽しい」を
実現させるために
こだわったポイント

走り始めと止まるとき1/2

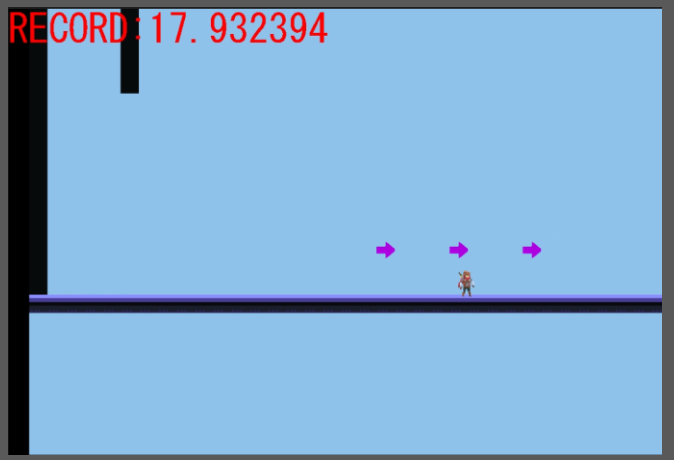


走っている途中に止まる

この位置で移動入力をやめる



走っていた時のスピード分滑る！！



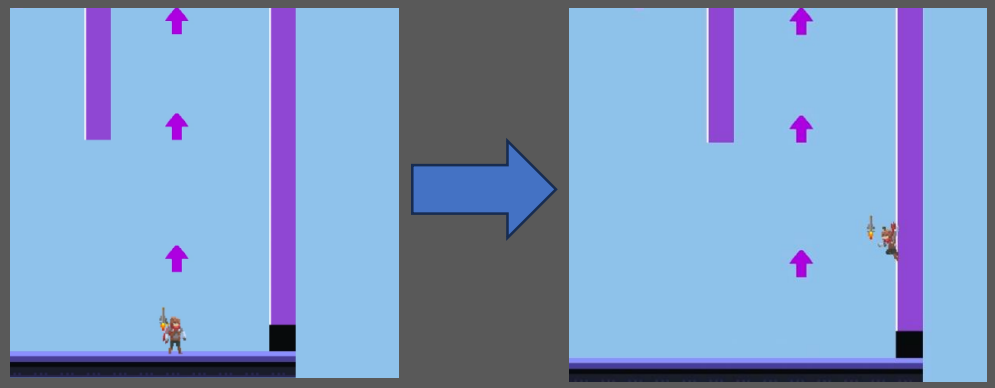
走り始めと止まるとき2/2

数行の地味なプログラムでも
少しの動きの違いで、
遊びの楽しさが変わるので、
細々とした工夫をちりばめています。

```
void Player::Move(Input& input)
{
    float speed = 0.2f;
    //右とは左キーが押されていないとき
    if (!input.IsPrassed("right")&&!input.IsPrassed("left"))
    {
        if (_phase == &Player::MovePhase)
        {
            //移動量が0.1より大きかったら
            if (movePow_.x >= 0.1f)
            {
                movePow_.x -= speed;
            }
            //移動量が-0.1より小さかったら
            if (movePow_.x <= -0.1f)
            {
                movePow_.x += speed;
            }
            if (dir_LR_ == DIR_LR::RIGHT)
            {
                if (0.40f >= movePow_.x && movePow_.x >= 0.02f)
                {
                    movePow_.x = 0.0f;
                }
            }
            if (dir_LR_ == DIR_LR::LEFT)
            {
                if (-0.40f <= movePow_.x && movePow_.x <= -0.02f)
                {
                    movePow_.x = 0.0f;
                }
            }
        }
        //ジャンプアニメーション中じゃなかったら
        if (_phase == &Player::MovePhase) { lpAnimMng.SetAnime(animeStr_, "Idle"); }
    }
```

壁での動き1/2

操作ミスや敵の妨害以外で基本止まらない、ずっと動いているという
スピード感を保つために、壁ジャンプを工夫しました。

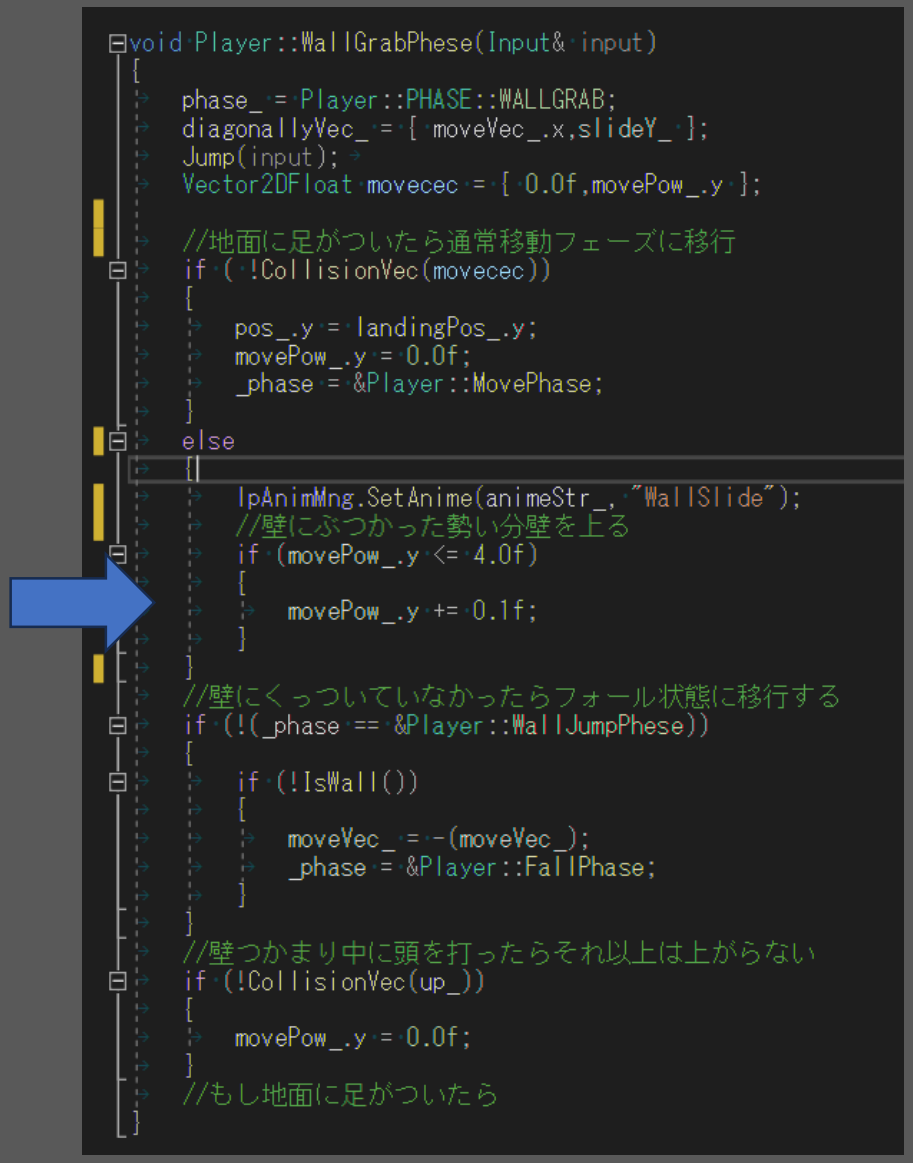


プレイヤーが壁にジャンプしてきた時の
スピードの分だけ、プレイヤーが
壁を少しずり上がるようにしました。
これによって連続の壁ジャンプをスムーズに
テンポ良く行うことができるようになりました。

上画像：助走なし
右画像：助走あり

壁での動き2/2

実際の壁つかまり状態でのコード

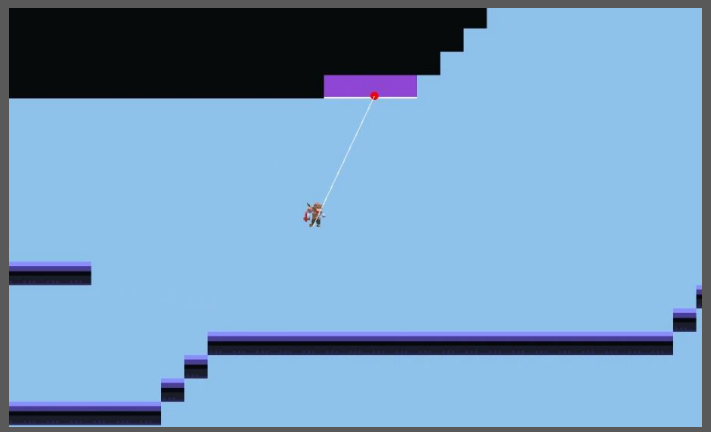


```
void Player::WallGrabPhese(Input& input)
{
    phase_ = Player::PHASE::WALLGRAB;
    diagonallyVec_ = { moveVec_.x,slideY_ };
    Jump(input);
    Vector2DFloat movecec = { 0.0f,movePow_.y };

    //地面に足がついたら通常移動フェーズに移行
    if ( !CollisionVec(movecec))
    {
        pos_.y = landingPos_.y;
        movePow_.y = 0.0f;
        _phase = &Player::MovePhase;
    }
    else
    {
        lpAnimMng.SetAnime(animeStr_, "WallSlide");
        //壁にぶつかった勢い分壁を上る
        if (movePow_.y <= 4.0f)
        {
            movePow_.y += 0.1f;
        }
    }
    //壁にくっついていなかったらフォール状態に移行する
    if (!(_phase == &Player::WallJumpPhese))
    {
        if (!IsWall())
        {
            moveVec_ = -(moveVec_);
            _phase = &Player::FallPhase;
        }
    }
    //壁つかまり中に頭を打ったらそれ以上は上がらない
    if (!CollisionVec(up_))
    {
        movePow_.y = 0.0f;
    }
    //もし地面に足がついたら
}
```

みんな大好きスウィングアクション1/1

ワイヤーが付くと初期パラメータを設定

```
void Wire::SetSwingPalam()
{
    auto lVec = player_.pos_ - fulcrum_;    //支点→錘のベクトル(紐)
    length_ = lVec.Magnitude();                    //紐の長さ
    vel_.x = player_.movePow_.x;    //初速度的な
    //ここでアングルの初期設定をする
    angle_ = atan2f(player_.pos_.x - fulcrum_.x, player_.pos_.y - fulcrum_.y);
    v_ = -2 * vel_.x * cosf(angle_);//x軸の速度

    if (player_.dir_LR_ == Player::DIR_LR::LEFT)
    {
        pow_ = 0.15f;
    }
    else
    {
        pow_ = -0.15f;
    }
    _phase = &Wire::SwingPhase;
    player_.StartSwing();
}
```

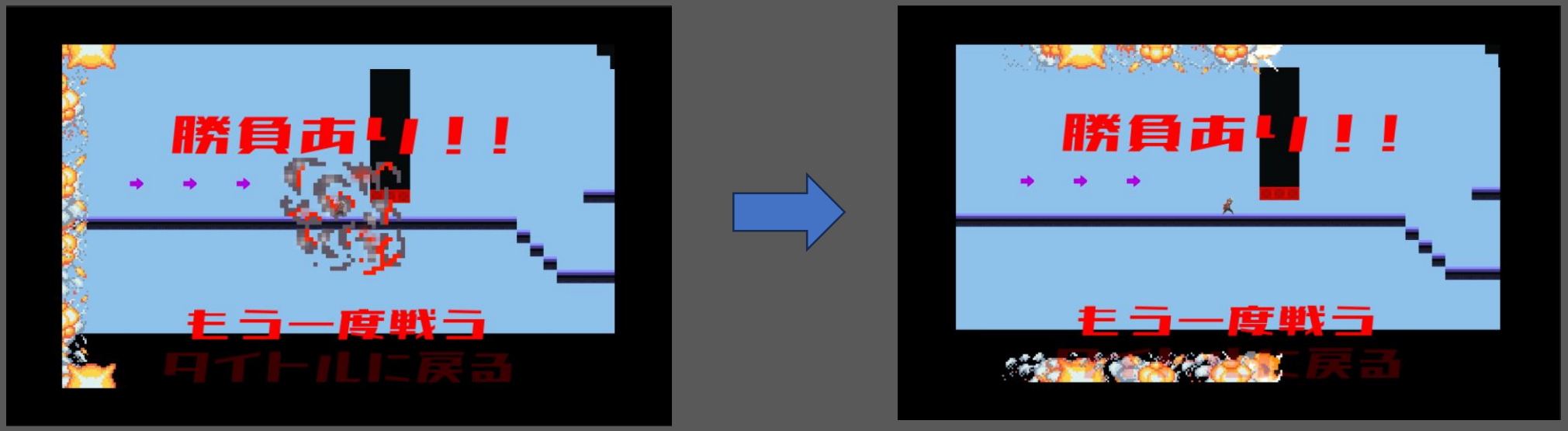


それを元に移動値を計算して反映

```
void Wire::SwingPhase()
{
    auto gravity = 0.5f;
    //アングルをけってい
    angle_ = atan2f(player_.pos_.x - fulcrum_.x, player_.pos_.y - fulcrum_.y);
    v_ += gravity * sinf(angle_);
    vel_ = { -v_ * cosf(angle_),v_ * sinf(angle_) };
    Vector2DFloat vel = { vel_.x,vel_.y };
    if (_phase == &Wire::SwingPhase)
    {
        player_.pos_ += vel;        //velを加算
        player_.pos_ = fulcrum_ + (player_.pos_ - fulcrum_).Normalized() * length_;//長さを補正
    }
    if (player_.pos_.y <= fulcrum_.y + -150.0f)
    {
        SwingJump();
        player_.StartSwingJump();
    }
}
```

その他演出1/2

プレイヤーの脱落を派手に教えてくれる画面の端を走る爆発



その他演出2/2

指定の画面端まで行くと
進む向きを変える

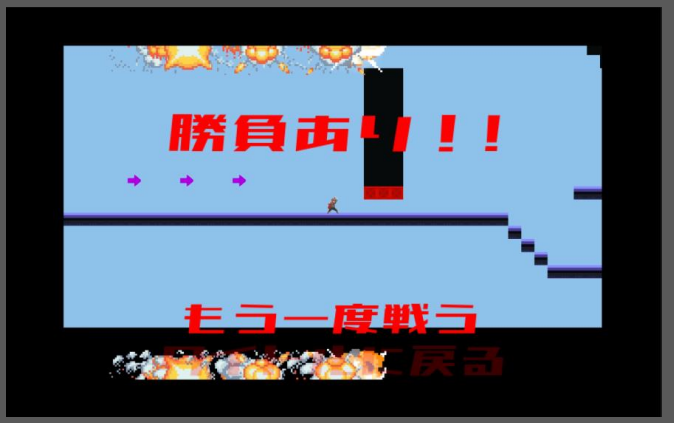
```
if (isExploding_)
{
    //lowerは左回り
    //画面右上だったら左に行く
    if (lowerPos_.y <= minPos_.y && lowerPos_.x >= maxPos_.x)
    {
        lowerVec_ = { -20.0f ,0.0f };
        lowerSide_ = SIDE::UP;
    }
    //画面右下なら上に行く
    if (lowerPos_.x >= maxPos_.x && lowerPos_.y >= maxPos_.y)
    {
        lowerVec_ = { 0.0f,-20.0f };
        lowerSide_ = SIDE::RIGHT;
    }
    //画面左下なら右に行く
    if (lowerPos_.y >= maxPos_.y && lowerPos_.x <= minPos_.x)
    {
        lowerVec_ = { 20.0f ,0.0f };
        lowerSide_ = SIDE::DOWN;
    }
    //画面左上なら下に行く
    if ( lowerPos_.x <= minPos_.x &&lowerPos_.y <= minPos_.y)
    {
        lowerVec_ = { 0.0f ,20.0f };
        lowerSide_ = SIDE::LEFT;
    }

    //画面左上になったら右に行く
    if (upperPos_.y<= minPos_.y && upperPos_.x <= minPos_.x)
    {
        upperVec_ = { 20.0f,0.0f };
        upperSide_ = SIDE::UP;
    }
    //画面右上になったら下に行く
    if (upperPos_.x >= maxPos_.x && upperPos_.y <= minPos_.y)
    {
        upperVec_ = { 0.0f,20.0f };
        upperSide_ = SIDE::RIGHT;
    }
    //画面右下になったら左に行く
    if (upperPos_.y >= maxPos_.y && upperPos_.x >= maxPos_.x)
    {
        upperVec_ = { -20.0f ,0.0f };
```

```
//一定時間ごとに爆発させる、あと音も出す
if ((frame_) % 3 == 0)
{
    upBombs_.emplace_back(upperPos_, upperSide_);
    downBombs_.emplace_back(lowerPos_, lowerSide_);
    PlaySoundMem(ExplosionSound_, DX_PLAYTYPE_BACK, true);
}
SideChange(upperPos_, upperSide_);
SideChange(lowerPos_, lowerSide_);

//上からの爆弾と下からの爆弾が重なったらどっちも消す
if (lowerPos_.distance(upperPos_)<40)
{
    PlaySoundMem(ExplosionSound_, DX_PLAYTYPE_BACK, true);
    StartJoypadVibration(padNum_, 1000, 400);
    upBombs_.clear();
    isExploding_ = false;
    bigExploding_ = true;
    bigFrame_ = 0;
}
frame_++;
auto bombsCheck = [this](std::list<Bomb>& bomb)
{
    for (auto& b : bomb)
    {
        b.frame_++;
        SideChange(b.pos_, b.side_);

        if (b.frame_ > 29)
        {
            b.isDead = true;
        }
    }
};
bombsCheck(upBombs_);
bombsCheck(downBombs_);

upBombs_.remove_if([](const Bomb& b){
    return b.isDead;
    });
downBombs_.remove_if([](const Bomb& b){
    return b.isDead;
```

二列の爆発が重なると
大爆発する

狭まる画面端にも対応

狭い



広い