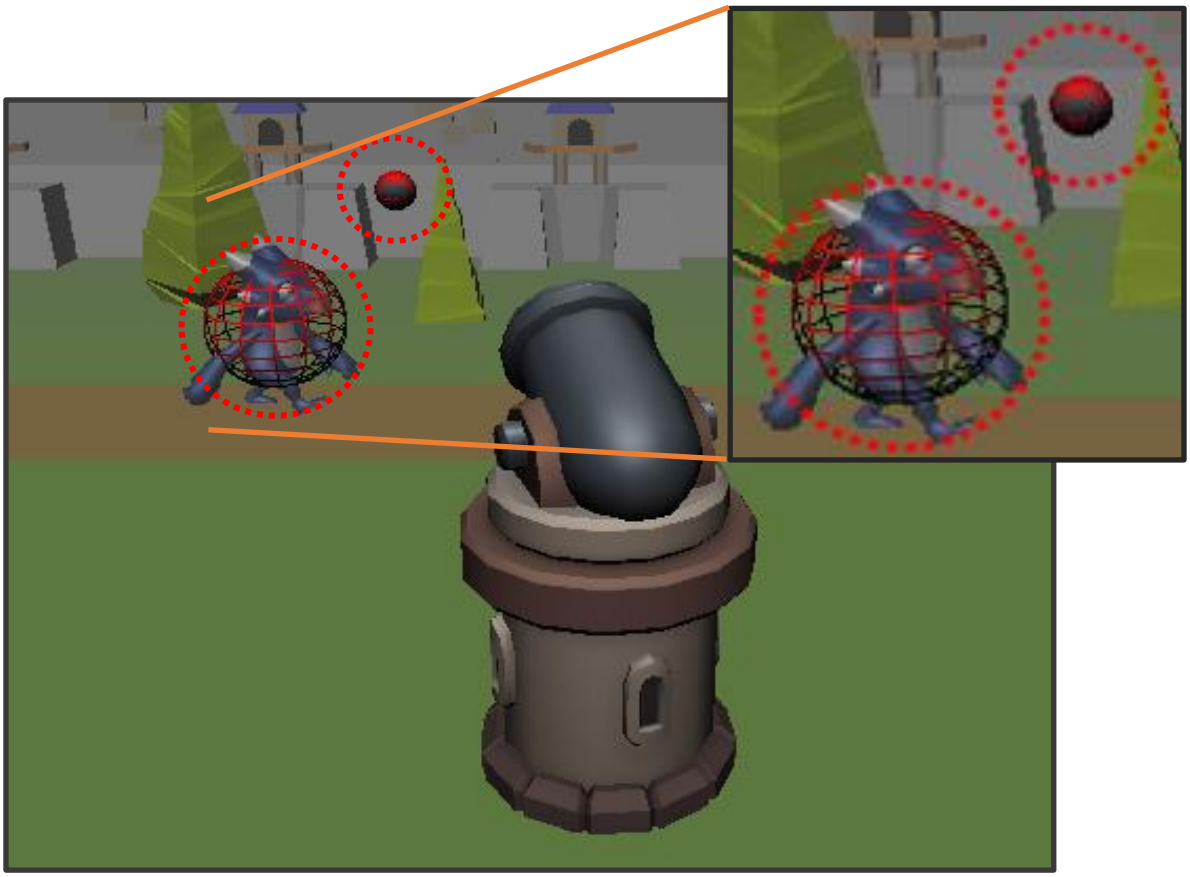
# 仕上げのゲームオーバー判定

弾と敵との衝突判定を作って、敵を倒せるようにして、
その後、敵が右の陣地に到達したら、ゲームオーバーという流れを
作っていきましょう。

ステージの衝突判定で実装したように、敵キャラのポリゴンと球体(弾)の
衝突判定をとっても良いですが、できるだけ処理負荷はかけない方が
良いですので、実装も簡単で、処理速度も速い、球体と球体の衝突判定で
実装していきましょう。



実装の詳細については、"衝突判定３Ｄ_球体と球体"をご参照ください。

衝突処理を作る時は、球体なら球体を表示させて、
衝突範囲がしっかり視認できるようにしておきましょう。

DrawSphere3D(座標, 半径, 10, 0xff0000, 0xff0000, false);
↑第６引数を false にすることで球体がワイヤーフレーム表示になり、
　範囲とモデルが見やすくなります

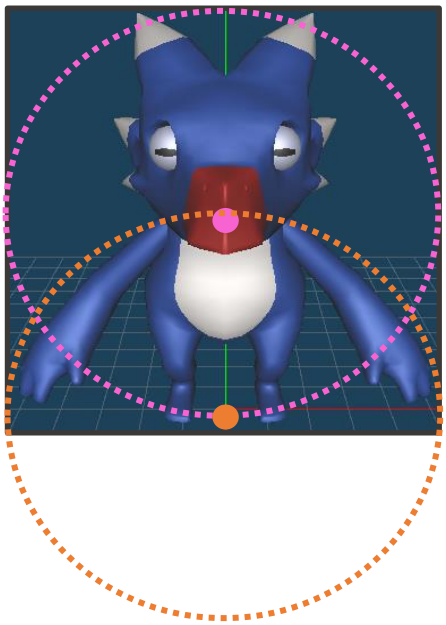
EnemyBaseに衝突判定用の球体情報である、中心位置と半径を持たせて、
衝突判定自体は、GameSceneに実装していきましょう。

EnemyBase.h

```
class EnemyBase
{
public:
    ～ 省略 ～

    // 衝突用の中心座標の取得
    VECTOR GetCollisionPos(void);

    // 衝突用の球体半径の取得
    float GetCollisionRadius(void);

protected:
    ～ 省略 ～

    // 衝突判定用の球体半径
    float collisionRadius_;

    // 衝突判定用の球体中心の調整座標
    VECTOR collisionLocalPos_;
```

EnemyBase.cpp

```
void EnemyBase::SetParam(void)
{
    ～ 省略 ～

    // HPの設定
    hp_ = hpMax_ = 2;

    // 衝突判定用の球体半径
    collisionRadius_ = 35.0f;

    // 衝突判定用の球体中心の調整座標
    collisionLocalPos_ = { 0.0f, 50.0f, 0.0f };
}
```



ロックマンの時の用に、
衝突判定用の座標を
collisionLocalPos_ で調整しましょう。

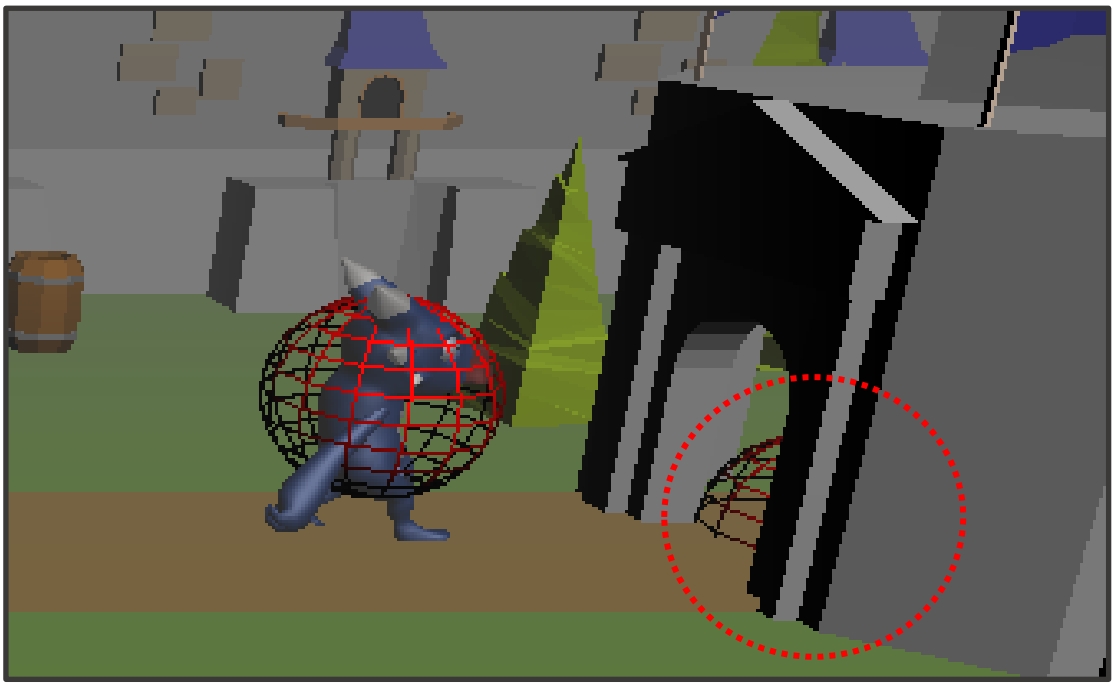
敵キャラの座標は、
pos_（オレンジの点）で制御していますが、
そこを球体の中心としてしまうと、
意図した衝突判定からズレてしまいますので、
pos_ に collisionLocalPos_ を足して上げて、
敵キャラ全体のモデルを覆うような球体に
していきます。

GameScene.cpp

```
void GameScene::Update(void)
{
    ～ 省略 ～

    auto shots = cannon_->GetShots();
    for (auto shot : shots)
    {
        ～ 省略 ～

        // 敵との衝突判定
        for (auto enemy : enemys_)
        {

            // 球体と球体の衝突判定
            ？？？
            {
                // 敵にダメージを与える
                enemy->Damage(1);
                shot->Blast();
                break;
            }

        }
```

敵キャラのHPが2に対して、弾のダメージが1ですので、
2回弾を当てると、敵キャラが非表示になるはずです。

次にゲームオーバー判定です。
今回は、シーン遷移もせずに、パっと済ませましょう。
敵キャラが右側の建物に侵入したら、ゲームオーバーとしたいと思います。
敵キャラの座標(位置)で判断しても良いですが、
せっかくなので復習も兼ね、先ほどの球体と球体の衝突判定を使いましょう。



GameScene.h

```
～ 省略 ～

class GameScene : public SceneBase
{

    // ゲームオーバー地点衝突判定用球体半径
    static constexpr float OVER_COL_RADIUS = 35.0f;

    ～ 省略 ～

private:
    ～ 省略 ～

    // ゲームオーバー地点
    VECTOR gameoverPoint_;

    // ゲームオーバー判定
    bool isGameover_;

    // ゲームオーバー画像
    int imgGameover_;
};
```

GameScene.cpp

```
void GameScene::Init(void)
{
    ～ 省略 ～

    // ゲームオーバー地点
    gameoverPoint_ = { 450.0f, 30.0f, 75.0f };

    // ゲームオーバー判定
    isGameover_ = false;

    // ゲームオーバー画像
    imgGameover_ =
          LoadGraph((Application::PATH_IMAGE + "Gameover.png").c_str());
}
```

```
void GameScene::Update(void)
{

    // シーン遷移
    InputManager& ins = InputManager::GetInstance();
    if (ins.IsTrgDown(KEY_INPUT_SPACE))
    {
        SceneManager::GetInstance().ChangeScene(SceneManager::SCENE_ID::TITLE);
    }

    if (isGameover_)
    {
        // ゲームオーバーなら処理しない
        return;
    }

    ～ 省略 ～

    // ゲームオーバー判定
    ？？？

    敵のループを回して、１体１体ゲームオーバーポイントとの
    衝突判定を確認して、衝突していたら、ゲームーバーフラグを true にする。

}

void GameScene::Draw(void)
{

    ～ 省略 ～

    // ゲームオーバー画像の表示
    ？？？
    どのようにすれば良いか、考えながら実装してください。

}
```

敵がゲームオーバーポイントに到達したら、
画像が表示され、Cannonの操作が不能になったらＯＫです。


シーン遷移を行うと、どうしても多少の演出時間が入るため、
ユーザを待たせることになります。

これまではシーンの流れを覚えて貰うために、
ゲームオーバーシーンを作ってきましたが、テンポよくゲームクリアや
リトライをしたい場合は、このようにGameScene内で、
ゲームーオーバー判定を取っても問題ありません。

これで基本授業は終了となりますが、
たくさん改造して、皆さんなりのゲームに仕上げてみてください。