最終制作作品

作品名　　：Monster Must Die!
ジャンル　：3Dアクションタワーディフェンス
開発環境　：C++/DxLib、Unity、HLSL、Effekseer
対応機種　：Windows
制作時期　：2024/10/10 ~
制作期間　：約4か月目
制作人数　：1人
担当　　：モデル、UI、サウンド以外

GitHubURL　：
https://github.com/nkd10121/MonsterMustDie

GitHubQR ：

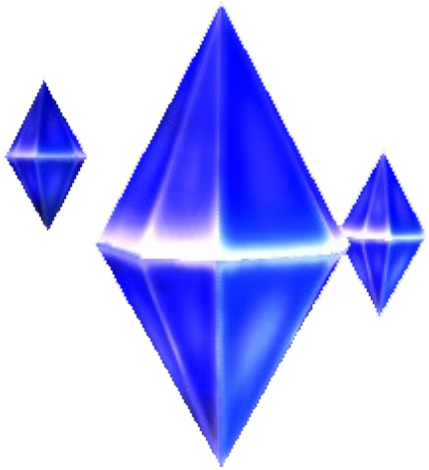
動画URL　：
https://youtu.be/er6HSM8xqhI

動画QR ：

ゲーム概要

床や壁に**罠**を設置して

迫りくるモンスターから**クリスタル**を守り切ろう!



拘ったポイント

タワーディフェンスのため、複数の当たり判定を持った敵や罠を大量に出しても重くならないように工夫しました。

↓当たり判定の計算をしない相手のオブジェクトを事前に設定

```cpp
/// <summary>
/// コンストラクタ
/// </summary>
TrapBase::TrapBase():
    ObjectBase(Collidable::Priority::Low, GameObjectTag::Trap),
    m_status(),
    m_isExist(false),
    m_isAttack(false),
    m_trapName(),
    m_isPreview(false),
    m_angle(0.0f)
{
    //敵以外のオブジェクトとは当たり判定をとらない
    AddThroughTag(GameObjectTag::Crystal);
    AddThroughTag(GameObjectTag::Player);
    AddThroughTag(GameObjectTag::SwarmEnemy);
    AddThroughTag(GameObjectTag::Portion);
    AddThroughTag(GameObjectTag::PlayerShot);
    AddThroughTag(GameObjectTag::Trap);
}
```

Object/Trap/TrapBase.cpp

↓当たり判定の計算前に相手のタグを確認

```cpp
//どちらかのオブジェクトが相手のオブジェクトタグを無視するようになっていたら当たり判定の計算対象に追加せずに次へ
if (obj1->IsThroughTarget(obj2) || obj2->IsThroughTarget(obj1))
{
    continue;
}
```

MyLib/Physics.cpp

↓敵同士の押出判定以外の計算を省略

```cpp
//敵同士かつ、どちらかが通常の当たり判定じゃないときは次に行く
bool isContinue = false;
if (objA->GetTag() == GameObjectTag::Enemy && objB->GetTag() == GameObjectTag::Enemy)
{
    if (colA->collideTag != MyLib::ColliderBase::CollisionTag::Normal ||
        colB->collideTag != MyLib::ColliderBase::CollisionTag::Normal)
    {
        isContinue = true;
    }
}
if (isContinue)
{
    continue;
}
```

MyLib/Physics.cpp

↓当たり判定の計算前に距離を計算して計算するか判断

```cpp
//もし規定距離より比較する二つの大きさの合計が大きければ問答無用で追加する
if (maxDistance < (lengthA + lengthB)* (lengthA + lengthB))
{
    // 判定リストに追加
    ret.push_back(obj1);
    ret.push_back(obj2);
    isAdd = true;
    break;
}

//もし2つのオブジェクトの距離が規定距離より短ければ当たり判定の計算を行う
if ((pos1 - pos2).SqLength() < maxDistance)
{
    // 判定リストに追加
    ret.push_back(obj1);
    ret.push_back(obj2);
    isAdd = true;
    break;
}
```

MyLib/Physics.cpp

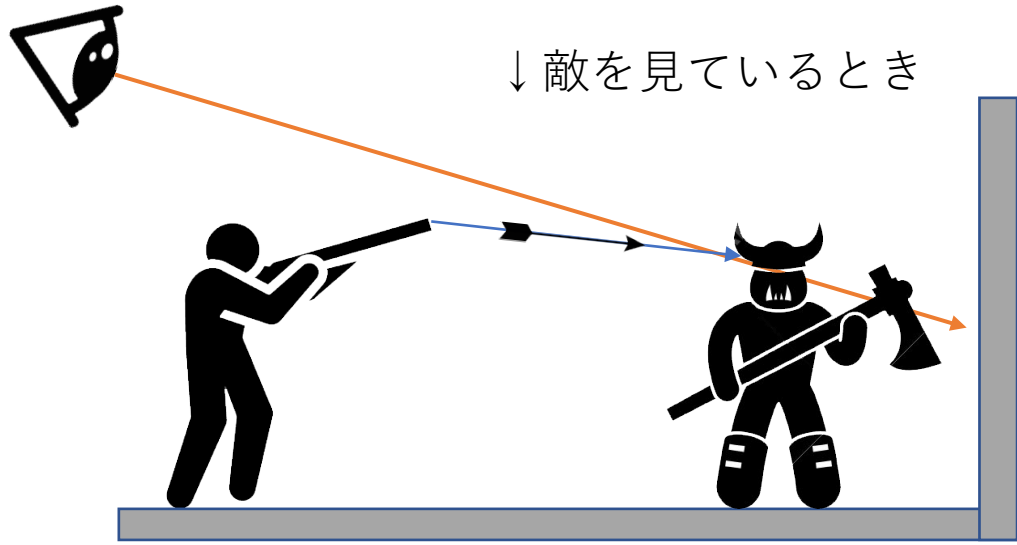
技術紹介:プレイヤーの攻撃

プレイヤーの遠距離攻撃は、
カメラから飛ばしたレイが当たっている地形や敵の座標を求めて、プレイヤーの手元から地形や敵の座標をめがけて矢を発射するようにしています。

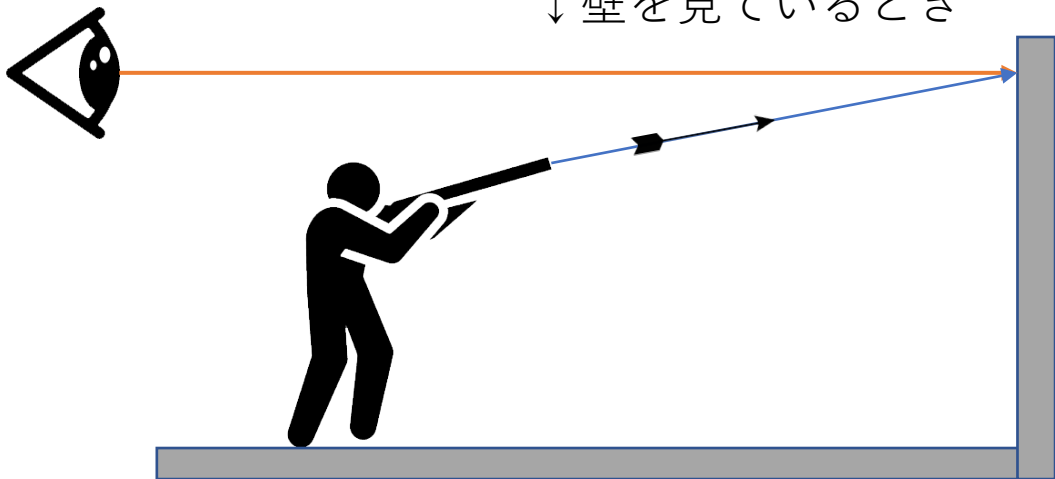
同じように、カメラから飛ばしたレイを使用して、
罠を設置する場所も決定しています。

このように実装することで、矢は照準通りに飛んでいき、
罠は見ているところに設置できるようになりました。

↓敵を見ているとき

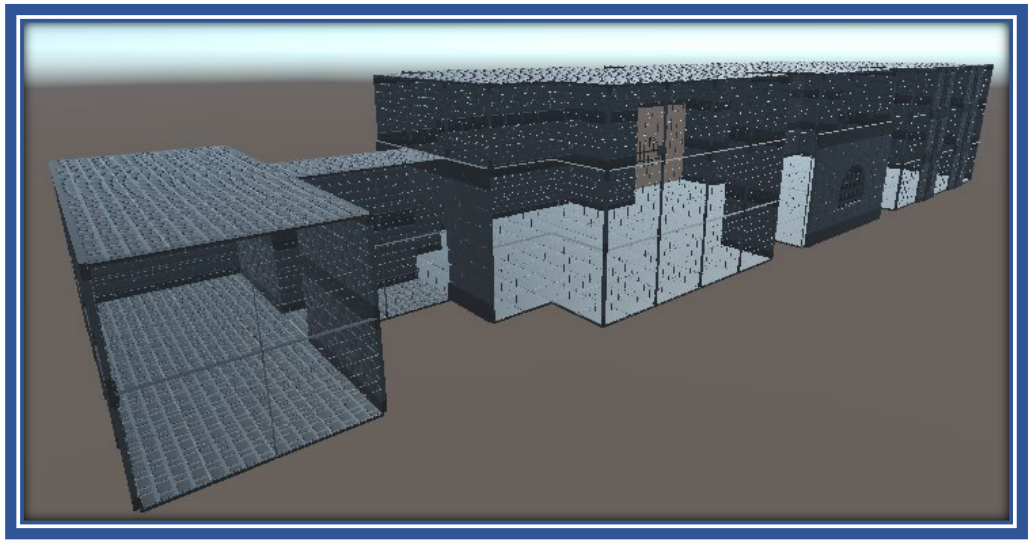


↓壁を見ているとき



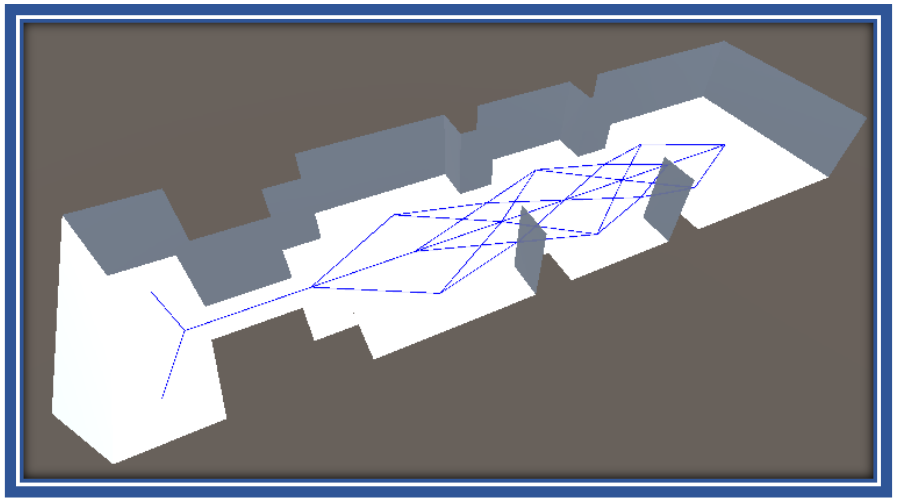
技術紹介:マップ制作

Unityをマップ制作ツールとして使用し、
マップを作成しました。
マップ構造、簡易当たり判定、
罠設置ポイント、敵のウェイポイント
を出力してゲームに使用しています。

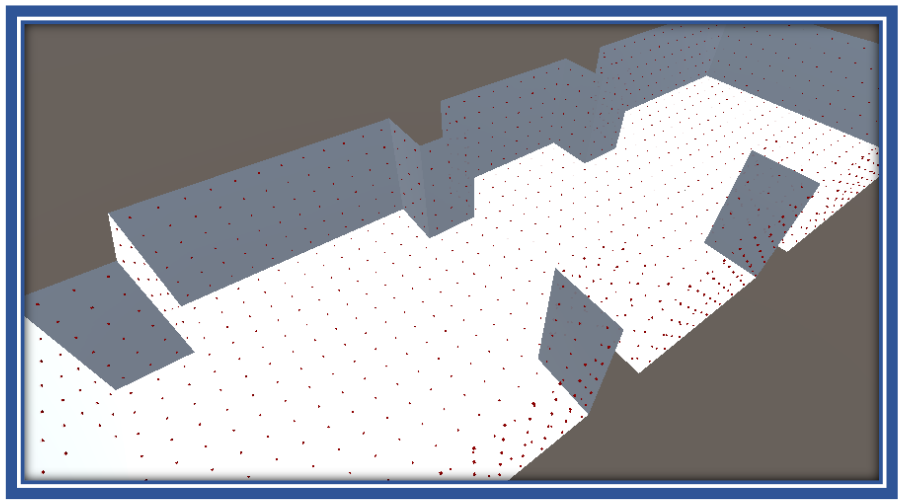


↑マップ構造

↓青い線が敵の移動ルート



↓赤い点が罠設置ポイント



↑白いオブジェクトが簡易当たり判定↑

技術紹介:外部ファイル化

プレイヤーや罠、敵のステータス、
各ステージ情報や敵の出現情報など
を外部ファイルで管理しています。

↓敵の出現情報

出現フェーズ	敵ID	出現秒数
1	EnemyNormal	0
1	EnemyNormal	0.5
1	EnemyNormal	1
1	EnemyNormal	1.5
1	EnemyNormal	7.5
1	EnemyNormal	8
1	EnemyNormal	8.5
1	EnemyNormal	9
1	EnemyNormal	10
2	EnemyNormal	0
2	EnemyNormal	0.5
2	EnemyNormal	1
2	EnemyNormal	1.5
2	EnemyNormal	2
2	EnemyBig	2.5

↓ステータス

名前	体力	攻撃力	防御力	移動速度	ドロップポイント	クリスタル与ダメージ
Player	200	12	0	0.5	0	0
EnemyNormal	75	15	0	0.36	20	1
EnemyFast	35	0	0	0.54	15	1
EnemyBig	120	25	2	0.3	30	3

名前	攻撃力	索敵範囲	攻撃範囲	クールタイム	設置コスト	設置場所
Spike	25	6.5	8	260	350	0
ArrowWall	35	5.5	8	360	600	1
Frame	14	6	8	600	800	0
Cutter	6	6	8	0	300	0
IronSnare	0	6	8	0	450	0
IronImpact	50	6	8	800	700	1

↓各ステージ情報

ステージIdx	ステージID	通常当たり判定ID	敵当たり判定ID	フェーズ数	初期罠ポイント	初期クリスタルHP	目標クリアタイム	ポーションがドロップする最大数	ステージ名	ミニマップID
0	stage1	M_STAGECOLLISION1	M_E_STAGECOL	3	1800	20	14400	3	ステージ1	I_STAGE1MINIMAP
1	stage2	M_STAGECOLLISION2	M_E_STAGECOL	3	1800	20	14400	3	ステージ2	I_STAGE2MINIMAP
2	stage1	M_STAGECOLLISION1	M_E_STAGECOL	3	1800	20	14400	3	ステージ3	I_STAGE1MINIMAP