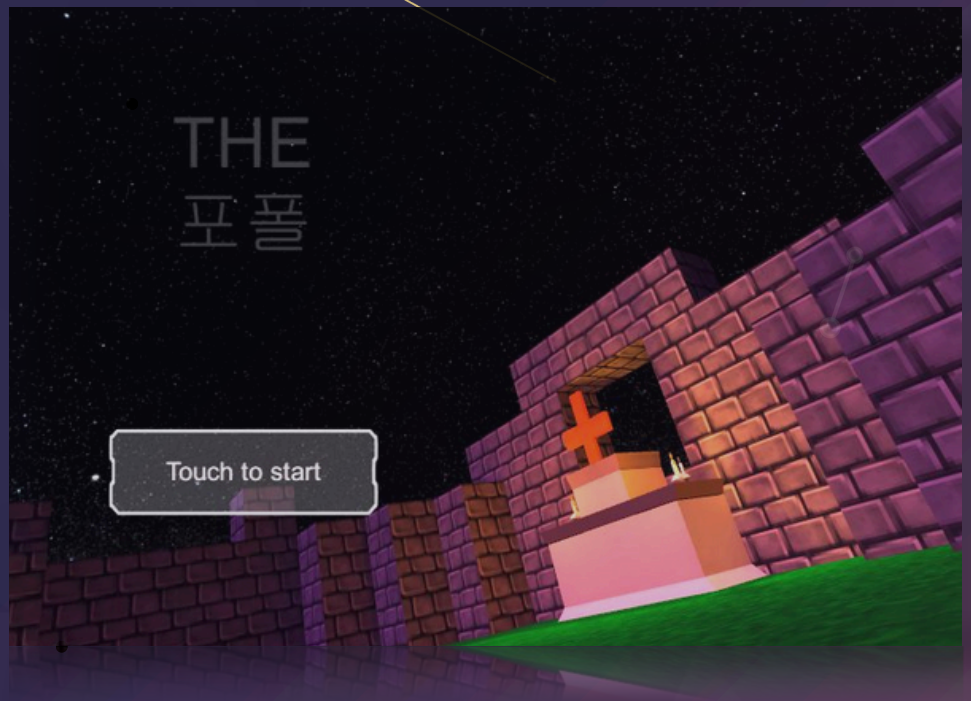
ユニティポートフォリオ

キム・デヨン
eodud1992@gmail.com

はじめに



ジャンル	アクションアドベンチャー
製作スタッフ数	1人
製作期間	約3週間
ビデオ	https://youtu.be/jaazfGNK0KE

目次

プレイヤー

操作


PLAYER
一時停止
II
在庫
INVN
カメラの回転角度
◄
►
スキル、攻撃、ダッシュ
ジョイスティック
SKILL3
SKILL2
DASH
ATTACK
SKILL1

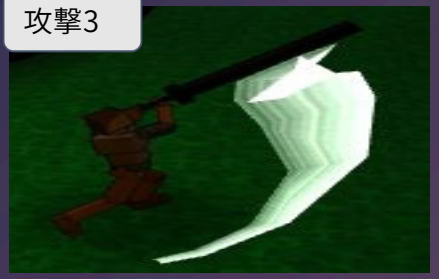
プレイヤー

攻撃

攻撃1

攻撃2

攻撃3



攻撃ボタン

連打で

コンボ攻撃が

可能

スキルセット

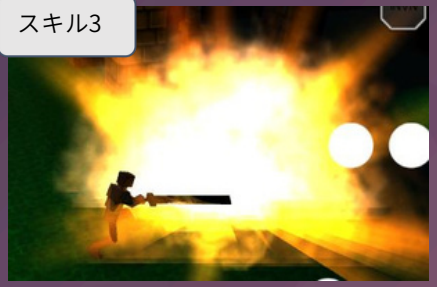
スキル1

前方に球状の炎を発射

スキル2

6方向に雷を放つ

スキル3

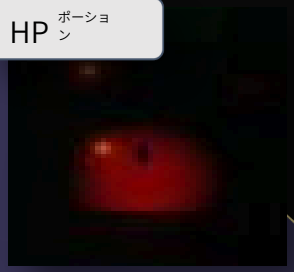


前方に爆弾を detonating し、周囲の敵に損傷を与える。

プレイヤー

アイテム

HP ポーション



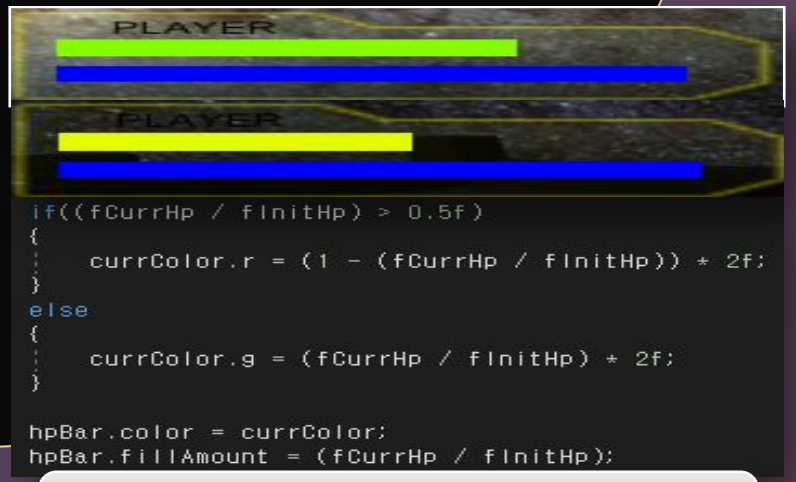
プレイヤーの
体力を回復

スタミナポーション

プレイヤーの
体力 回復

鍵

閉じて通行
できない
ドアを開ける




```
if((fCurrHp / fInitHp) > 0.5f)
{
    currColor.r = (1 - (fCurrHp / fInitHp)) * 2f;
}
else
{
    currColor.g = (fCurrHp / fInitHp) * 2f;
}

hpBar.color = currColor;
hpBar.fillAmount = (fCurrHp / fInitHp);
```

HPおよびSPはFillAmountで表現されます。
色を補間し、HP量に応じて色を調整する。

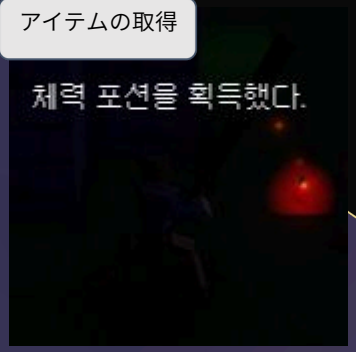
鍵を用いて扉を施錠する

プレイヤ

ー

在庫

アイテムの取得




```
public Stack<ItemInfo> slot;
```

アイテムを保存するスタック

```
public void AddItem(ItemInfo item)
{
    slot.Push(item);
    UpdateInfo(true, item.itemData.defaultImg);
}
```

入手したアイテムをスタックに保存し、
各アイテムに設定した2D画像を表示する。

在庫

入手した
アイテムは
インベントリに
保存されます
（複数の場合は
個別に表示
されます）

モンスター

プレイヤーが不在のとき


```
public List<Transform> wayPointList;
```

侦察ポイントを保存する

```
iNextIdx = ++iNextIdx % wayPointList.Count;
```

次の侦察地点を探す

```
void MoveWayPoint()
{
    if (agent.isPathStale)
        return;

    agent.destination = wayPointList[iNextIdx].position;
    agent.isStopped = false;
}
```

パスが有効であることを確認した後に移動します。

設定されたポイントに戻る

モンスター

プレイヤーを発見したとき

スケルトン

武器による近接攻撃

プレイヤーを追跡し、攻撃する

バット

4方向に遠距離攻撃を行う

```
if(Time.time >= fNextAttack)
{
    Attack();
    fNextAttack = Time.time + fAttackRate + Random.Range(0f, 2f);
}
```

攻撃周期をランダムに設定する

```
var _bullet = bulletPool[i];
if (_bullet != null)
{
    _bullet.transform.position = FirePos_Center.position;
    _bullet.transform.rotation = FirePos_Center.rotation;
    _bullet.SetActive(true);
}
```

オブジェクトプールに生成した弾丸を
アクティブ設定を使用して有効化/無効化

ボス

プレイヤー追跡

衝突体
(頭,体など)

プレイヤー
を検出して
攻撃するための視野

```
public bool bIsViewPlayer()
{
    bool bIsView = false;
    RaycastHit hit;

    Vector3 vecDir = (trPlayer.position - trEnemy.position).normalized;

    if(Physics.Raycast(trEnemy.position, vecDir, out hit, fViewRange, iLayerMask))
    {
        bIsView = (hit.collider.CompareTag("Player"));
    }

    return bIsView;
}
```

レイキャストを利用して
プレイヤーの位置を特定

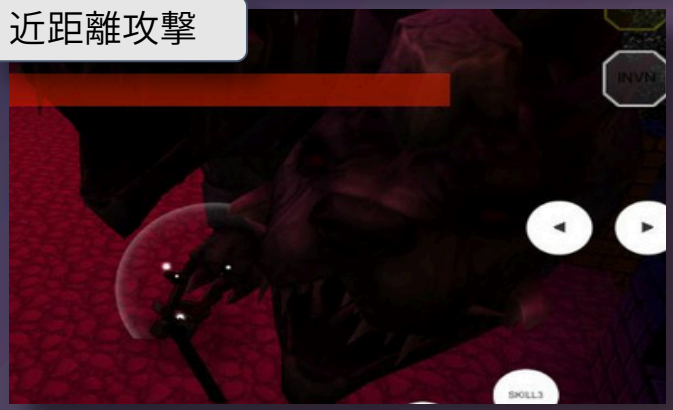
ボス

攻撃およびパターン

プレイヤーが遠い場合

遠距離攻撃

近距離攻撃



```
switch (eState)
{
    case State.IDLE:
        anim.SetBool(iHashChase, false);
        bossAttack.bIsAttack = false;
        bossAttack.bIsFireAttack = false;
        break;
    case State.CHASE:
        anim.SetBool(iHashChase, true);
        bossAttack.bIsAttack = false;
        bossAttack.bIsFireAttack = false;
        break;
    case State.ATTACK:
        anim.SetBool(iHashChase, false);
        bossAttack.bIsAttack = true;
        bossAttack.bIsFireAttack = false;
        break;
    case State.FIRE_ATTACK:
        anim.SetBool(iHashChase, false);
        bossAttack.bIsAttack = false;
        bossAttack.bIsFireAttack = true;
        break;
```

状態に応じてパターンを選定する

パズル

順に火を灯す

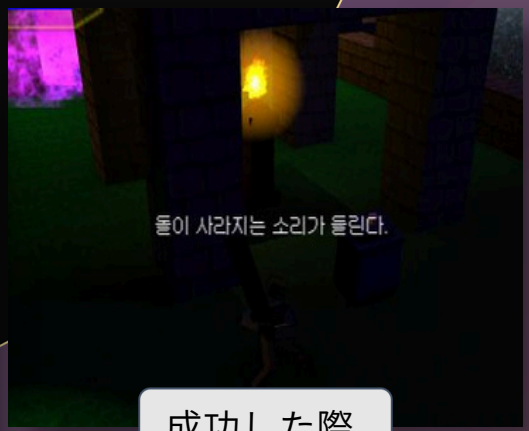
通路を妨げている障害物

順次点灯し、消灯することが可能です。

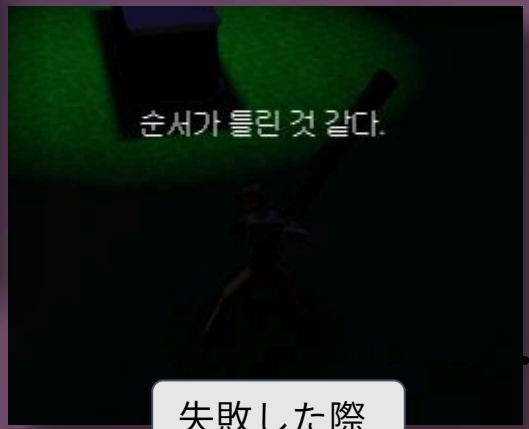
```
for(int i = 0; i < iTriggerCnt; i++)
{
    if(list[i].Equals(iCorrectIdx[i]) == false)
    {
        for(int j = 0; j < iTriggerCnt; j++)
        {
            list.Remove(j);
            GameManager.instance.GetFLTrigger(j).bIsFireLit = false;
        }

        GameManager.instance.TurnOffAllOfFire();
        SoundManager.instance.PlayAudio("Wrong", "SE");
        SetText("순서가 틀린 것 같다.Wn");
        return;
    }
}
```

火をつけた順序を確認し、
一致しない場合は火を消す。



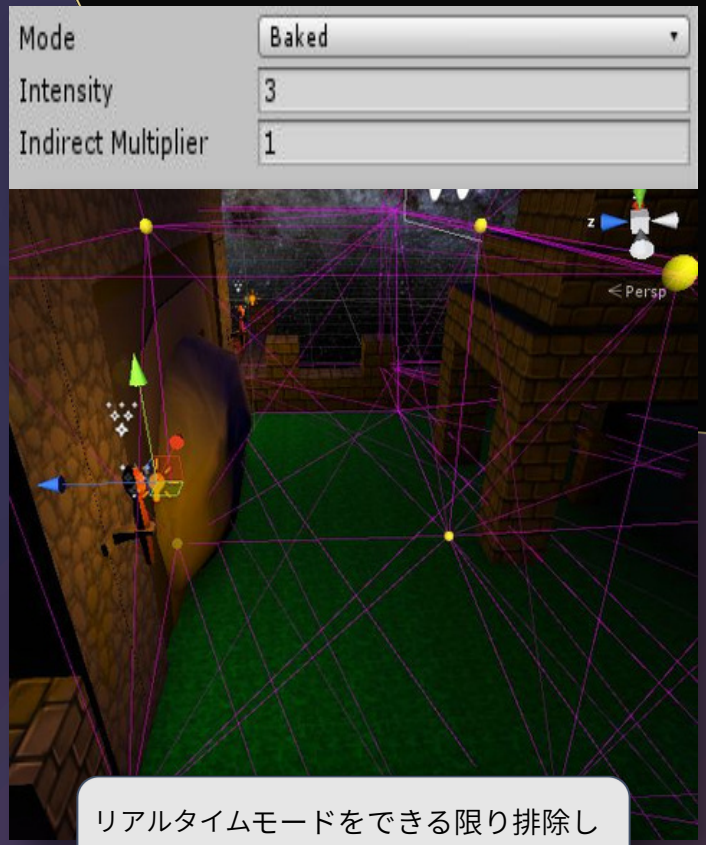
成功した際



失敗した際

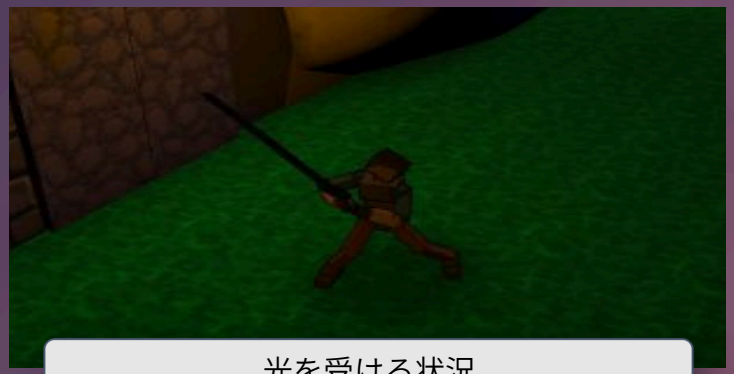
その他

ライト



リアルタイムモードをできる限り排除し
ライトプローブを活用

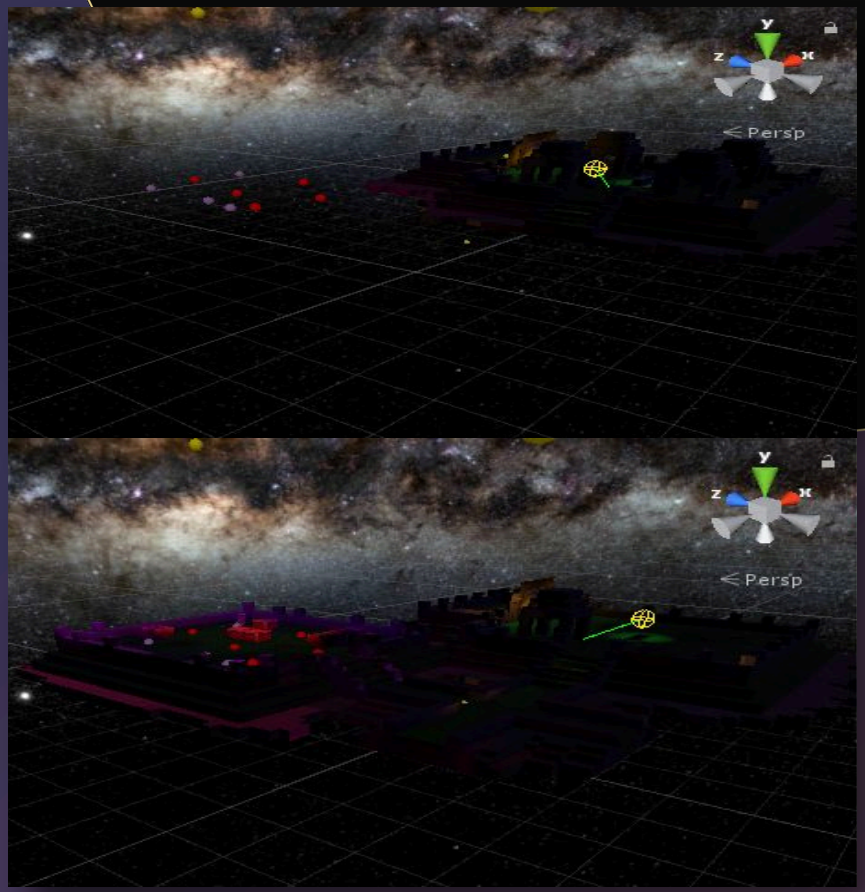
光を受けない状況



光を受ける状況

その他

オクルージョンカリング



カメラに映る
領域だけ表示します。

ありがとうございます。