メンタル鋼のプログラマー

名前　うえだ れんか
上田 蓮果

学校　福岡情報ITクリエイター専門学校

趣味　ゲーム/イラスト制作/ゲーム実況を見る

スキル

Visual Studio

C/C++ 2年
C# 1年

Unity

1年

GitHub

2年

Photoshop

2年

Illustrator

1年

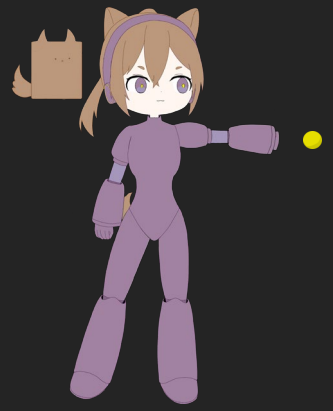
Blender

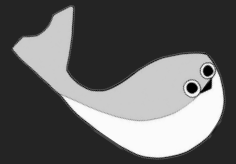
半年

Maya

半年

目次





Uedaが往くRE

▼GitHub
https://github.com/renka1225/UedaGaYuku_Re.git

▼プレイ動画
https://youtu.be/V4ljvildwcI

ジャンル	3Dアクションアドベンチャー
制作期間	2024年9月下旬～ (現在制作中)
制作人数	1人
開発環境	C++/Dxライブラリ
作品概要	前作の「Uedaが往く」を拡張し、ゲーム性を変更しました。 敵と1対多数で戦えたり、武器で攻撃できるようになどの 要素を追加しました。

Uedaが往くRE

ゲーム概要

街中にいる敵を倒してお金を稼ぐ

プレイヤーを強化する

ラスボスを倒す

Uedaが往くRE

技術紹介①　敵AI

敵の行動は「優先度」をもとに決定しています。
優先度は、プレイヤーとの距離や現在の状態、 他の敵の行動などを参照して決めています。
優先度を基に数フレームごとにランダムで行動を決定しています。

一度に攻撃できる敵の数を制限することで、
多数の敵に同時に攻撃されることを防いでいます。

外部ファイルを使用して、 敵の種類ごとに異なる優先度を
設定できるようにしました。

	A	B	C	D	E	F	G	H
1	キャラクターID	idle	walk	run	punch	kick	avoid	guard
2	DEFAULT	25	5	3	5	4	6	5
3	TUTO	20	6	3	5	3	8	10
4	BOSS	10	8	5	20	10	10	3
5	BOB	15	8	5	5	3	8	10
6	SATO	15	8	5	5	3	8	10
7	ABE	15	8	5	5	3	8	10

enemyAI.csv参照 ☒

▼EnemyAI.cpp参照

```
    }

    // バトル中プレイヤーから離れた場合
    if (pPlayer.GetIsBattle() && dist >= kMaxChaseRange)
    {
        // バトルを終了状態にする
        pPlayer.SetIsBattle(false);
        return;
    }

    // プレイヤーに一定距離近づいた場合
    if (dist < kMinApproachRange)
    {
        AddRandomPriority();
        // 移動しない
        AddIdlePriority();
    }
    // プレイヤーとの距離が離れている場合
    else if (dist > kMinChaseRange)
    {
        // プレイヤーに近づく
        AddMovePriority();
    }
    else
    {
        // 他の敵が攻撃中の場合
        if (attackEnemyNum >= kMaxAttackEnemyNum)
        {
            UpdatePriority(EnemyStateBase::EnemyStateKind::kIdle, m_acitonProbaility[m_charaId].idle);
            UpdatePriority(EnemyStateBase::EnemyStateKind::kWalk, m_acitonProbaility[m_charaId].walk);
            UpdatePriority(EnemyStateBase::EnemyStateKind::kRun, m_acitonProbaility[m_charaId].run);
        }
        // 1体も攻撃していない場合
        else if (attackEnemyNum == 0)
        {
            // 攻撃優先
            AddAttackPriority();
        }
        else
        {
            // ランダム
            AddRandomPriority();
        }

        // プレイヤーが攻撃中の場合
        if (pPlayer.GetIsAttack())
        {
            UpdatePriority(EnemyStateBase::EnemyStateKind::kGuard, m_acitonProbaility[m_charaId].guard);
            UpdatePriority(EnemyStateBase::EnemyStateKind::kAvoid, m_acitonProbaility[m_charaId].avoid);
        }
        // プレイヤーが回避中の場合
        else if (playerState == AnimName::kAvoid)
        {
            AddMovePriority();
            AddAttackPriority();
        }
        // プレイヤーがガード中の場合
        else if (playerState == AnimName::kGuard)
        {
            AddMovePriority();
        }
        // その他
        else
        {
            // ランダム
            AddRandomPriority();
        }
    }
```

Uedaが往くRE

技術紹介② ステートパターン

Stateパターンを使用して、キャラクターの状態を管理しています。
プレイヤーと敵のそれぞれのStateBaseクラスで簡単に状態の切り替えができるようにしました。

▼PlayerStateBase.cpp参照

```cpp
void PlayerStateBase::Update(const Input& input, const Camera& camera, Stage& stage, Weapon& weapon, std::vector<std::shared_ptr<EnemyBase>> pEnemy)
{
    GetJoypadAnalogInput(&m_analogX, &m_analogY, DX_INPUT_PAD1); // アナログスティックの入力状態
    m_pPlayer->Move(m_moveVec, stage, false);   // 移動情報を反映する

    // 特定の状態中は更新しない
    if (IsStateInterrupt()) return;

    // 移動できない場合
    if (!m_pPlayer->GetIsPossibleMove())
    {
        ChangeStateIdle();
        return;
    }

    // バトル中でない場合
    if (!m_pPlayer->GetIsBattle()) return;

    if (m_pPlayer->GetHp() <= 0.0f)
    {
        ChangeStateDeath();
    }

    // ダメージを受けた場合
    if (m_pPlayer->GetIsOnDamage())
    {
        ChangeStateDamage();
        return;
    }

    // ゲージが最大まで溜まっているか
    bool isSpecial = m_pPlayer->GetGauge() >= m_pPlayer->GetStatus().maxGauge;

    // 掴みのボタンが押された場合
    if (input.IsTriggered(InputId::kGrab))
    {
        ChangeStateGrab(weapon);
        return;
    }

    // 攻撃のボタンが押された場合
    if (input.IsTriggered(InputId::kPunch) || input.IsTriggered(InputId::kKick))
    {
        ChangeStateAttack(input);
        return;
    }
    // ガードのボタンが押された場合
    if (input.IsPressing(InputId::kGuard))
```

▼EnemyStateBase.cpp参照

```cpp
void EnemyStateBase::ChangeState(EnemyStateKind nextState)
{
    if (GetKind() == nextState) return;

    switch (nextState)
    {
    case EnemyStateBase::EnemyStateKind::kWalk:
        ChangeStateWalk();
        break;
    case EnemyStateBase::EnemyStateKind::kRun:
        ChangeStateRun();
        break;
    case EnemyStateBase::EnemyStateKind::kAvoid:
        ChangeStateAvoid();
        break;
    case EnemyStateBase::EnemyStateKind::kPunch:
        ChangeStatePunch();
        break;
    case EnemyStateBase::EnemyStateKind::kKick:
        ChangeStateKick();
        break;
    default:
        break;
    }
}
```

Uedaが往くRE

技術紹介③　武器配置

Unityを使用して武器の配置を行いました。
武器の位置・角度・サイズ・武器名・武器の種類(片手or両手)を
バイナリファイルで書き出し、ゲーム内で読み込みを行っています。

外部ファイルで、武器ごとに耐久力や当たり判定のサイズを
変更できるようにしました。

▼weaponData.csv参照

	A	B	C	D	E	F	G	H	I	J	K
1	武器ID	武器名	耐久力	当たり判定	当たり判定	当たり判定	当たり判定	当たり判定	当たり判定	当たり判定半径	
2	bar_1	bar	4	0	5	0	5	0	5	8	
3	bar_2	bar	4	0	5	0	5	0	5	8	
4	bar_3	bar	4	0	5	0	5	0	5	8	
5	bar_4	bar	4	0	5	0	5	0	5	8	
6	bar_5	bar	4	0	5	0	5	0	5	8	
7	bat_1	bat	4	0	5	0	5	0	5	8	
8	bat_2	bat	4	0	5	0	5	0	5	8	
9	bat_3	bat	4	0	5	0	5	0	5	8	
10	bat_4	bat	4	0	5	0	5	0	5	8	
11	bat_5	bat	4	0	5	0	5	0	5	8	
12	pipe_1	pipe	5	0	5	0	5	0	5	8	
13	pipe_2	pipe	5	0	5	0	5	0	5	8	

▼Weapon.cpp参照

```
200  void Weapon::LoadLocationData()
201  {
202      m_loadLocationData = FileRead_open((kWeaponFileName + "data.loc").c_str());
203
204      int dataCnt = 0; // データ数
205      FileRead_read(&dataCnt, sizeof(dataCnt), m_loadLocationData);
206      m_locationData.resize(dataCnt);
207
208      for (auto& loc : m_locationData)
209      {
210          // オブジェクト名をロード
211          byte nameCnt = 0;
212          FileRead_read(&nameCnt, sizeof(nameCnt), m_loadLocationData);
213          loc.name.resize(nameCnt);
214
215          // MEMO:loc.name.data()の部分はC++20だとエラーにならない
216          FileRead_read(loc.name.data(), sizeof(char)* loc.name.size(), m_loadLocationData);
217
218          // タグをロード
219          byte tagCnt = 0;
220          FileRead_read(&tagCnt, sizeof(tagCnt), m_loadLocationData);
221          loc.tag.resize(tagCnt);
222          FileRead_read(loc.tag.data(), sizeof(char) * loc.tag.size(), m_loadLocationData);
223
224          // 座標情報
225          FileRead_read(&loc.pos, sizeof(loc.pos), m_loadLocationData);
226          loc.initPos = loc.pos;
227          // 回転情報
228          FileRead_read(&loc.rot, sizeof(loc.rot), m_loadLocationData);
229          loc.initRot = loc.rot;
230          // スケール情報
231          FileRead_read(&loc.scale, sizeof(loc.scale), m_loadLocationData);
232      }
233      FileRead_close(m_loadLocationData);
234
235      // モデルのパスを設定
236      for (auto& loc : m_locationData)
237      {
238          if (m_objHandle.find(loc.name) == m_objHandle.end())
239          {
240              std::string modelPath = kWeaponFileName + loc.name + ".mv1";
241              int modelHandle = MV1LoadModel(modelPath.c_str());
242              m_objHandle[loc.name] = modelHandle;
243          }
244      }
245  }
```

Uedaが往くRE

技術紹介④ コンボ攻撃

先行入力の実装を行い、最大3コンボのパンチ攻撃ができるようにしました。

攻撃中の特定のフレーム内の入力状態を保存し、
入力があった場合のみ次の攻撃へ遷移する仕組みになっています。

▼Player.cpp参照

```cpp
bool Player::CheckCommand(const std::vector<std::string>& command, const std::vector<CommandInput>& inputLog)
{
    // ログがない場合はチェックしない
    if (inputLog.empty()) return false;

    // コマンドの最終入力を確認する
    int index = command.size() - 1;
    // 最新の入力時間
    int currentTime = inputLog.back().frameCount;

    // MEMO:rbegin()は逆イテレータを返す
    for (auto it = inputLog.rbegin(); it != inputLog.rend(); it++)
    {
        if (it->button == command[index])
        {
            // 入力が確認できた場合
            index--;
            // すべてのコマンドが一致している場合
            if (index < 0) return true;
        }
    }
    return false;
}
```

▼PlayerStateAttack.cpp参照

```cpp
    // アニメーション時間の更新
    m_animEndTime--;
    if (m_animEndTime > 0.0f) return;

    // パンチコマンドが入力されている場合
    if (m_pPlayer->CheckCommand({ InputId::kPunch, InputId::kPunch }, m_pPlayer->GetInputLog()))
    {
        // パンチに移行
        if (m_attackKind == AnimName::kKick)
        {
            Init(AnimName::kPunch1);
            return;
        }

        // 2コンボ目に移行する
        if (m_attackKind == AnimName::kPunch1)
        {
            Init(AnimName::kPunch2);
            return;
        }
        // 3コンボ目に移行する
        else if (m_attackKind == AnimName::kPunch2)
        {
            Init(AnimName::kPunch3);
            return;
        }
        // 現在最終コンボの場合
        else if (m_attackKind == AnimName::kPunch3)
        {
            // 攻撃を終了する
            m_isAttackEnd = true;
            return;
        }
    }
    // コンボ入力がない場合
    else
    {
        // 攻撃を終了する
        m_isAttackEnd = true;
    }
}
```

Uedaが往く

▼GitHub
https://github.com/renka1225/UedaGaYuku.git

▼プレイ動画
https://youtu.be/eWVjdxmv50E

ジャンル	3D格闘アクション
制作期間	2024年7月上旬～2024年9月上旬(約400時間)
制作人数	1人
開発環境	C++/Dxライブラリ
作品概要	龍が如くをもとに制作 敵と1対1で戦い、3連勝するとクリア

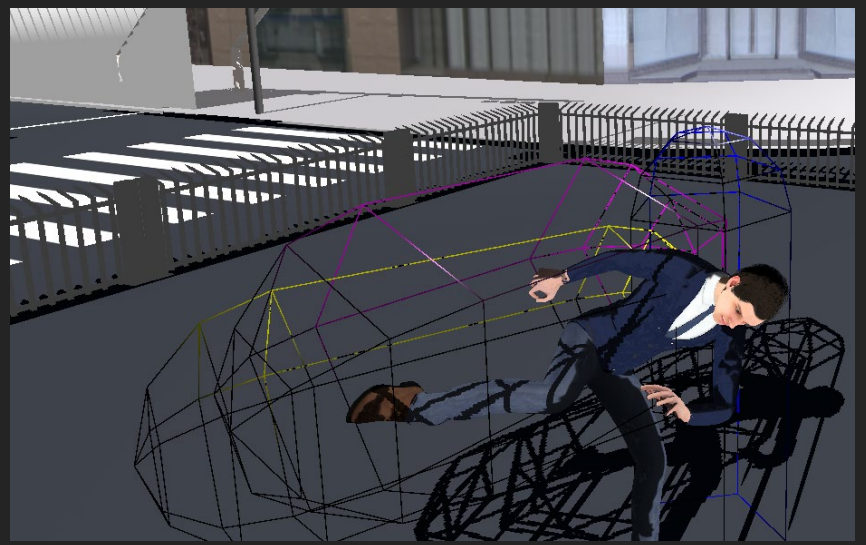
Uedaが往く

技術紹介①　当たり判定

行列を使用し、キャラクターの角度と座標をもとに
当たり判定の位置を調整しています。

外部データで、キャラクターごとの当たり判定情報を
簡単に調節できるようにしました。

▼デバッグ画面



▼Collision.csv参照

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Character	BodyHeigl	BodyRadiu	AimRadiu	LegRadius	ArmStartP	ArmStartP	ArmStartP	ArmEndtP	ArmEndPo	ArmEndPo	LegStartP	LegStartP	LegStartP	LegEndtP	LegEndtP	LegEndtPosZ
2	Player	43	10	15	12	5	35	10	-5	0	28	3	30	10	3	0	36
3	EnemyTut	45	12	4	5	3	35	0	-3	0	25	3	30	10	3	5	30
4	Ninja	45	12	12	5	3	35	0	-3	0	25	3	30	10	3	0	28
5	Chef	45	10	7	8	3	35	0	-3	0	25	3	30	10	3	5	25
6	Abe	45	9	10	8	3	35	0	-3	0	25	3	30	10	3	5	28

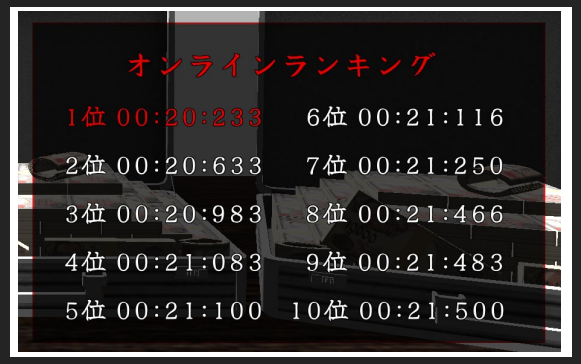
▼CharacterBase.cpp参照

```
70  /// <summary>
71  /// 当たり判定位置更新
72  /// </summary>
73  void CharacterBase::UpdateCol()
74  {
75      // キャラクターの向きをもとに当たり判定の位置を調整する
76      MATRIX rotationMatrix = MGetRotY(m_angle);
77
78      // プレイヤー全体の当たり判定位置を更新
79      m_col.bodyTopPos = VAdd(m_pos, (VTransform(VGet(0.0f, m_colInfo.bodyHeight, 0.0f), rotationMatrix)));
80      m_col.bodyBottomPos = m_pos;
81
82      // 腕の当たり判定位置を更新
83      m_col.armStartPos = VAdd(m_pos, (VTransform(m_colInfo.armStartPos, rotationMatrix)));
84      m_col.armEndPos = VAdd(m_col.armStartPos, (VTransform(m_colInfo.armEndPos, rotationMatrix)));
85
86      // 脚の当たり判定位置を更新
87      m_col.legStartPos = VAdd(m_pos, (VTransform(m_colInfo.legStartPos, rotationMatrix)));
88      m_col.legEndPos = VAdd(m_col.legStartPos, (VTransform(m_colInfo.legEndPos, rotationMatrix)));
89  }
```

Uedaが往く

技術紹介② オンラインランキング

レンタルサーバーとPHPを用いてランキングの実装を行いました。
DxLibのネットワーク機能を利用して、
リアルタイムでランキングを更新できるようにしました。



▲クリア画面

▼PHP

```php
<?php
// データベースの作成と接続(既にある場合は接続のみ)
$db = new PDO('sqlite:rankingStage1.db');

// データの受け取り
$getClearTime = $_GET['clearTime'];

// 新しいクリア時間をデータベースに挿入
$que = "INSERT INTO Ranking (clearTime) VALUES ('{$getClearTime}')";
$db->query($que);

// データを昇順にソートする
$que ="SELECT * FROM Ranking ORDER BY clearTime";
//連想配列にする
$value = $db->query($que)->fetchAll();

// テーブルデータの書き直し
$rank=1;
foreach ($value as $row)
{
    $clearTime = $row['clearTime'];

   // 同じクリア時間の場合はスキップ
    if ($clearTime == $previousClearTime) continue;

    $que = "UPDATE Ranking SET rank = {$rank} WHERE clearTime = '{$clearTime}'";
    echo $que . "\n";

    //$db->query($que);
    $rank++;
}
?>
```

```php
<?php
// データベースの作成と接続(既にある場合は接続のみ)
$db = new PDO('sqlite:rankingStage1.db');

// ランキング上位10件を取得するクエリ
$que = "SELECT * FROM Ranking ORDER BY clearTime LIMIT 10";
$value = $db->query($que)->fetchAll(PDO::FETCH_ASSOC);

// JSON形式で結果を返す
echo json_encode($value);
?>
```

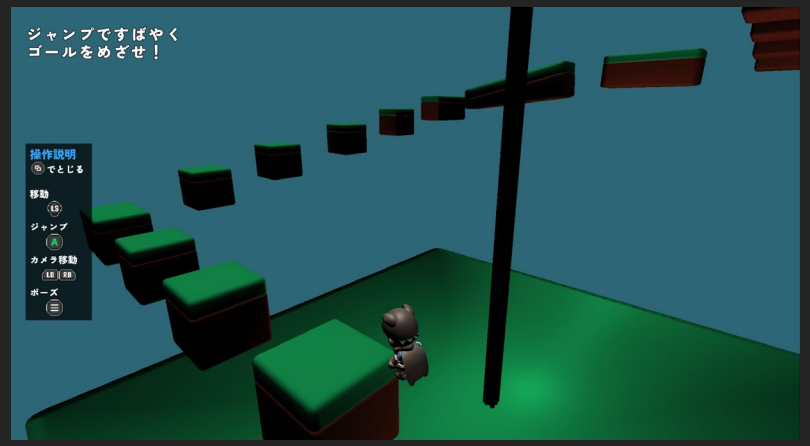
▼Ranking.cpp参照

```cpp
/// <summary>
/// Http通信のGet命令を送る
/// </summary>
/// <param name="domain">ドメイン名</param>
/// <param name="uri">URI</param>
/// <returns>ランキングを取得</returns>
std::string Ranking::HttpGet(const char* domain, const char* uri)
{
    char HttpCmd[128] = "";     // Http通信を作成するための変数

    // DxLibの不要な機能をoffにする
    SetUseDXNetWorkProtocol(false);

    // DNSからドメイン名でIPアドレスを取得
    GetHostIPbyName(domain, &Ip);

    // 通信を確立
    NetHandle = ConnectNetWork(Ip, kPortNum);

    // 確立が成功した場合のみ中の処理をする
    if (NetHandle != -1)
    {
        //Http命令の作成
        sprintf_s(HttpCmd, "GET %s HTTP/1.1¥nHost: %s¥n¥n", uri, domain);
#ifdef _DEBUG
        //DrawFormatString(0, 60, 0xffffff, "HttpCmd:¥n%s", HttpCmd);
#endif // _DEBUG

        // データ送信(http命令を送る)
        NetWorkSend(NetHandle, HttpCmd, static_cast<int>(strlen(HttpCmd)));

        // データがくるのを待つ
        while (!ProcessMessage())
        {
            // 取得していない受信データ量を得る
            DataLength = GetNetWorkDataLength(NetHandle);

            // 取得してない受信データ量が-1じゃない場合はループを抜ける
            if (DataLength != -1)
            {
                break;
            }
        }

        // データ受信
        NetWorkRecv(NetHandle, StrBuf, kDataSize);    // データをバッファに取得

        // 接続を断つ
        CloseNetWork(NetHandle);
    }


#ifdef _DEBUG
    //取得したIPアドレスの確認
    //DrawFormatString(0, 20, 0xffffff, "IpAdress:%d,%d,%d,%d", Ip.d1, Ip.d2, Ip.d3, Ip.d4);
    //DrawFormatString(0, 40, 0xffffff, "NetHandle:%d", NetHandle);
#endif // _DEBUG

    return StrBuf;
```

JumpUp!

▼GitHub
https://github.com/renka1225/JumpUp.git

▼プレイ動画
https://youtu.be/4LY0KWWuWIc



ジャンル	3Dアクション
制作期間	2024年5月下旬～2024年6月中旬
制作人数	1人
開発環境	C++/Dxライブラリ
作品概要	ジャンプで頂上にあるゴールを目指す

頑張ったこと・学んだこと

3Dのジャンプ挙動や物理挙動
3Dモデルのアニメーション
マップの当たり判定
カメラ挙動

HOPPINGRACE

▼GitHub
https://github.com/renka1225/HOPPINGRACE.git

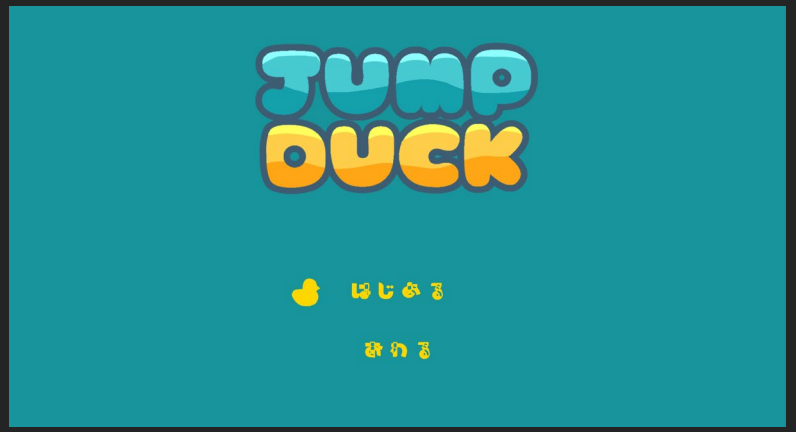
▼プレイ動画
https://youtu.be/S8kh_nkfH-A

ジャンル	2.5Dアクション
制作期間	2024年4月下旬～2024年5月中旬
制作人数	1人
開発環境	C++/Dxライブラリ
作品概要	カービィ64の「けんけんレース」、マリオパーティDSの「ぴょんぴょんレース」をもとにした作品 正しいボタンを押してゴールを目指す
頑張ったこと・学んだこと	3Dモデルの使用方法 カメラの理解 DxLibの3Dの機能の理解

JUMPDUCK

▼GitHub
https://github.com/renka1225/JUMPDUCK.git

▼プレイ動画
https://youtu.be/gHdaRHInnKA

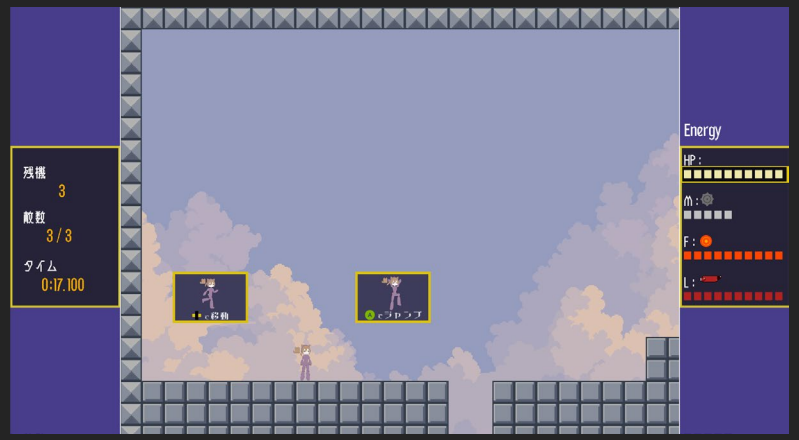


ジャンル	2.5D横スクロールジャンプアクション
制作期間	2024年3月上旬～2024年4月上旬
制作人数	1人
開発環境	C++/Dxライブラリ
作品概要	90秒間ジャンプで敵をよけ続ける 敵にあたったらゲームオーバー
頑張ったこと・学んだこと	外部ファイルでの敵位置管理 3Dモデルの使用方法 カメラの理解

DOGMAN

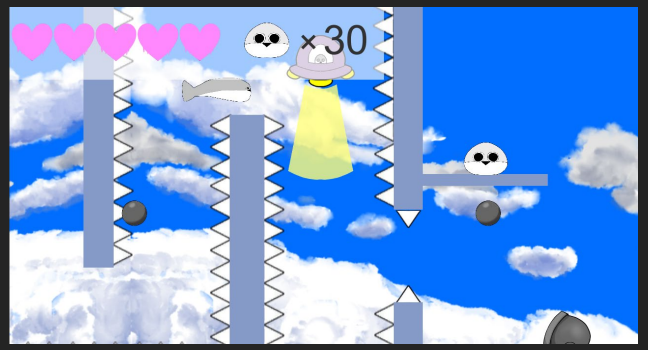
▼GitHub
https://github.com/renka1225/DOGMAN.git

▼プレイ動画
https://youtu.be/K3CmvOp4BxQ



ジャンル	2D横スクロールアクション
制作期間	2023年12月～2024年2月(約320時間)
制作人数	1人
開発環境	C++/Dxライブラリ
作品概要	ロックマン2をオマージュした作品 敵をすべて倒したらクリア 4種類の武器を使い分けて短時間でクリアを目指す
頑張ったこと・学んだこと	C++，DxLibの基本の理解 マップチップの当たり判定 マップスクロール ジャンプの実装

サカバンバスピスの大冒険!



ジャンル	2D横スクロールアクション
制作期間	2023年7月～8月
制作人数	3人
開発環境	Unity
担当箇所	空ステージ(3ステージ)実装 敵の当たり判定実装 敵グラフィック作成 タイトルロゴ、エンディング作成
頑張ったこと・学んだこと	チーム制作の経験 Unityの基本性能の理解