

# 令和 4 年度卒業研究報告書

画像認識によるアクションバトルゲームの開発

情報技術科 天沼 優作

宇部 駿

加藤 晴太

指導教員 安倍 春菜

# 内容

令和 4 年度卒業研究報告書 .....	0
画像認識によるアクションバトルゲームの開発.....	0
第 1 章　はじめに.....	4
第 2 章　研究概要.....	5
2.1 開発環境 .....	5
2.2 使用したライブラリについて.....	5
2.2.1 Unity .....	5
2.2.2 Blender.....	5
2.2.3 C#.....	6
2.2.4 Python .....	6
2.2.5 MediaPipe .....	6
2.3 環境構築 .....	6
第 3 章　ゲーム構成.....	8
3.1 ゲーム企画.....	8
3.2 ゲームの仕様 .....	8
3.3 ゲーム詳細.....	10
3.3.1 情報技術科フロア .....	10
3.3.2 電子技術科フロア .....	11
3.3.3 実習棟エリア.....	12
3.4 キャラクター詳細.....	17
第 4 章　システム構成 .....	18
4.1 基本的なゲームシステム .....	18

4.1.1	ターン制バトルの仕組み .....	19
4.1.2	マップ探索の仕組み.....	21
4.2	ジェスチャータイム .....	24
4.2.1	Mediapipe による手の座標検出.....	25
4.2.2	Unity と Python の連動 .....	26
4.2.3	ハンドトラッキング(ゲーム起動時の処理).....	28
4.2.4	ハンドトラッキング(ジェスチャータイム時の処理) .....	33
第 5 章	マップ作成.....	44
5.1	マップ概要.....	44
5.2	使用 Asset.....	45
第 6 章	キャラクターの作成 .....	47
6.1	使用技術 .....	47
6.1.1	Blender.....	47
6.1.2	Unity .....	47
6.2	制作過程 .....	47
6.3	キャラクターの作成方法.....	48
6.3.1	メッシュの作成方法.....	48
6.3.2	アーマチュアの接続.....	51
6.3.3	アニメーションの作成 .....	53
6.3.4	Unity へのエクスポート .....	55
6.3.5	アニメーションの設定 .....	58
6.3.6	Animator の設定 .....	59
第 7 章	エフェクトの作成.....	61

7.1 使用技術 .....	61
7.1.1    Blender.....	61
7.1.2    Unity .....	61
7.2 制作過程 .....	61
7.3 エフェクトの作成方法 .....	61
7.3.1    Blender でのテクスチャ作成 .....	61
7.3.2    メッシュの作成 .....	62
7.3.3    パーティクルの設定.....	62
7.3.4    シェーダーグラフの作成 .....	70
7.3.5    エフェクトの Prefab 化 .....	72
第 8 章 ED,PV の作成.....	78
8.1 使用技術 .....	78
8.1.1    ED ムービーの作成.....	78
8.1.2    紹介映像の作成 .....	79
8.1.3    実況動画の作成 .....	79
第 9 章 おわりに.....	80

# 第1章　はじめに

前期の授業で在学生に入学時の当校の認知度に関するアンケートを実施した際、受験した学科については知っているが、他科の授業内容について知られていないことがわかった。

しかし、当校では当校を受験した学生に第1希望の学科の他に第2希望を採る制度を行っている。この際に受験する学科以外の科も知ることで受験の幅が広がると考えた。

また、当校には産業を支える素晴らしい科が沢山あることから、在学生にも他科を知るきっかけになるものを作りたいと考えた。

そこで、誰でも親しみやすいゲームに目を向け、簡単に他科の雰囲気を知ってもらいたいと考えた。

ゲーム作成にあたり、近年注目されている画像認識に着目した。昨年の卒業研究にも利用されていた Mediapipe によるハンドトラッキングを応用したいと考えこのテーマを選定した。

## 第2章 研究概要

### 2.1 開発環境

本研究の開発環境を以下の表1に示す。

OS	Windows11
仕様ゲームエンジン	Unity2019.4
仕様モデリングソフト	Blender
使用言語	C#・Python
仕様ライブラリ	MediaPipe

表1 開発環境

### 2.2 使用したライブラリについて

#### 2.2.1 Unity

Unityは,Unity Technologiesが開発,販売している,IDEを内蔵するゲームエンジン.主にC#を用いたプログラミングでコンテンツの開発が可能.

PC(Windows, macOS)だけでなくモバイル(iOS, Android)やウェブブラウザ(WebGL),家庭用ゲーム機(PlayStation 4, Xbox One, Nintendo Switch等)といったクロスプラットフォームに対応しており,VR/AR/MR機器向けのコンテンツ開発にも対応している.

#### 2.2.2 Blender

Blender(ブレンダー)はオープンソースの統合型 3DCG 製作, 2D アニメーション製作, VFX 向けデジタル合成, 動画編集ソフトウェアである.

### 2.2.3 C#

C#は、マイクロソフトが開発したプログラミング言語である。そのためオブジェクト指向が取り入れられている。また、C#はプログラミング言語の中でも比較的使いやすく、初心者にとって易しいのが特徴であり、「Unity」と相性が良いことで有名である。

### 2.2.4 Python

Python(パイソン)はインタープリタ型の高水準汎用プログラミング言語である。

### 2.2.5 MediaPipe

Mediapipe は、Google が提供しているライブメディアやストリーミングメディア向けの ML(Machine Learning)ソリューション。顔や手、ポーズのリアルタイム検出を行えるライブラリである。

## 2.3 環境構築

### 1. Python3.10 をインストール

ゲームの exe ファイルと同じフォルダにある「python-3.10.9-amd64.exe」を開く。

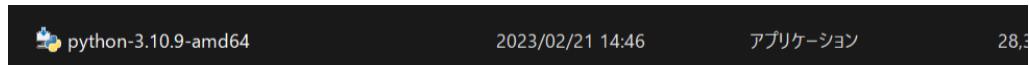


図 1 フォルダ画面



図 2 インストール画面

2. 「Add python.exe to PATH」にチェックを入れ, Install Now を押す.

インストールが終わったら close を押す.

動作に必要なモジュールをインストール

ゲームの exe ファイルと同じフォルダにある「PythonSetup.bat」を開く.



図 3 フォルダ画面

# 第3章 ゲーム構成

## 3.1 ゲーム企画

ゲーム作成にあたって、誰でも気軽に遊べるように、パソコン1台でプレイできるものがいいと考えた。ゲームの方向性としては、ゲーム内のキャラクターとリンクし、実際に自分の体を動かして遊べるバトルゲームを作りたいと考えた。しかし、調べていく中で、「LeapMotion」や「VR機器」など、他の機材を用いた開発がほとんどであった。そこで、去年の卒業研究で取り組まれていた画像認識に着目し、ハンドトラッキングを応用した、まだ世にない新しいアクションバトルゲームの開発がしたいと考えた。

今回の卒業研究では3人で1つのゲームを作るということで、ただのアクションバトルゲームでは研究内容として不十分になってしまふ。そこで、アクションバトルを含む、ロールプレイングゲーム（以下「RPG」と称する）にすることで、オープニング・ストーリー・エンディングなど、ゲーム全体の制作に携わることができ、研究内容のボリュームが増えることから、RPG作成に取り組んだ。

## 3.2 ゲームの仕様

ゲームの内容は、産業技術短期大学校（以下「産技短」と称する）を舞台としたRPGとした。ゲーム内で使用するキャラクターは、今年度行われた当校の学園祭のポスター（図4）をもとに作成を行った。キャラクターが使用する技では、その科にちなんだ技にすることで少しでも雰囲気を掴めるようにした。また、ゲーム内マップは、産技短の各科のフロアを再現し、ゲーム内で産技短を周ることができるようとした。マップ探索時には、その科の学習に紐づけたギミックを導入した。

産技短の紹介を絡ませたオープニングストーリーが始まり,キャラクター選択画面から各科のイメージキャラクターを選択し,マップ選択画面から各科のフロアマップを選択する. 産技短内を探索し,各科のキャラクターと戦闘を行い,次のステージにいるボスを目指す. ボスを無事に倒すことが出来ればクリアとなる.クリア後,エンディングロールが流れ,ゲーム終了となる.

ゲームの流れを以下の図5に示す.



図4 学園祭のポスター

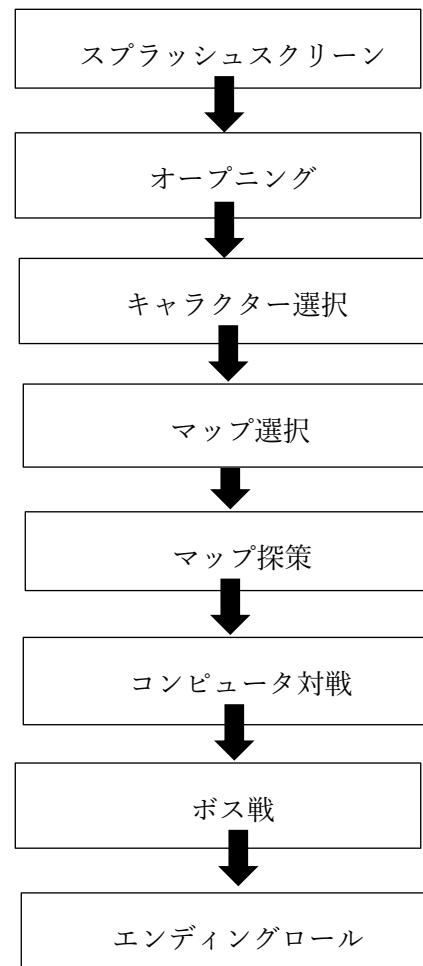


図5 ゲームの流れ

### 3.3 ゲーム詳細

#### 3.3.1 情報技術科フロア

ゲームマップ、「情報技術科フロア」について説明する。

マップはこのようになっている。

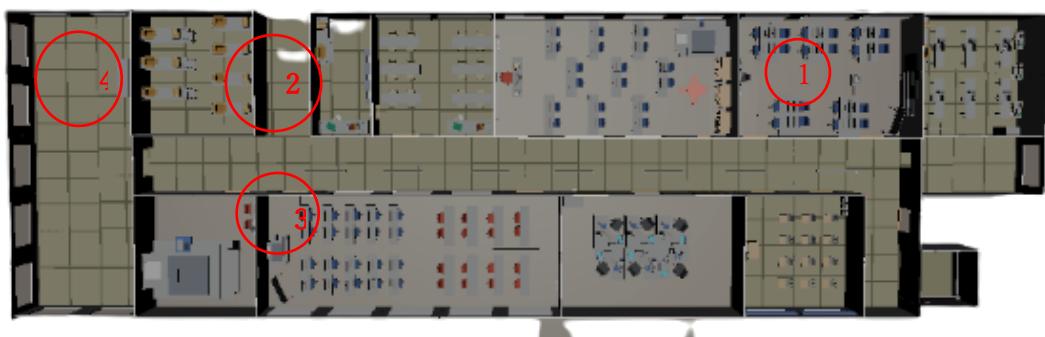


図 6 情報技術科フロア

以下に情報技術科フロアのゲームの流れを説明する。

##### ① キャラクターと会話する

マップ内にいるキャラクターに近づくと  
会話テキストが表示される。その内容から  
ゲーム進めていく。



図 7 エラーとの会話

##### ② マップを探査する

マップ内を探査していくと、鍵が掛けら  
れているドアがある。その手掛かりを探す  
為に、マップ内に潜む敵を見つける。



図 8 鍵付きのドア

### ③ アイティーくんと戦う

敵を見つけることができたら、敵と戦い、  
勝利することで、鍵を入手できる。



図9 アイティーくんとの会話

### ④ ボスステージに向かう

敵との戦いに勝利することで、ボス  
ステージに続くドアへの鍵を入手できる。



図10 鍵の入手

## 3.3.2 電子技術科フロア

続いて、「電子技術科フロア」について説明する。

マップはこのようになっている。



図11 電子技術科フロア

以下に電子技術科フロアのゲームの流れを説明する。

### ① マップを探索する

マップ内を探索していくと,暗号付きのドアがある.大きく「9」と示されており,横には「ニシングスウ」と書かれた紙が貼られている.マップを探索し,暗号の手掛かりを探す.

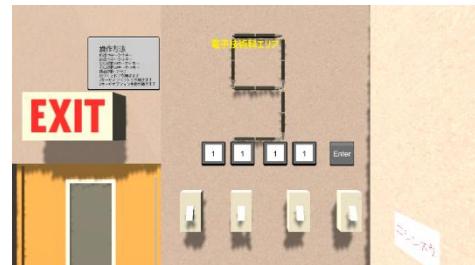


図 1 2 暗号付きのドア

### ② いーていーくんと戦う

敵と戦い勝利することで,バスの居場所を教えてくれるが,暗号の答えを知れないので,もう一度マップを探索し,手掛かりを探す.



図 1 3 いーていーくんとの会話

### ③ 暗号のヒントを得る

マップ内にいる「押露須弧負君」に話かけることで,暗号に繋がるアイテム入手できる.このヒントから暗号(1001)を導き,バスステージに向かう.



図 1 4 押露須弧負君との会話

### 3.3.3 実習棟エリア

最後に「実習棟エリア」について説明する.  
マップはこのようになっている.

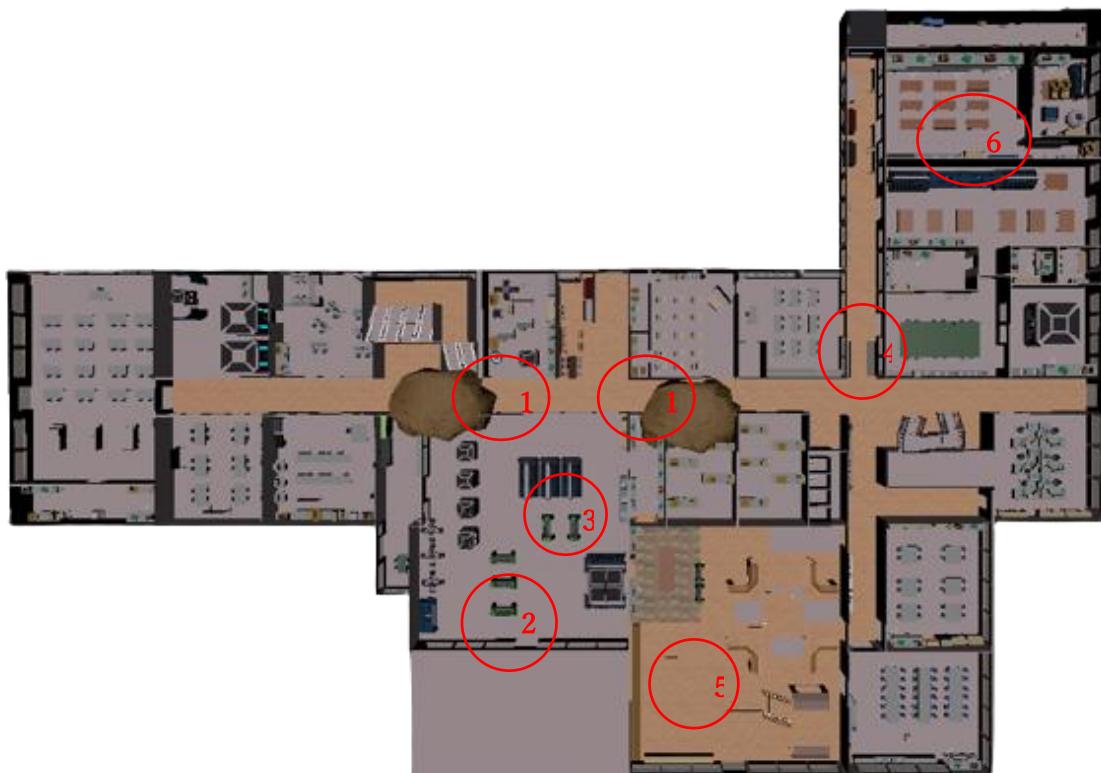


図15 実習棟エリア1階

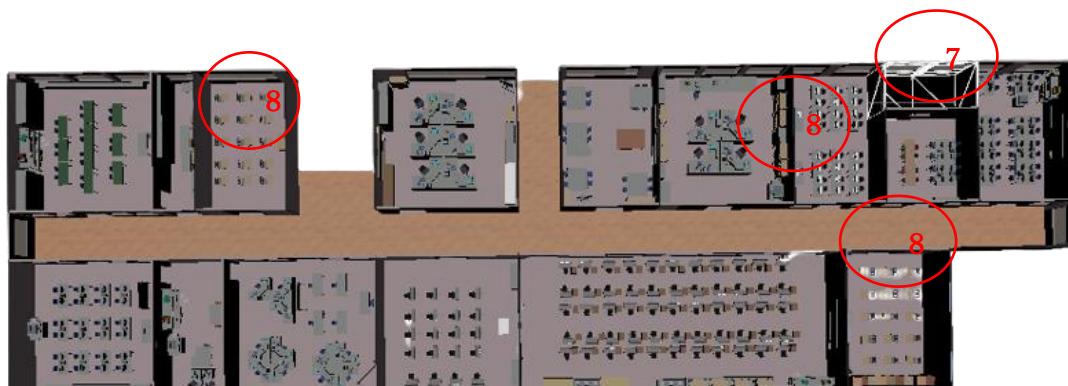


図16 実習棟エリア2階

以下に実習棟エリアのゲームの流れを説明する。

### ① 岩を壊す

実習棟エリアでは、マップ内に大きな岩があり、探索に制限がかけられている。  
探索できる範囲で攻略を探す。



図 17 マップ探索

### ② MT さんと戦う

キャラクターに話しかけ、戦いに勝つと  
旋盤が使えるようになる。



図 18 MT さんの会話



図 19 戦闘前



図 20 戦闘勝利後

### ③ ハンマーを作る

使えるようになった旋盤で、ハンマーを生成し、岩を壊す。  
岩の先のマップを探索する。



図 21 ハンマー作成

#### ④ 橋を作る

マップを探索していると、床が抜けた箇所がある。ここを渡る為に建築科の実習場へ向かう。



図 2 2 マップ探索

#### ⑤ 英ディと戦う

実習場に行くと、キャラクターがいる。  
近づくと戦いに進む。戦いに勝利することで、アイテムが入手できる。



図 2 3 建築実習場

アイテムを使用することで、橋を作ることができる。



図 2 4 建築キット使用



図 2 5 橋作成

#### ⑥ あいでいちゃんと戦う

橋の先のマップを探索していくと、キャラクターがいる。近づくと戦いに進む。  
戦いに勝利することでボスの居場所と、  
そこに掛けられている暗号を解くためのアイテムを入手できる。



図 2 6 あいでいちゃんと会話

## ⑦ 暗号を解く

あいでいちゃんに示された場所に向かうと、3つのイラストと色が変更するボタンが表示される。暗号を解く為に、マップ内からヒントを探す。



図27 暗号付きのドア

## ⑧ 暗号のヒントを見つける

実習棟エリアは3つの科から構成されている。各科の教室に各科のイメージカラーで染められたマスコットがあり、それが暗号の答え（メカトロニクス技術科：青、建築科：緑、産業デザイン科：ピンク）を示している。



図28 暗号の答え



図29 ヒント①



図30 ヒント②



図31 ヒント③

暗号を解くと扉が開き、ボスステージに進むことができる。



図32 ボスステージ前

### 3.4 キャラクター詳細

ゲーム内で使用できるキャラクターにはそれぞれ特性が有り,キャラクター性能(表2)を理解することで,バトルを有利に進めることができる.  
どのキャラクターにも,アタック技とガード技を備えており(表3),アタック技を使用した際に与えるダメージ量が攻撃力から出力されている.また,ガード技を使用した際には,相手から与えられた攻撃力から,自分の防御力を引いたダメージ量が出力される.

		HP	攻撃力	防御力
攻撃型	MTさん	230	55	20
体力型	いーていーくん	400	60	35
防御型	英ディ	220	30	30
回復系	あいでいちやん	280	40	30
普通型	アイティーくん	250	50	25
	ギサンタ太夫	500	50	0

表2 キャラクター値

スキル1	スキル2	スキル3
CADウェポン (55+次ターン攻撃力アップ35)	ディフェンドロボ (ガード)	メカトロキャノン (0,20,55,85)
ノイズオルゴール (60)	きばんのたて (ガード)	はんだづけ (0,20,55,85)
模型作り (防御力アップ10++)	ドラフターパンチ (30+持続攻撃力アップ5[重複可能])	ハンマーチョップ (0,20,55,85)
伝統工芸の盾 (ガード)	ドレーインパクト (攻撃力+20回復)	伝統工芸のつるぎ (0,20,55,85)
マインドハック (カウンター)	プロテクトコード (防御力20アップ)	シャットダウン (0 or 10000)

表3 技の早見表

# 第4章 システム構成

## 4.1 基本的なゲームシステム

本ゲームの基本的なゲームシステム(キャラクターが歩く, 扇が開く, バトルシーンのダメージ処理 等)は C#により制御している。本ゲームの動作の大本となる部分の制御について記す。(図3-3)

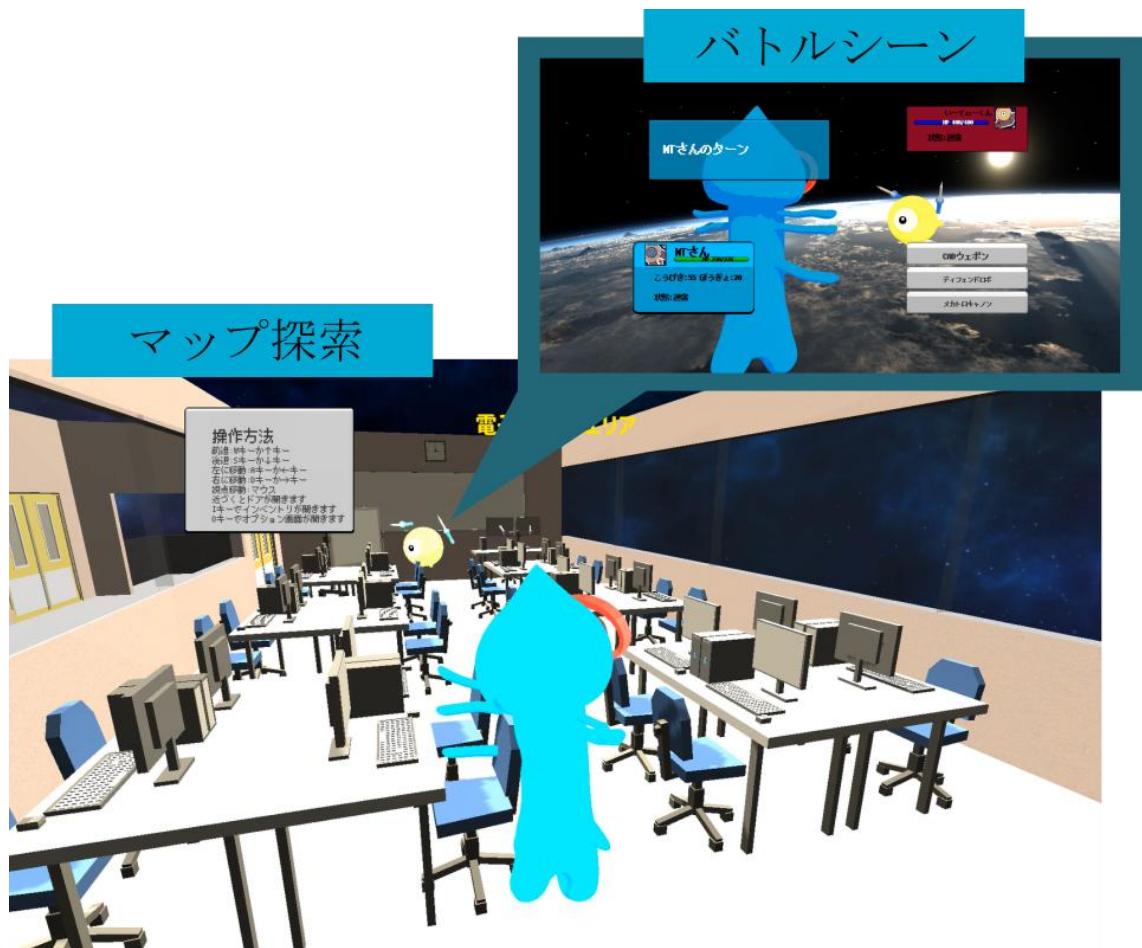


図3-3 ゲームシステムのイメージ

### 4.1.1 ターン制バトルの仕組み

本ゲームは大きく、マップ探索とバトルの2つに分かれている。ここではターン制バトルの仕組みについて記す。

バトルシーンでは、以下の処理がループで行われている。(図3-4)

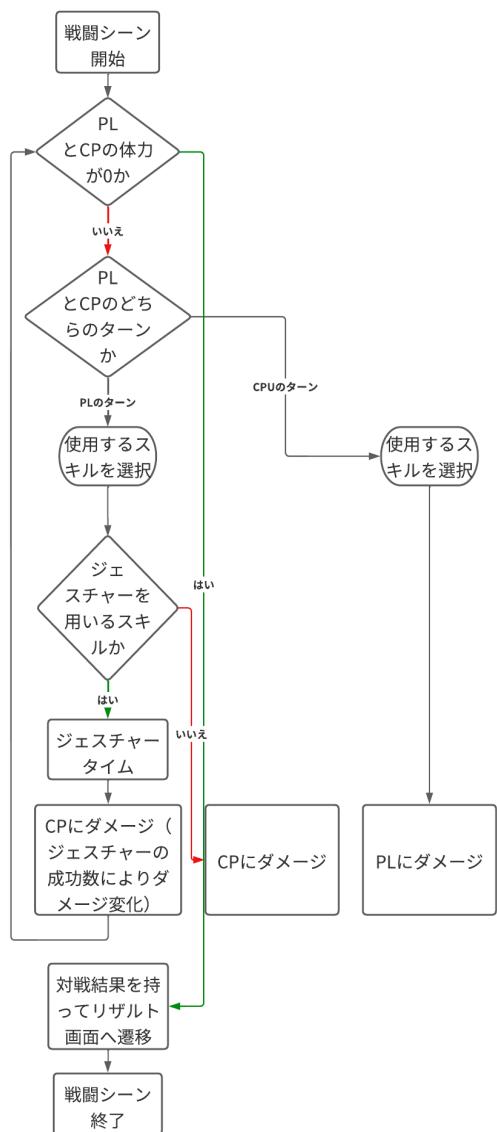


図3-4 バトルシーンのフローチャート

キャラクターのステータスは構造体で作成されている。

マップ探索のシーンからバトルのシーンに遷移する際にどのキャラクターを呼びだすかを PCode で判定し、一致する PCode を持っているステータスを呼び出す仕組みになっている。

```
15
16     public struct status
17     {
18         public int PCode;
19         //アイコン
20         public Sprite image;
21         public string name;
22         public int HP;           //ヒットポイント
23
24         public string Skill1;
25         public string Skill2;
26         public string Skill3;
27         public string Skill4;
28         public string Skill5;
29
30         public int Pw;          //パワー
31         public int Gd;
32
33         public int abnm;        //混乱
34         public int build;       //攻撃無効化
35         public int skip;
36         public int pdu;
37         public int gdu;
38         public int pdd;
39         public int gdd;
40     }
```

図3.5 キャラクターステータスの構造体

### 4.1.2 マップ探索の仕組み

ここではマップ探索の仕組みについて記す。

本ゲームのテーマは学校紹介であり,操作するイメージキャラクターも一から作成しているため,マップ探索のシーンはキャラクターがよく見える TPS 視点にした。

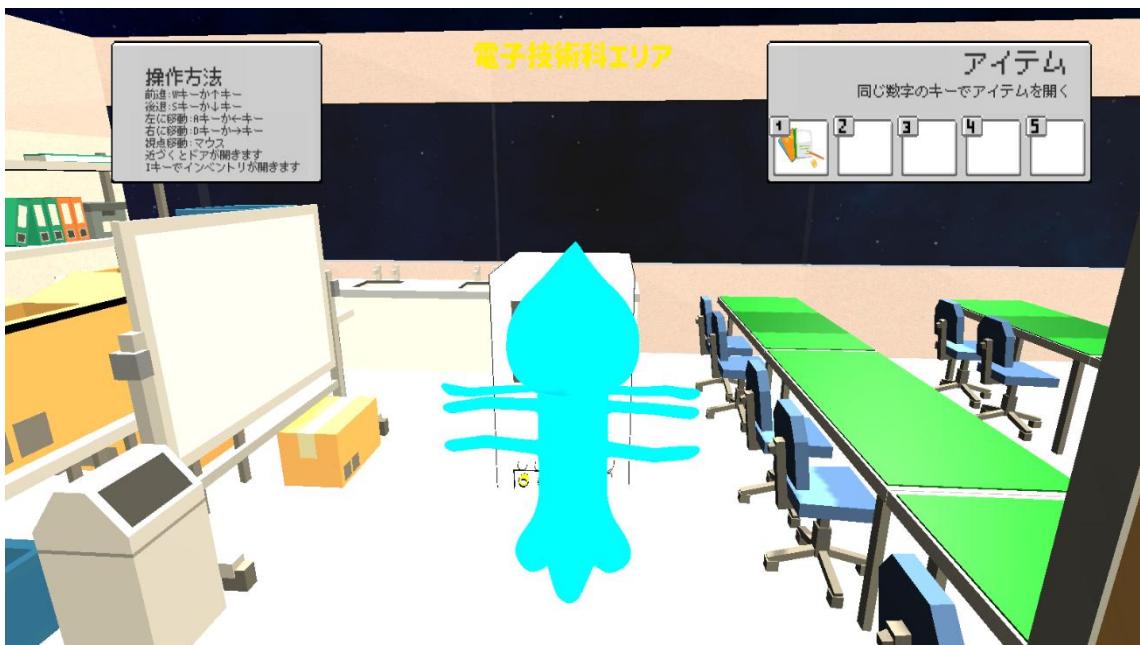


図 36 マップ探索の様子

TPS 視点の仕組みについては,下記の記事を参考にした。

sasanon. “Unity で 3D 空間内を TPS っぽく WASD 移動するキャラをつくる”. ささみ雑記帳. 2017-09-17. <https://sasanon.hatenablog.jp/entry/2017/09/17/041612>. (参照 2022-11-22).

マップに配置されたキャラクターとの会話は,「Fungus」というアセットを使用した。

Fungus は,Unity 用の会話シーン作成ができるアセット。

作成のほとんどを GUI で進めることができ,使い勝手が非常に良かった。

会話の作成だけでなく,アタッチされたスクリプトの関数呼び出し,セーブ,時間の停止

など、会話以外の演出を作成する際にも使用した。(図 6)



図 3 7 マップ探索の会話シーン

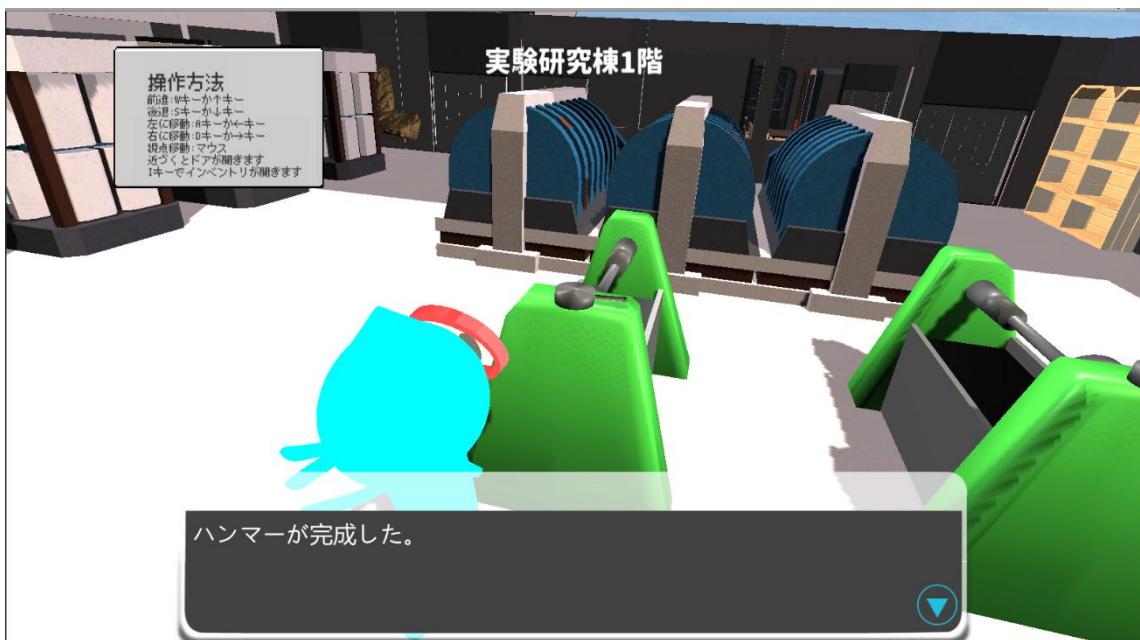


図 3 8 Fungus を使用したギミック

Fungus の使用方法やインストール方法は下記の記事を参考にした。

sensiki. “Unity のアセットストアから消えた「Fungus」をインストールする方法”. 好奇心俱楽部. 2021-12-11. <https://trend-tracer.com/fungas-install/>.(参照 2022-1-20).

Kirikabu\_ueda. “【Unity】 Fungus でゲーム制作(1)会話シーンを作る”. Qiita. 2020-10-30. [https://qiita.com/Kirikabu\\_ueda/items/bfc6e086d408b1cba34b](https://qiita.com/Kirikabu_ueda/items/bfc6e086d408b1cba34b).(参照 2022-1-20).

本ゲームでは各学科の学びと紐づけたギミックをマップに取り入れている。

ギミックの仕組みについて記す.(図39)

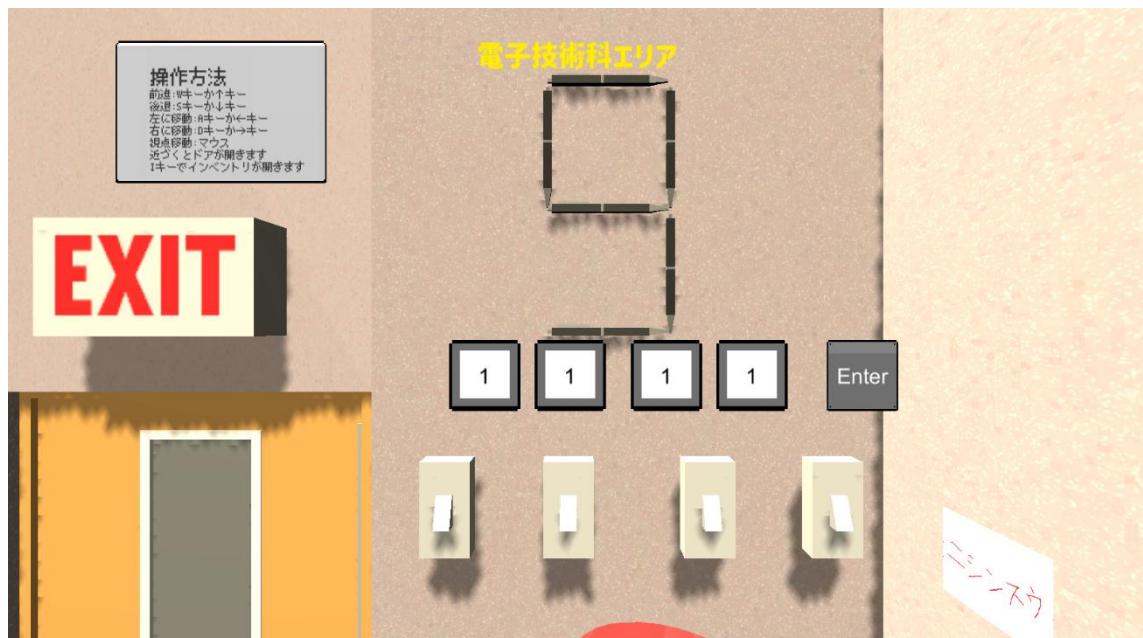


図39 マップ探索のギミック

上記のギミックは電子技術科フロアに実装した2進数を用いたパスワード式扉である。

パスワードを入力するスイッチの上に9の字が刻まれている。隣にはニシンスウと書かれた紙が貼られている。パスワードの答えは9の2進数である1001となっている。

パスワードは1と0を切り替えられるスイッチが4つ並んだ形になっており、スイッチをクリックすることで1と0の切り替えを行える。

このギミックの流れを以下の図 4 0 に示す.

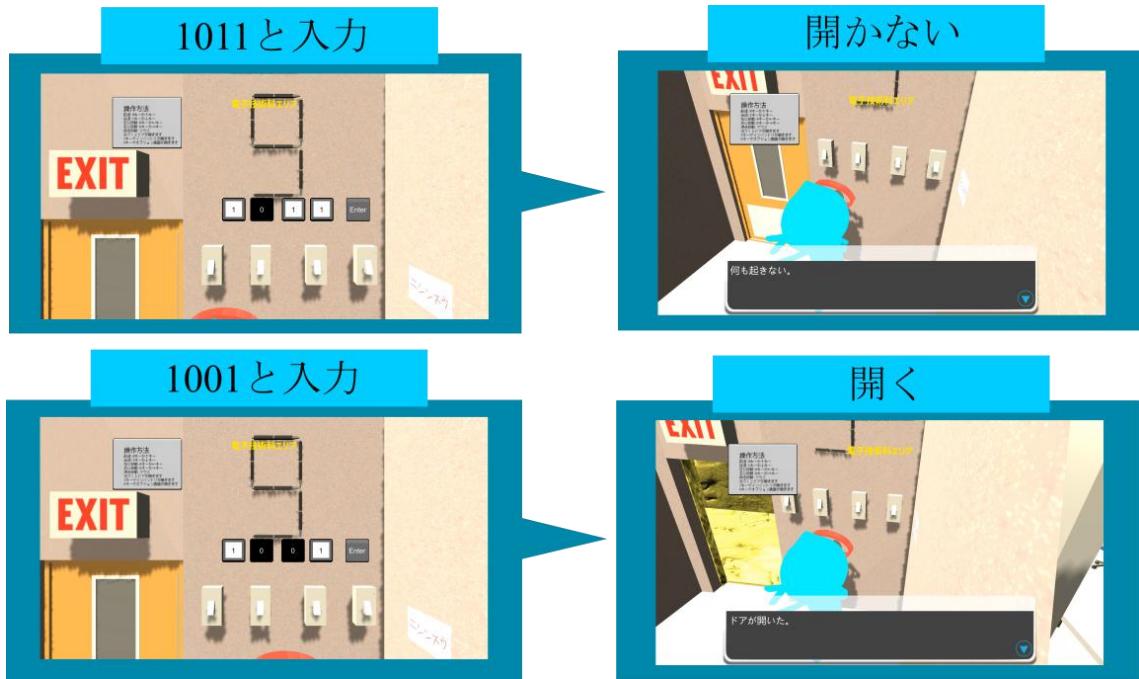


図 4 0 ギミックの動作

## 4.2 ジエスチャータイム

本ゲームにはジェスチャータイムと称した,手のポーズを用いたゲームシステムが実装されている.(図 4 1, 4 2)

ジェスチャータイムとは,連続的に表示されるポーズを真似するギミック.バトルシーンのスキル使用時に開始され,ポーズの成功数によってスキルの威力が変わるというもの.(図)  
このギミックのポーズを認識する処理に Mediapipe を使用している.



図4 1 ジェスチャータイム

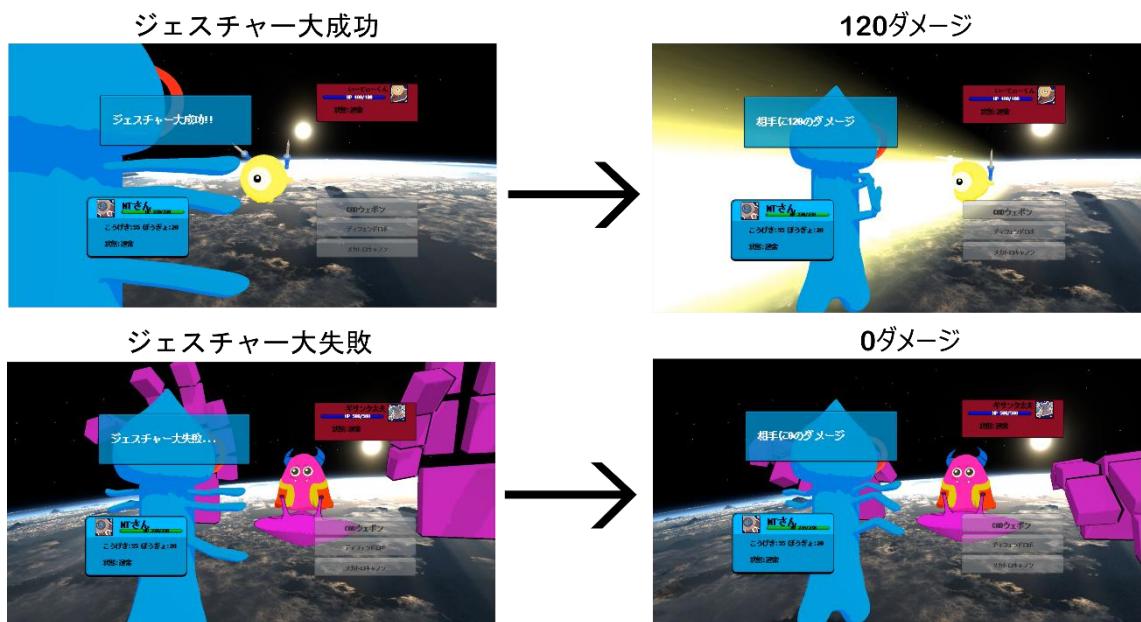


図4 2 ジェスチャータイム 2

ジェスチャータイムの制御を行うプログラムとそのシステム構成について記す。

#### 4.2.1 Mediapipe による手の座標検出

ハンドトラッキングを行う Mediapipe の Python プログラムについて記す。

Mediapipe による手の座標検出は,指先, 第一関節, 第二関節, 第三関節, 手首の座標を検出することが可能.(図 4 3 )

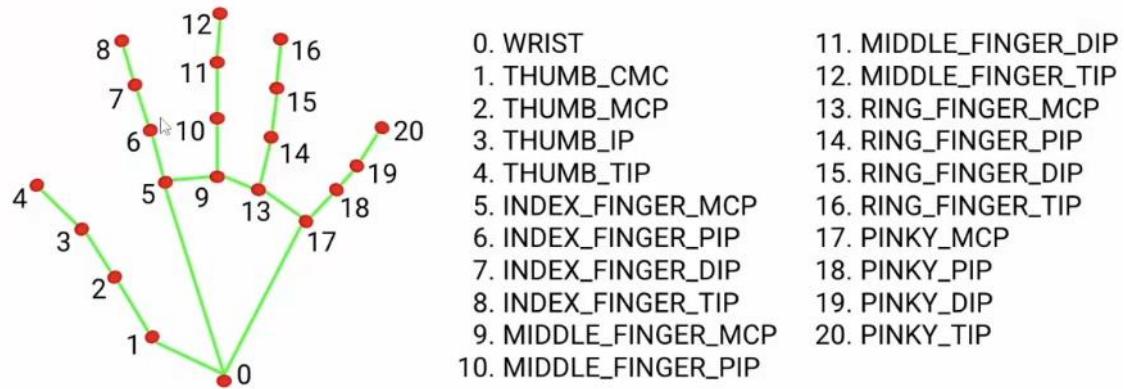


図 4 3 検出可能な手の座標

この検出した座標のひとつひとつを Unity 内のゲームオブジェクトに割り当てている。

#### 4.2.2 Unity と Python の連動

本ゲームでは Unity と Python の連動にソケット通信を使用している.具体的には,Unity に組み込まれた C#と Python をソケット通信により連動させ,データのやり取りを行っている。

ゲーム起動時に,Unity プロジェクトに組み込まれた C#スクリプトが手の座標検出を行う Python プログラムを起動させる.(図 4 4 )この Python プログラムがユーザーの手の骨格を読み取っている。

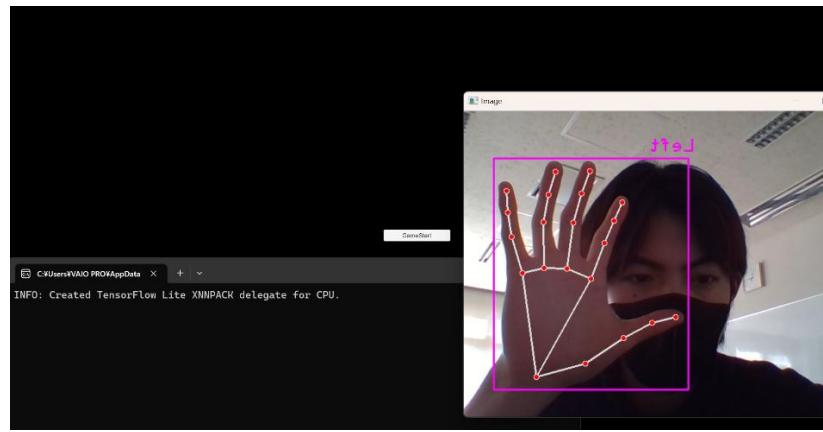


図 4 4 ゲーム起動直後の様子

この時,C#スクリプトと Python プログラムはソケット通信により,Python プログラムから C#スクリプトへ毎秒データを送るようになっている。(図 3)

ゲーム起動時から常に Python プログラムを動かし続ける理由は,Python プログラムの起動には時間がかかるためである。ジェスチャータイムの開始と同時に Python プログラムの起動を行おうとした場合,ゲームのスピード感を損なってしまうことになる。それを防ぐため,ゲーム起動時に Python プログラムの起動も行う設計にした。

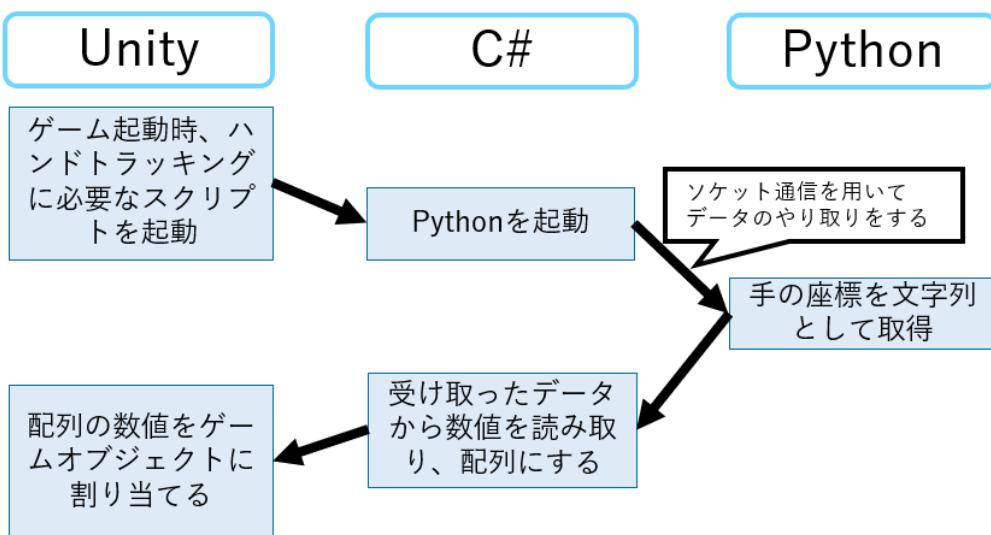


図 4 5 手の座標データの流れ

### 4.2.3 ハンドトラッキング(ゲーム起動時の処理)

起動時,ハンドトラッキングの処理を行う Python プログラム(main.py)について記す.

ハンドトラッキングを行うプログラムを起動すると,PC の内側カメラが起動する.(図 4 6 )

```
6  cap = cv2.VideoCapture(0)
7  cap.set(1, 1280)
8  cap.set(2, 720)
9  success, img = cap.read()
10 h, w, _ = img.shape
11 detector = HandDetector(detectionCon=0.8, maxHands=2)
12
```

図 4 6 カメラの起動

カメラを起動させた後,C#とデータのやり取りをする為のソケットを生成する.(図 4 7 )

```
12
13 # ソケットを生成
14 sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
15 serverAddressPort = ("127.0.0.1", 5052)
16
```

図 4 7 ソケットの生成

ソケットを生成したのち,以下の処理を繰り返す.(図 4 8 )

1. 画像フレームを取得する.
2. カメラに手が映っているか探し,手を認識できたら座標を取得する.
3. 手の座標を入れるための文字列配列を生成.

```
17 while True:
18     # Get image frame
19     success, img = cap.read()
20     # Find the hand and its landmarks
21     hands, img = detector.findHands(img) # with draw
22     # hands = detector.findHands(img, draw=False) # without draw
23     data = []
24
```

図 4 8 手の座標検出 1~3

認識できた手がひとつだった場合,以下の処理を行う.(図 4 9 )

4. 取得した座標を 3 で生成した文字列配列に代入する.
5. 文字列を C#側に送る.

```

25     # Hand 1
26     if (len(hands)==1):
27         hand = hands[0]
28
29         lmList = hand["lmList"] # List of 21 Landmark points
30             # List of 21 Landmark points
31             for lm in lmList:
32                 data.extend([lm[0], h - lm[1], lm[2]])
33
34             sock.sendto(str.encode(str(data)),serverAddressPort)
35

```

図 4 9 手の座標検出 4~5

認識できた手がふたつだった場合,以下の処理を行う.(図 5 0 )

4. 取得した座標を片手ずつのデータに分ける.
5. 4 のデータの片方を 3 で生成した文字列配列に代入する.
6. エラーを防ぐため,5 の文字列に区切りとして T を代入する.(のちに詳しく記載)
7. 4 のデータの片方を 6 に代入する.
8. 文字列を C#側に送る.

```

36     elif(len(hands)==2):
37         hand = hands[0]
38         hand2 = hands[1]
39
40         lmList = hand["lmList"] # List of 21 Landmark points
41         lmList2 = hand2["lmList"]
42             # List of 21 Landmark points
43             for lm in lmList:
44                 data.extend([lm[0], h - lm[1], lm[2]])
45
46             data.extend('T')
47
48             # List of 21 Landmark points
49             for lm in lmList2:
50                 data.extend([lm[0], h - lm[1], lm[2]])
51
52             sock.sendto(str.encode(str(data)),serverAddressPort)
53

```

図 5 0 手の座標検出 4~8

上記したプログラムの実行結果は以下の通りである。

手が認識されなかった場合は、[] が出力される。(図 5 1)

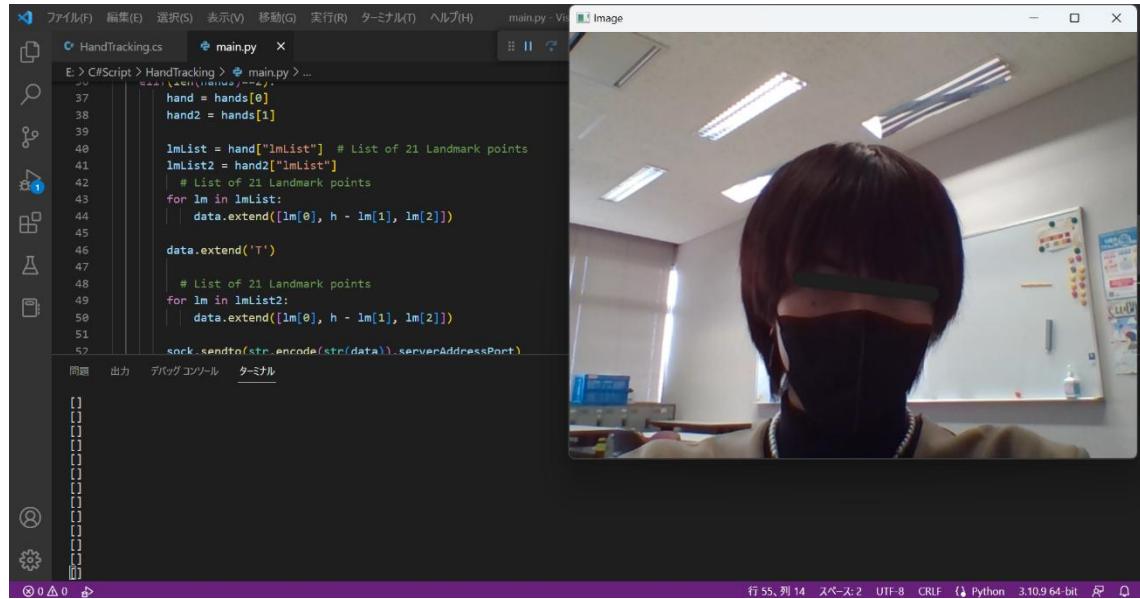


図 5 1 手をかざさなかった場合

片手が認識された場合は、[(手の x 座標, y 座標, z 座標) × 認識された骨格の数] が出力される。(図 5 2)

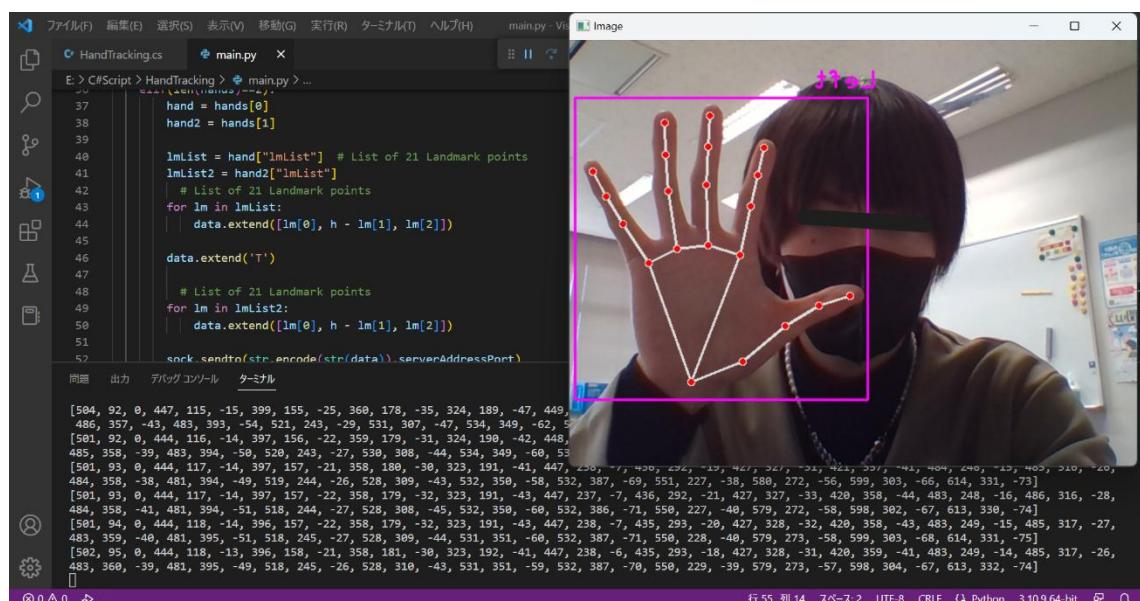


図 5 2 片手をかざした場合

両手が認識された場合は、[(手の x 座標, y 座標, z 座標) × 認識された片手の骨格の数 ,

T ,

(手の x 座標, y 座標, z 座標) × 認識された片手の骨格の数]が出力される。(図 5.3)

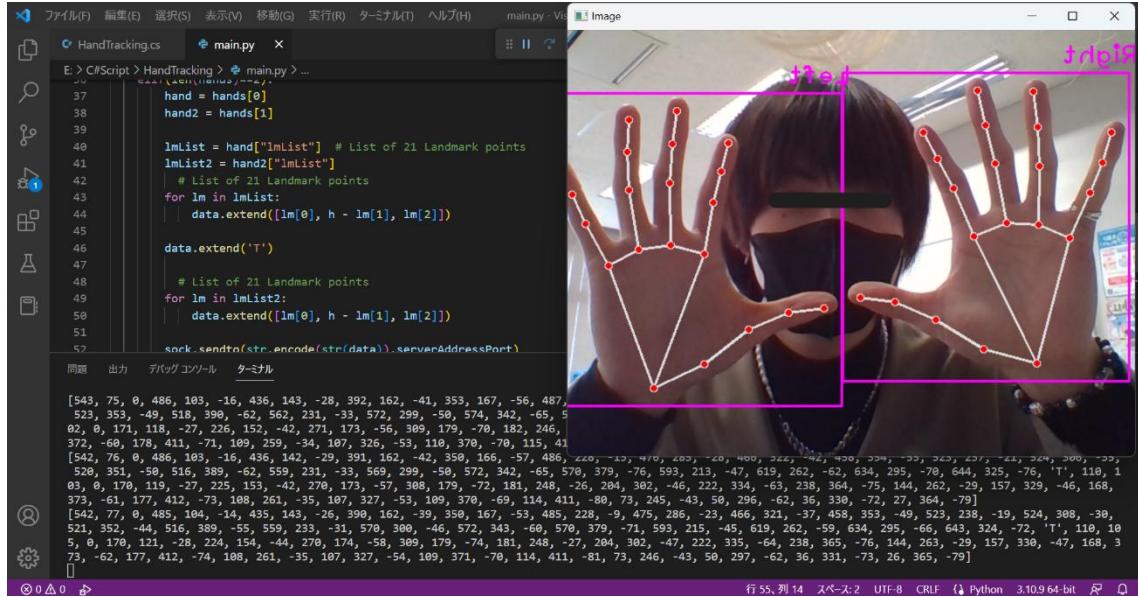


図 5.3 両手をかざした場合

Python プログラムの起動および手の座標データの受け取りを行う C# スクリプト (UDPReceive.cs)について記す。

Python の exe ファイルを起動するための文字列を生成。(図 5.4.11 行目)

Python プログラムの絶対パスを指定するための文字列を生成。(図 5.4.13 行目)

```
9  public class UDPReceive : MonoBehaviour
10 {
11     const string PythonExe = "python.exe";
12
13     const string PythonApp = "F:/Python_Shippuden/main.py";
14 }
```

図 5.4 C#スクリプトの解説(変数生成)

Python プログラムからデータを受け取るためのスレッドを起動。(図 5.5.46 行目)

Python プログラムを起動するための関数を呼ぶ。(図 5.5.48 行目)

```

41     public void Start()
42     {
43         receiveThread = new Thread(
44             new ThreadStart(ReceiveData));
45         receiveThread.IsBackground = true;
46         receiveThread.Start();
47
48         Main();
49     }

```

図 5 5 C#スクリプトの解説(Start 関数)

任意のアドレス,任意のポートからの送信を許可する.(図 5 6.59 行目)

受信するまで待機する.(図 5 6.61 行目)

受信したデータを UTF-8 文字列に変換して表示する.(図 5 6.63~65 行目)

```

51     // receive thread
52     private void ReceiveData()
53     {
54         client = new UdpClient(port);
55         while (startRecieving)
56         {
57             try
58             {
59                 IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
60
61                 byte[] dataByte = client.Receive(ref anyIP);
62
63                 data = Encoding.UTF8.GetString(dataByte);
64
65                 if (printToConsole) { print(data); }
66             }
67             catch (Exception err)
68             {
69                 print(err.ToString());
70             }
71         }
72     }
73 }
74 }
```

図 5 6 C#スクリプトの解説(受信用スレッド)

以下の設定を行い,Python プロセスを起動.(図 5 7 )

- ・プロセスの起動に OS のシェルを使用しない.
- ・テキスト出力を StandardOutput ストリームに書き込む.
- ・PythonExe を呼び出す際にコマンドライン引数(Python プログラムの絶対パス)をセットする.

```
--  
22     public static void Main()  
23     {  
24         using (var process = new Process  
25         {  
26             StartInfo = new ProcessStartInfo(PythonExe)  
27             {  
28                 UseShellExecute = false,  
29  
30                 RedirectStandardOutput = true,  
31  
32                 Arguments = PythonApp  
33             }  
34         })  
35         {  
36             process.Start();  
37         }  
38     }  
39 }
```

図 5 7 C#スクリプトの解説(Python 起動)

#### 4.2.4 ハンドトラッキング(ジェスチャータイム時の処理)

ジェスチャータイム時,ハンドトラッキングの処理を行うプログラム(HandTracking.cs)について記す.ジェスチャータイムの仕組みは以下の通りである.(図 5 8 )

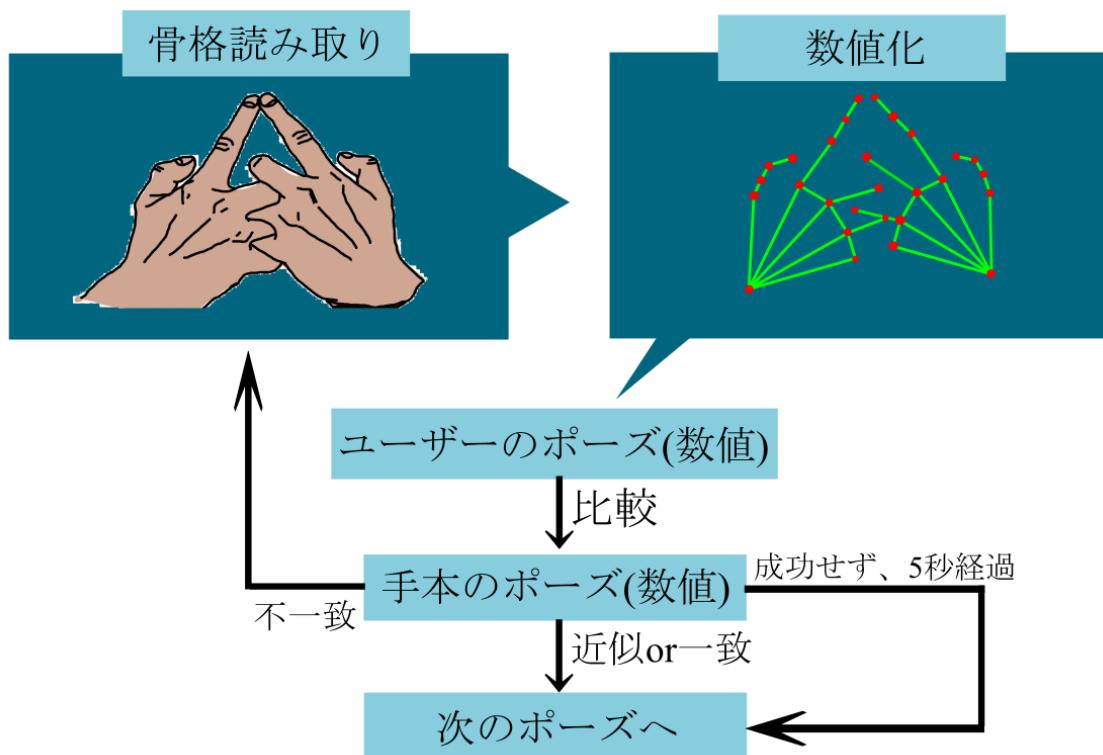


図 5.8 ジェスチャータイムの仕組み

ジェスチャータイム時のような動き(図 5.9)を実装するために行った工程は以下の通りである。

1. 座標を割り当てるゲームオブジェクトを定義する。
2. 文字列を数値化(float 型の変数)する。
3. 数値化したデータをゲームオブジェクトに割り当てる。
4. 判定式を用いてポーズがうまくできているかを判定する。



図 5.9 ジェスチャータイムの様子

1. 座標を割り当てるゲームオブジェクトを定義する。

このスクリプトは、ゲームオブジェクトに座標を割り当てるものであるため、ゲームオブジェクトの配列を public 変数で定義する。

配列にする理由は、手の座標が複数あるためである。また、public 変数にする理由は、public 変数は中身を Unity 側から変更できるため。

```
7  public class HandTracking : MonoBehaviour
8  {
9      // Start is called before the first frame update
10     public GameObject[] handPoints;
11     public GameObject[] handPoints2;
12     public bool handCheck = false;
13 }
```

図 6 0 ゲームオブジェクトの定義

2. 文字列を数値化(float 型の変数に)する。

前述したデータ受信用の C# スクリプトから、座標の入った文字列配列を取得する。

(図 6 0)

文字列配列から [を取り除く。

新しい文字列配列 hands を用意し、0 番目の要素に

[文字列配列の 0 番目から、 T が入った要素まで]の文字列を代入する。

hands の 1 番目の要素に、

[T が入った要素の次から、 文字列配列の最後まで]の文字列を代入する。(図 61,27 行目)

hands の 0 番目の要素(片手分の座標が入った文字列)をコンマごとに区切り、新しい文字列配列 hand1 に入れる。(図 61,29 行目)

```

20     void Update()
21     {
22         string data = UDPReceive.data;
23
24         data = data.Replace("[", "");
25         data = data.Replace("]", "");
26
27         string[] hands = data.Split('T');
28         //string[] points = data.Split(',');
29         string[] hand1 = hands[0].Split(',');
30
31         if (hands[0].Length != 1)
32             handCheck = true;

```

図 6 1 座標データを数値化

```

[15:21:0] 138,-6,0,199,4,-30,258,29,-46,303,55,-81,343,80,-75,223,141,-15,259,192,-34,281,223,-52,300,253,-65,189,158,-15,212,223,-30,233,258,-44,251,282,-55,152,158,-20,154,221,-38,165,237,-46,177,242,-49,113,146,-28,109,196,-44,114,212,-46,120,216,-44
[15:21:0] MonoBehaviour:print(Object)

```

図 6 2 hands[0] , hands[1]を出力した結果

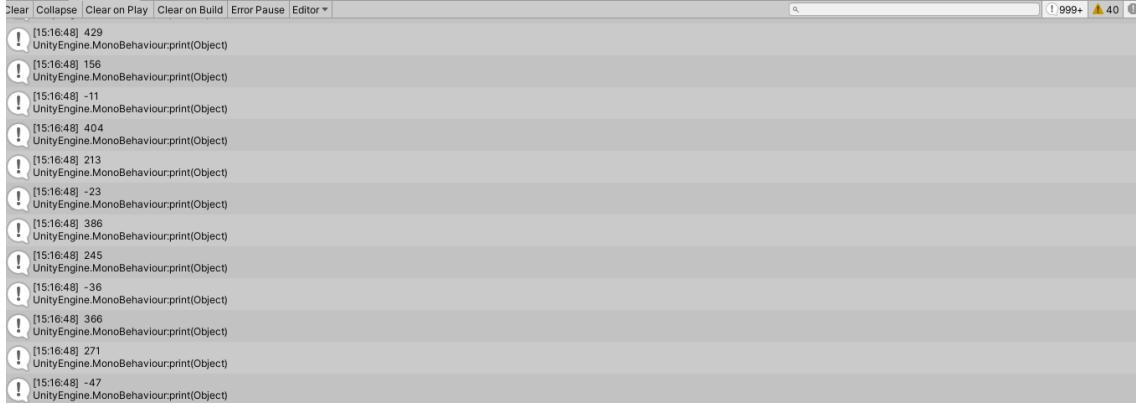


図 6 3 hand1 の要素を表示した結果

### 3.数値化したデータをゲームオブジェクトに割り当てる。

上記した文字列配列を float 型に変換し,ゲームオブジェクトに割り当てる.(図 6 4)

```

36
37     for (int i = 0; i < hand1.Length / 3; i++)
38     {
39         float x = 7 - float.Parse(hand1[i * 3]) / 100;
40
41         float y = float.Parse(hand1[i * 3 + 1]) / 100;
42
43         float z = float.Parse(hand1[i * 3 + 2]) / 100;
44
45         handPoints[i].transform.localPosition = new Vector3(x, y, z);
46     }
47

```

図 6 4 取り出した座標データの割り当て(片手)

上記した処理のみでは、片手分のみの割り当てとなる。

ただもう一つ同じ処理をしようとするとき片手のみ認識していた場合にエラーが起きてしまうので、Try-catch を用いてエラーでゲームが停止することを回避する。(図 6 5 )

```

48     try
49     {
50         hands[1].Replace("T", "");
51         string[] hand2 = hands[1].Split(',');
52
53         int remove = 0;
54         hand2 = hand2.Where((source, index) => index != remove).ToArray();
55
56         for (int i = 0; i < hand2.Length / 3; i++)
57         {
58             float x = 7 - float.Parse(hand2[i * 3]) / 100;
59
60             float y = float.Parse(hand2[i * 3 + 1]) / 100;
61
62             float z = float.Parse(hand2[i * 3 + 2]) / 100;
63
64             handPoints2[i].transform.localPosition = new Vector3(x, y, z);
65         }
66     }
67     catch(Exception)
68     {
69     }
70 }

```

図 6 5 取り出した座標データの割り当て(両手)

次に、座標データを割り当てるためのゲームオブジェクトを作成する。

座標データを割り当てるためのゲームオブジェクトの完成イメージは以下の通りである。

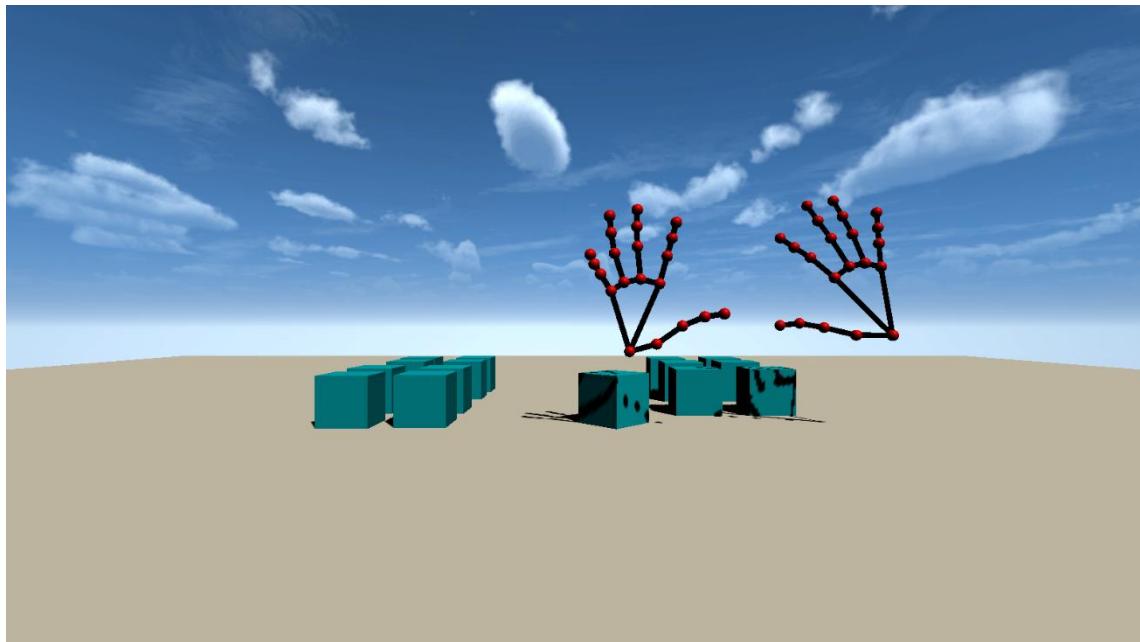


図 6 6 座標を割り当てたゲームオブジェクトの様子

このオブジェクトは、座標を示す「赤い球体」と骨を示す「緑の棒」で構成されている。

まずは赤い球体(Sphere)を作成する。

[Hierarchy]→[Create]→[3D Object]→[Sphere]から球体を作成できる。

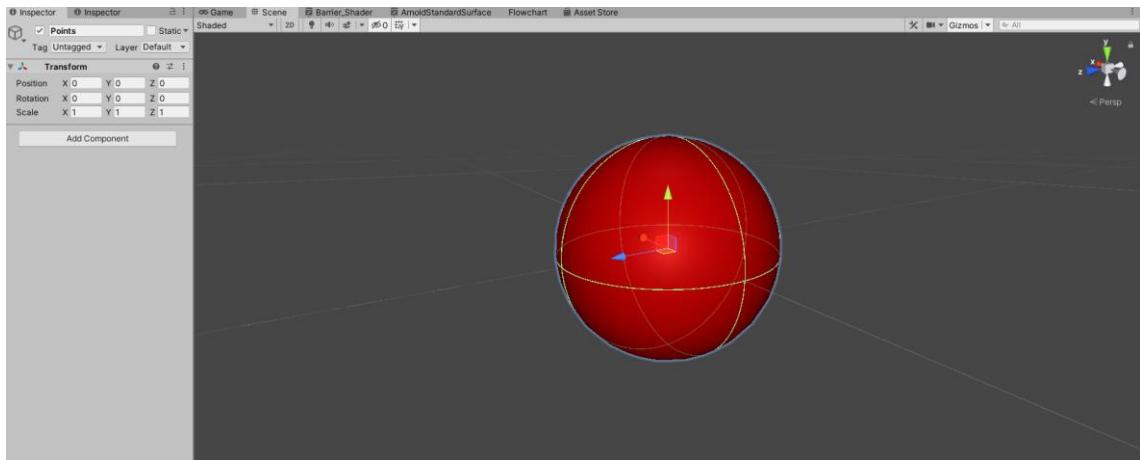


図 6 7 Unity で作成した赤い球体

片手の座標は 21 つあるため、上記のオブジェクトを両手分(42 つ)用意する。(図 6 8)空のオブジェクト(Points)を作成し、球体を 21 つずつに分けておくと作業がスムーズになる。

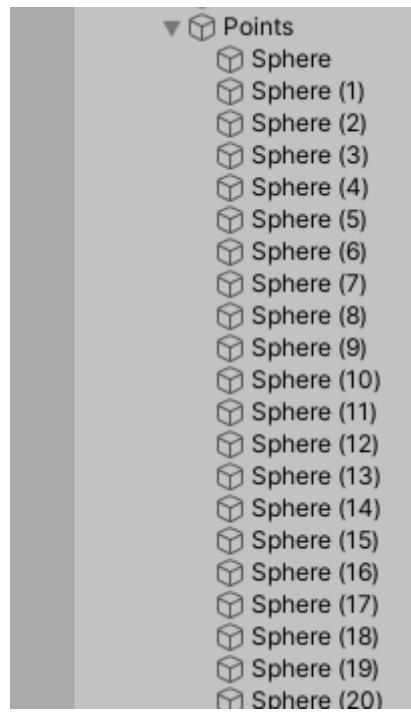


図 6.8 用意した球体

赤い球体の座標に Python から受信した座標を割り当てるため,赤い球体の座標を参照で  
きるようにする.空のオブジェクトを作成し,HandTracking.cs をアタッチする.(図 6.9 )

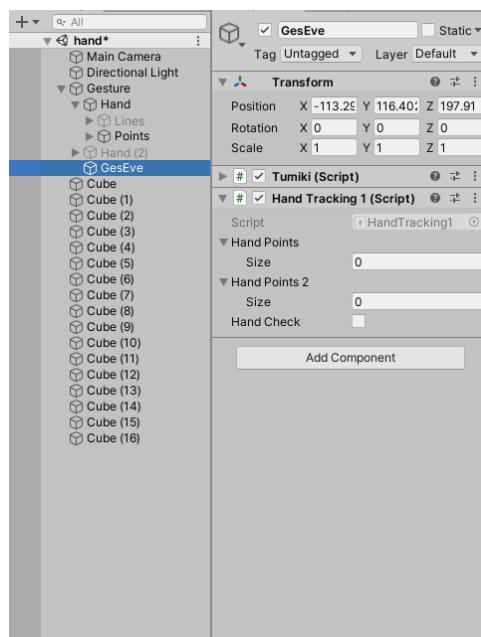


図 6.9 HandTracking.cs をアタッチした様子

アタッチしたスクリプトの Hand Points, Hand Points2 に図 7 0 のゲームオブジェクトをまとめて代入する。(Hand Points, Hand Points2 は public 変数として定義しているので Unity 側から代入することができる)

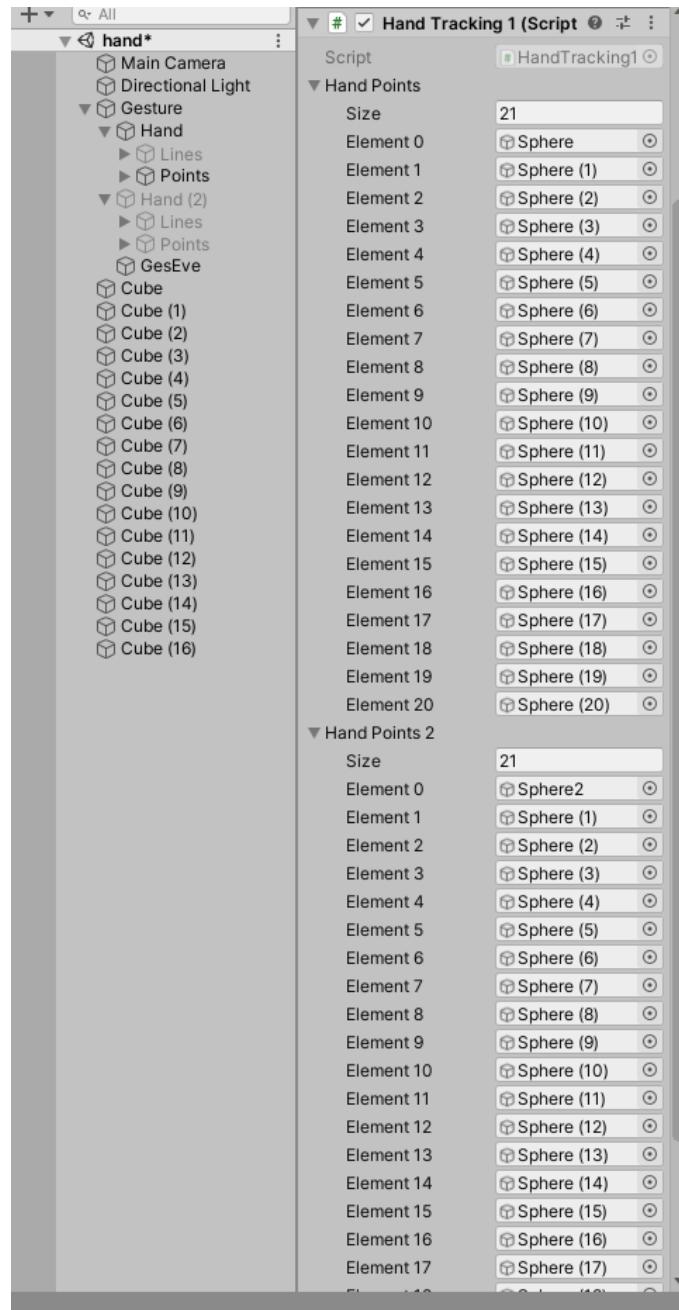


図 7 0 Hand Points, Hand Points2 に球体を代入した様子

次に骨を表す緑の棒を作成する。

[Hierarchy]→[Create]→[Effects]→[Line]から棒を作成できる。

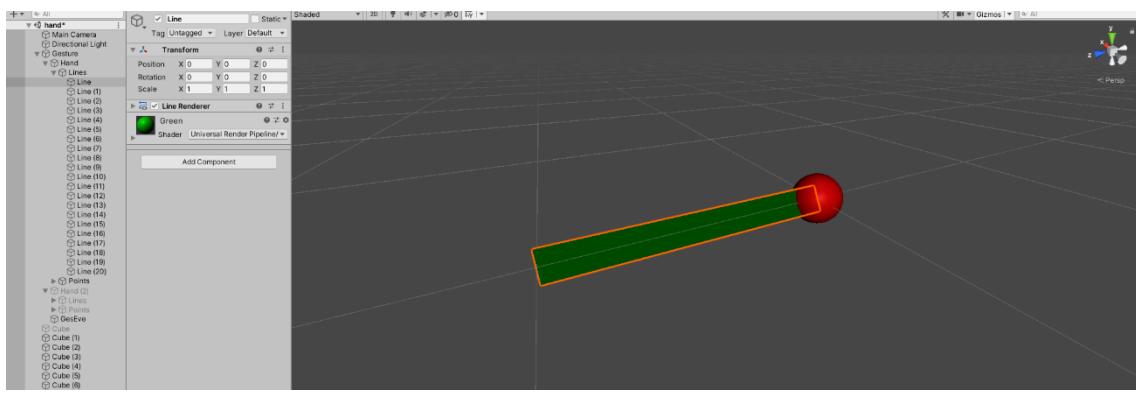


図 7 1 Unity で作成した緑の棒

棒は骨を表しているため,赤い球体と赤い球体を繋ぐ位置になければならない.そのための棒の座標の制御スクリプト(LineColor.cs)を作成する。

棒の始点と終点になる球体を指定するため,ゲームオブジェクトの座標(position)を

public 変数で 2 つ定義する。

```
5  public class LineCode : MonoBehaviour
6  {
7
8      LineRenderer lineRenderer;
9
10     public Transform origin;
11     public Transform destination;
12 }
```

図 7 2 LineCode.cs(変数定義)

棒の太さを指定するため,棒のコンポーネント(オブジェクトの処理を行うもの)を取得する。

取得したコンポーネントに,棒の太さを指定する.(図 7 3 )

```

13     // Start is called before the first frame update
14     void Start()
15     {
16         lineRenderer = GetComponent<LineRenderer>();
17         lineRenderer.startWidth = 0.1f;
18         lineRenderer.endWidth = 0.1f;
19     }

```

図 7 3 LineCode.cs(棒の太さを指定)

棒の始点と終点にそれぞれ球体の座標をセットする。

Update 関数は、ゲーム実行時に毎秒呼び出されるため、棒の始点と終点は常に球体の座標を指定され続ける。

```

20
21     // Update is called once per frame
22     void Update()
23     {
24         lineRenderer.SetPosition(0, origin.position);
25         lineRenderer.SetPosition(1, destination.position);
26     }
27 }

```

図 7 4 LineCode.cs(棒の始点終点の座標指定)

上記のスクリプトを棒のオブジェクトにアタッチし、球体を代入したものが下記の画像である。

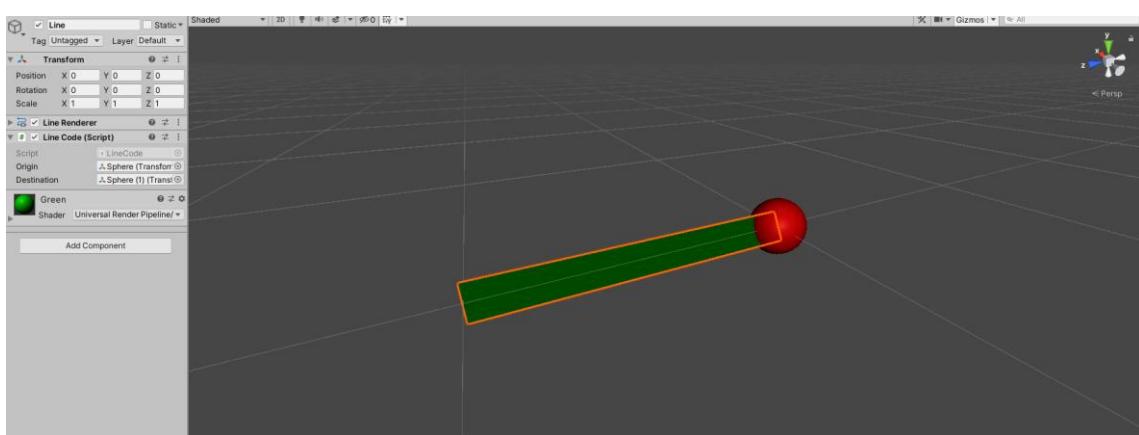


図 7 5 完成した棒の様子

4.判定式を用いてポーズがうまくできているかを判定する.

3 の工程で完成した手のオブジェクトをジェスチャータイムに使用するためには,オブジェクトの座標を参照できるようにする必要がある.球体を変数に代入するため,ゲームオブジェクトの配列を public 変数で定義する.

```
11     public Text timer;           //タイマー
12     public GameObject[] hand;    //手のオブジェクト
13     public GameObject[] handt;   //手のオブジェクト
14     public Text success;        //印を結べ！のどこ
15     public ParticleSystem graet;
16
17     public Image[] hapc;        //見本の絵
```

図 7 6 ジェスチャータイムスクリプト

Distance 関数を使い,球体同士の距離を参照する.

指定するポーズをとった際に,指同士がどのくらいの距離になるかを調べる(Distance 関数で取得した距離を print で表示するなど).  
調べた距離とゲームプレイ中の指同士の距離を比べる判定式を使い,ポーズが正しくとれているか調べることができるようとする.

```
125     float dis1 = Vector3.Distance(fin0, fint0);
126     float dis2 = Vector3.Distance(fin2, fint2);
127     float dis3 = Vector3.Distance(fin6, fint6);
128     float dis4 = Vector3.Distance(fin10, fint10);
129     float dis5 = Vector3.Distance(fin14, fint14);
130     float dis6 = Vector3.Distance(fin18, fint18);
131
132     if (dis1 <= 1 && dis2 <= 1 && dis3 <= 1 && dis4 <= 1 && dis5 <= 1 && dis6 <= 1)
133     {
134         scps = scps + 0.5;
135         pose = true;
136     }
```

図 7 7 ポーズが正しくとれているかの判定式

# 第5章 マップ作成

## 5.1 マップ概要

マップは産技短(矢巾キャンパス)の各科フロア（図78,79,80）をモチーフにして作成した。ボスステージ（図81）では、学校の崩壊をテーマに雰囲気のある空間を目的とした。マップ作りでは、オブジェクトを作成し、全体の外枠を組むことや、Assetを使用し、備品などの細かいところの再現。また、マテリアルを変更し、キャラクターと合うように設定を行う。



図78 情報技術科フロア

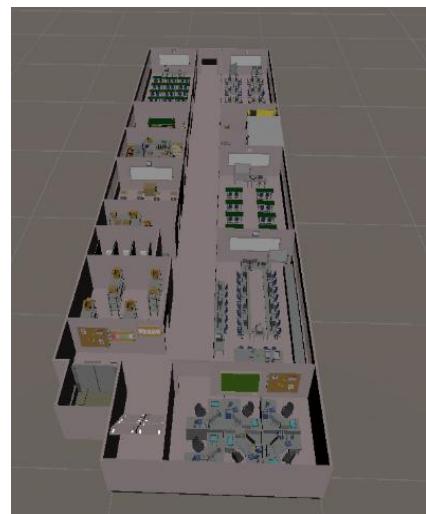


図79 電子技術科フロア



図80 実習棟フロア

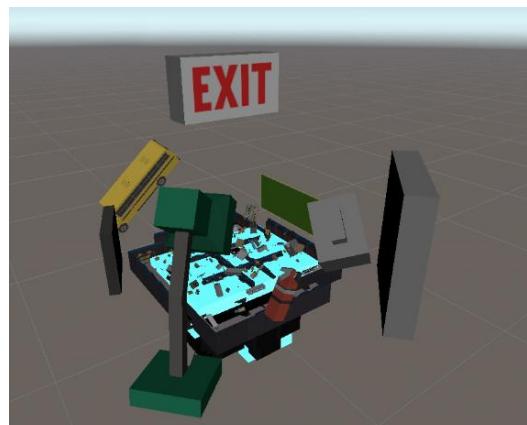


図81 ボスステージ

## 5.2 使用 Asset

- ① school
- ② SimpleOfficeInteriors
- ③ Sci-Fi Styled Modular Pack
- ④ Original Bricks Textures
- ⑤ Wooden floor

以下使用部分

- ① 黒板・ロッカー・本棚・ドアなど学校の設備備品に使用.また,情報技術科フロアの外枠や床に使用している.  
オブジェクトを以下の図8 2,8 3,8 4に示す.

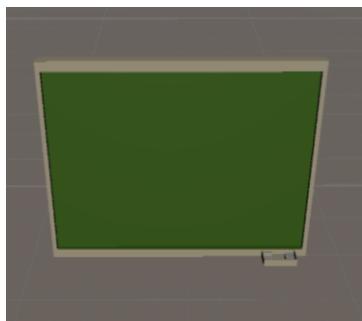


図8 2 黒板



図8 3 ロッカー

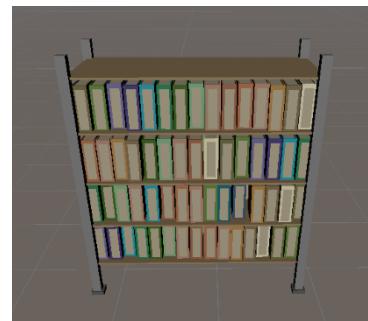


図8 4 本棚

- ② パソコン・机・椅子・ラック・プリンタなど学校の備品に使用.  
オブジェクトを以下の図8 5,8 6,8 7に示す.



図8 5 PC,机,椅子



図8 6 ラック

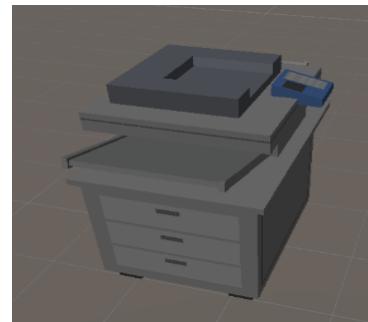


図8 7 プリンタ

- ③ 階段・モニュメント・ライト・ガラスのマテリアルなどに使用.また,電子技術科  
フロア・実習棟エリアの外枠や床に使用している.  
オブジェクトを以下の図8 8,8 9,9 0に示す.

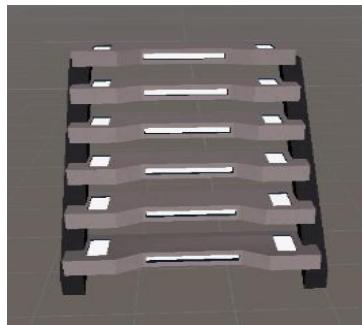


図8 8 階段



図8 9 モニュメント

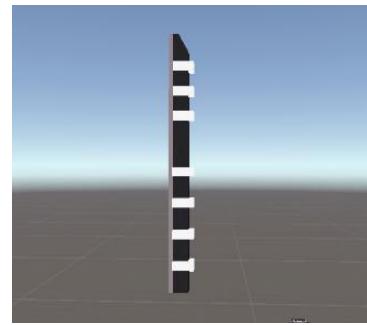


図9 0 ライト

- ④ 木製の机のマテリアル・建築科の講義室のマテリアルに使用.  
以下の図9 1に示す.



図9 1 講義室

- ⑤ 電子技術科フロア・実習棟エリアの床のマテリアル・建築科の実習場に使用.  
以下の図9 2に示す.



図9 2 実習場

# 第6章 キャラクターの作成

## 6.1 使用技術

### 6.1.1 Blender

3D モデリング, モーショングラフィックス, アニメーション, シミュレーション, レンダリング, デジタル合成 (コンポジット) などの機能を備えている。Unity にモデル, アニメーションのインポートをできることから、今回は Blender を使用した。

### 6.1.2 Unity

Blender からインポートしたキャラクターにマテリアルとアニメーションを接続するために使用する。キャラクターによってはアーマチュアが複数あるものが存在するため、数回に分けて接続する必要がある。

## 6.2 制作過程

基本的な流れとしては初めに Blender でメッシュ(キャラクターの外観)を使用してモデリングを行っていく。既存のメッシュを組み合わせ、形を変更させることで作成する。メッシュが完成したらメッシュ中にボーンを入れる。ボーンとメッシュをペアリングしたら、アニメーションを作成する。それぞれの作成について詳しくは以降に記載する。

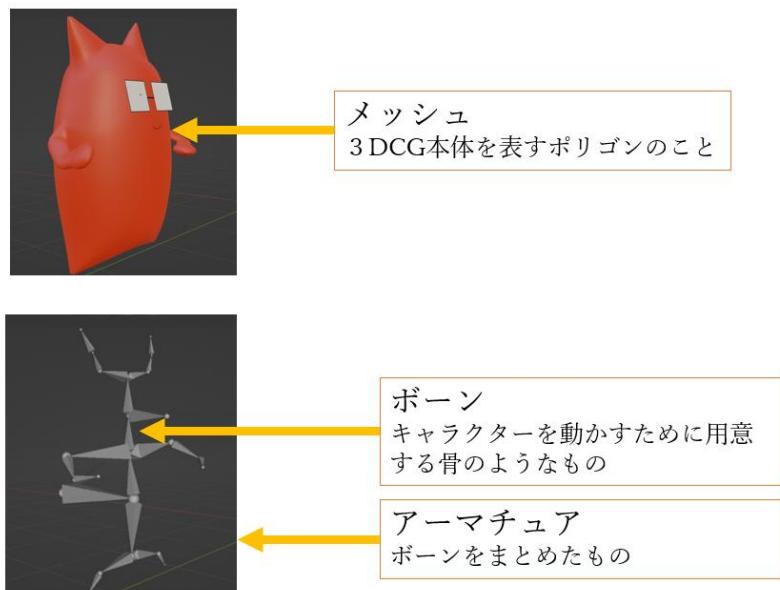


図9.3 Blendrの説明

## 6.3 キャラクターの作成方法

### 6.3.1 メッシュの作成方法

作成するキャラクターの形に近いメッシュをレイアウトモードで追加していく。近い形のないものは平面を追加して、少しづつ引き伸ばしながら作成する。平面を使う方法はかなり時間がかかるかつ、失敗もしやすいため、前者の作成方法が好ましい。

- ① 作成するキャラクターに近いメッシュを追加し、大まかな形を決める

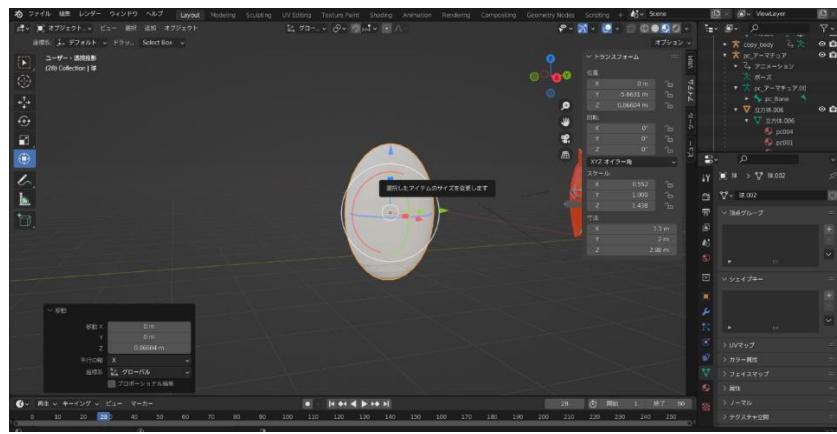


図9.4 メッシュの追加

- ② 形を整え不要な部分を削除する。プロポーショナル編集を使用することで、全体の形を均等に変えることができる。Fキーで空いている部分を埋めることができる。

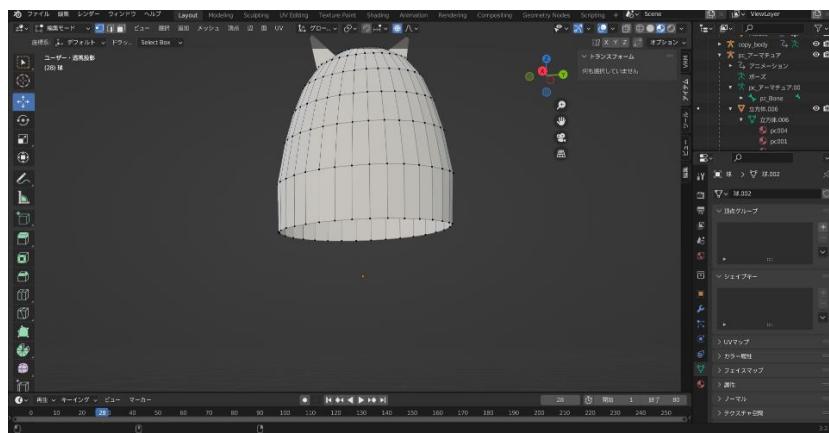


図9.5 メッシュの編集

- ③ プロポーショナル編集を活用し、形を作っていく。スムーズシェードとサブディビジョンサーフェイスを使用し、体の角を平らにする。Eキーで面を引き延ばすことができるの手や足、耳といった細かいパーツを付け足していく。新しくメッシュを追加し、パーツを作ってもいいが、アーマチュアが反応しなくなる場合があるため、Eキーで引き延ばした方がよい。

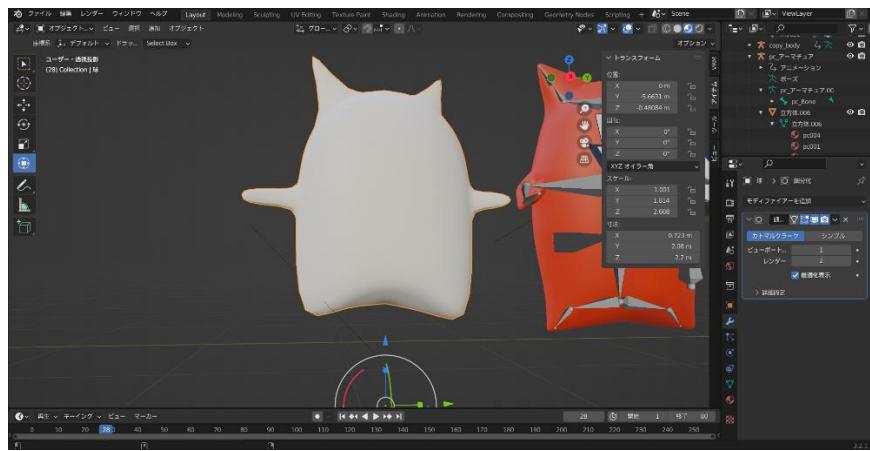


図9.6 メッシュの引き延ばし

④ 頂点を変形させたりして細かく形を整えていく。Unity 側にマテリアル(色)を移すことはできないが、見やすくするためにマテリアルを変更する。また、パーツごとにマテリアルを変更する場合は Blender の時点で設定しておく必要がある。Unity 側ではマテリアルの移動ができないがマテリアルを入れる部分は変更することができないため、Blender 側で設定しておく必要がある(マテリアルの枠組みを作るだけであって色まで変える必要はない)。

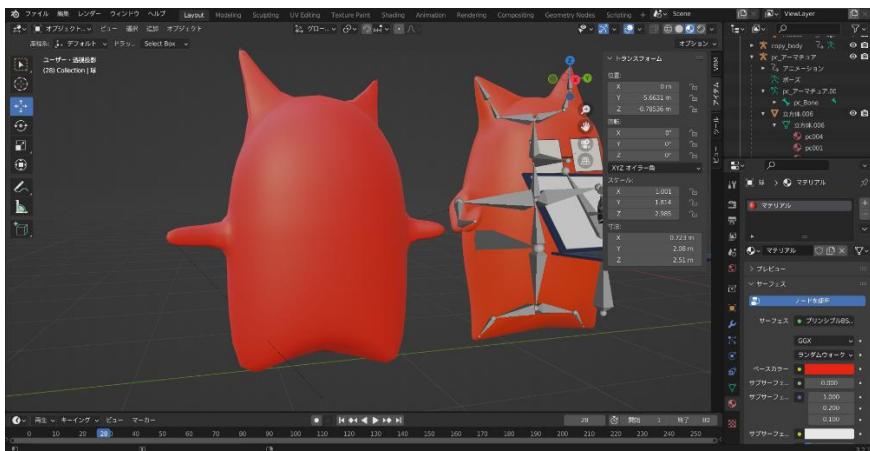


図9.7 頂点の編集、マテリアルの追加

⑤ 目や口といった小さなパーツ、キャラクターの持つアイテムもメッシュを変形させ作っていく。作り方は前記の通り。細かいパーツはアーマチュアとペアレンツする前に作っ

ておく必要がある。ペアレント後に作ってしまうとパーツが置き去りにされ、キャラクターが動いてしまう。



図9 8 パーツの作成

### 6.3.2 アーマチュアの接続

- ① メッシュを作成したらアーマチュアを連携させていく。追加からアーマチュア、単一ボーンを選択する。この時にアーマチュアのビューポート表示を最前面にしておくと、ボーンがメッシュを透けて見えるようになり作業しやすい。また、一つ目のボーンは体の外に置く。そうするとキャラクターをまとめて動かせやすくなりアニメーションが作りやすくなる。キャラクターによってはボーンを分けて作るキャラクターがいるので、この段階でボーンを追加する。

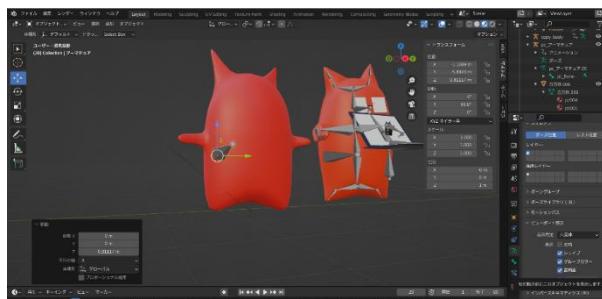


図9 9 アーマチュアの追加

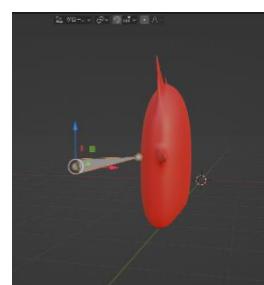


図1 0 0 一つ目のボーン



図101 アーマチュアの分解

- ② キャラクターの動かしたい部分にボーンを E キーで伸ばして追加していく。関節を作つて細かく動かしたい場合はその分ボーンを分けて追加していく。目や体についてあるアイテムにもボーンを伸ばしていく。同一のアーマチュアのボーンにすることで、小さなパーツもキャラクターに追随して動かすことができる。①でボーンを分けて作成した場合は名前を被らないように作成する必要がある。同じ名前のボーンがあるとアニメーションで動かしたときに別のボーンも動いてしまうので名前を変える必要がある。
- ③ ボーンの数が多い場合は対象化で自動的に作成することができる。初めに体の半分側だけにボーンを追加していく。追加したボーンに Arm\_L.001 と名前を付ける。名前と番号の間に L(R) と追加することで対象化を使用したときに、反対側に Arm\_R.001 といった逆にボーンを自動的に作成することができる。
- ④ 追加したアーマチュアとメッシュを連携する。オブジェクトモードでメッシュ、アーマチュアの順で選択してペアレントの自動ウェイトで連携させる。自動ウェイトを使用することで大体のウェイトを設定できるので使用する。動かしたくないパーツがあれば頂点ペイント、ウェイトペイントで設定していく。

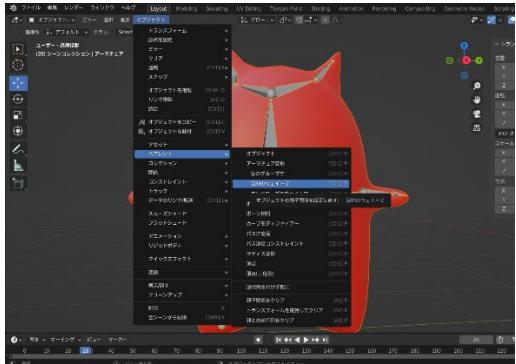


図 102 ウェイトの追加図

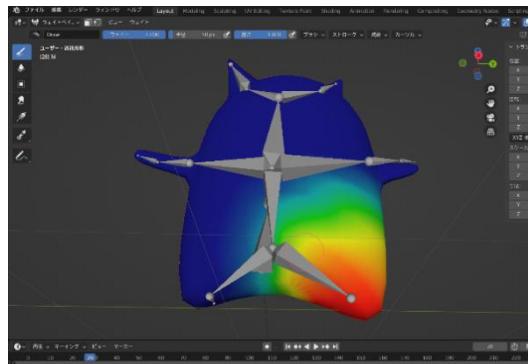


図 103 ウェイトペイントの説明

⑤ アーマチュアの連携ができたら,ポーズモードで確認していく.目や口などがうまくペアレントできず,動かない場合があるのでボーン全体を動かして確かめる.細かいパートなどが2つのボーンとペアレントして変な形になる場合があるので,ボーンの位置を変えてペアレントし,頂点ペイントとウェイトペイントで変更する必要がある.

### 6.3.3 アニメーションの作成

① Blender 上部のタブから Animation を選択する.Animation に移動するときにタイムラインが出現する.タイムラインの左上にあるタブからアクションを選択する.アクションでは動きを分割して数種類作ることができる.タイムラインの中央上にある部分から名前を変更する.今回のゲームではメインキャラが5つ,NPCキャラには3つずつアニメーションがある.どのような動きがわかりやすいように名前を変更していく.別画面を用意しておくと色々な角度から動きが見ることができるので用意しておくとよい.

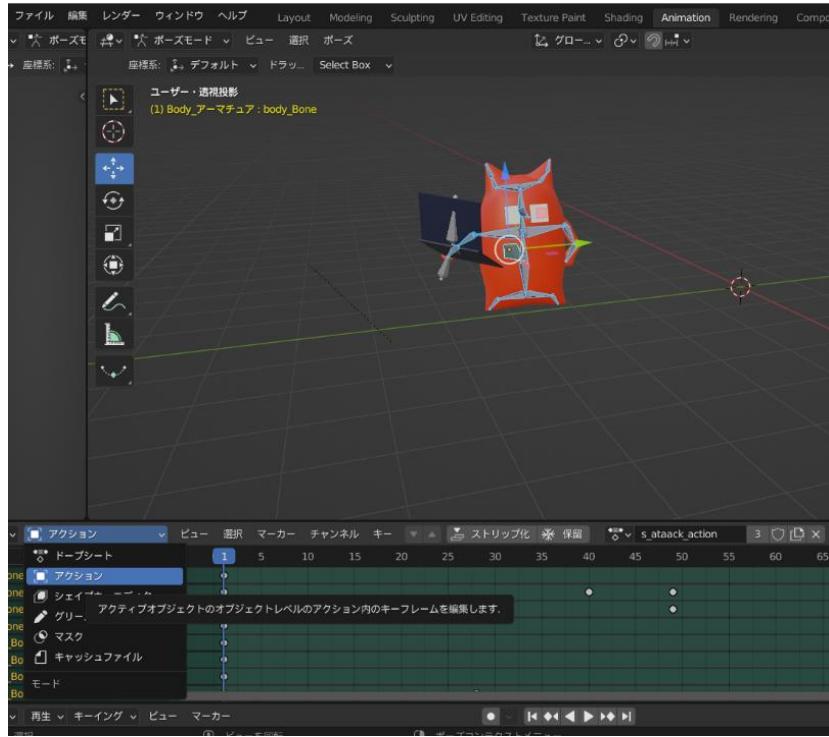


図104 アニメーションの作成

- ② アニメーションはボーンを少しづつ動かし,タイムラインに指定していく.歩く動きを作成する場合は人が歩くイメージをして動きを設定していく.片腕,片足,上半身,下半身と多くのパーツを少しづつ設定していくのでかなり時間がかかる.実際の動きに近づけるために緩急をつける必要がある.腕を振り上げたときは早く動かし,腕を振り上げ終えるときは少しゆっくり動かす.こうすることで実際の動きに近づき,歩いているように見せることができる.動くための座標はUnity側で設定するため移動のアニメーションを作成する際は前に進まず足踏みに形を作成する.歩くアニメーションはループ再生で実行するため短く作成する.
- ③ 一つ目のアニメーションが作成することができたら,名前タブの横にある×ボタンから新しいアニメーションを作成する.攻撃1,攻撃2,攻撃ヒット,通常立ちの動きも作成していく.基本的には①の流れで作る.攻撃とヒットの動きはゲーム側に時間が決められているため長すぎないように作っていく.攻撃やヒットのアニメーションはゲームの中

で座標を移動させないため、アニメーションの時点でキャラクターを移動させて動きをつける。また、アーマチュアを分けて作った際はすべてのアクション名を被らないようにする必要がある。

探索時に使用するアニメーションは座標を変えない。

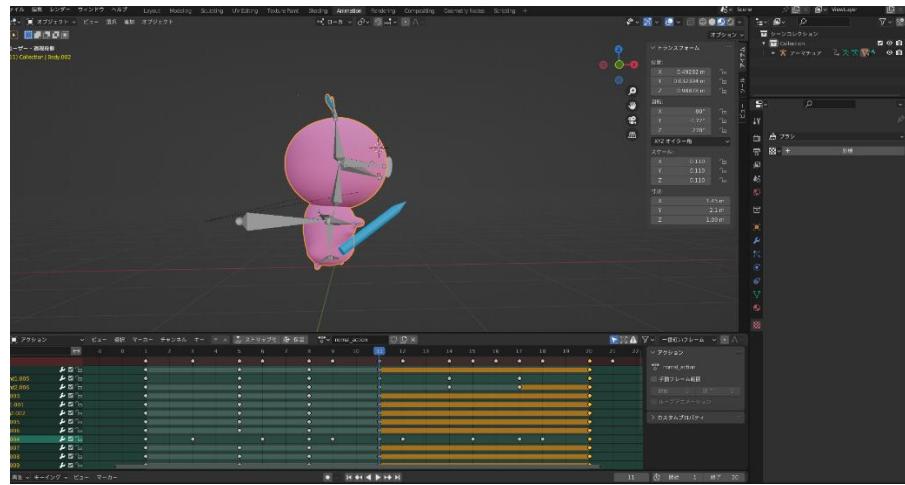


図105 歩き方,通常のアニメーション

バトル時に使うアニメーションは座標を変える。

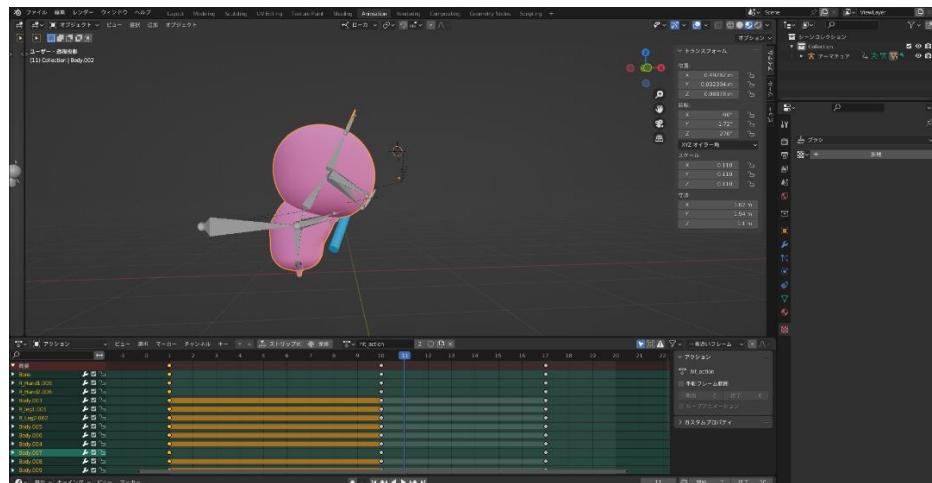


図106 攻撃,ヒットアニメーション

### 6.3.4 Unityへのエクスポート

Blender上部にあるタブの中からファイル、エクスポート、FBXを選択する。今回ゲームで

使用するのはアーマチュア(アニメーション入り)とメッシュの2つなのでカメラやランプは選択しないようにする。「スケールを適用」のなかにある全ローカルを「FBX 単位スケール」に変更する。こちらを変更することで Unity 側のサイズが全て同じになるので Unity 側での変更が楽になる。トランスフォームタブの中にある「トランスフォームを適用」にチェックを入れる。こちらを適用しないと Unity 側の Z 軸が 90 度曲がった状態になり、ゲームに支障ができる。下の方にある「アニメーションをベイク」にチェックを入れる。こちらにチェックを入れることでアーマチュアにアニメーションが連携され Unity 側で使用することができる。下にある「FBX をエクスポート」を選択し、FBX ファイルを作成する。

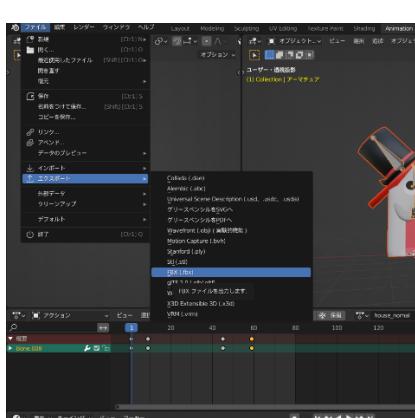


図 107 FBX ファイルの作り方

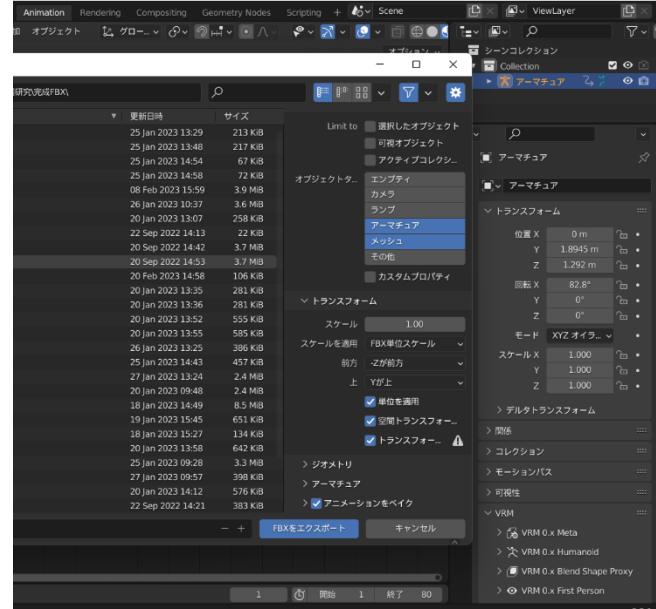


図 108 FBX のインポート、マテリアルの設定

- Unity は Blender のマテリアル、シェーダーに対応していない。Unity 上で一から設定していく必要がある。初めに Unity 側の設定を行っていく。今回は 3D アニメのようなキャラクターにしたいと思う。キャラクターの色合いをトゥーン調にしていく。トゥーン調の色合いを使うには Unity Chan Toon Shader をダウンロードする必要がある。Unity

Chan Toon Shader は細かい設定をすることなく色合いの調整ができるので使用しました.公式サイトから Unity Package をダウンロードして任意の場所で解凍をする.中に入っている Package.json を Unity にダウンロードします.上部のタブから Window→Package manager→+→Add package from disk を選択し,Package.json をダウンロードします.URP を使用しない場合は別バージョンをダウンロードする必要がある.

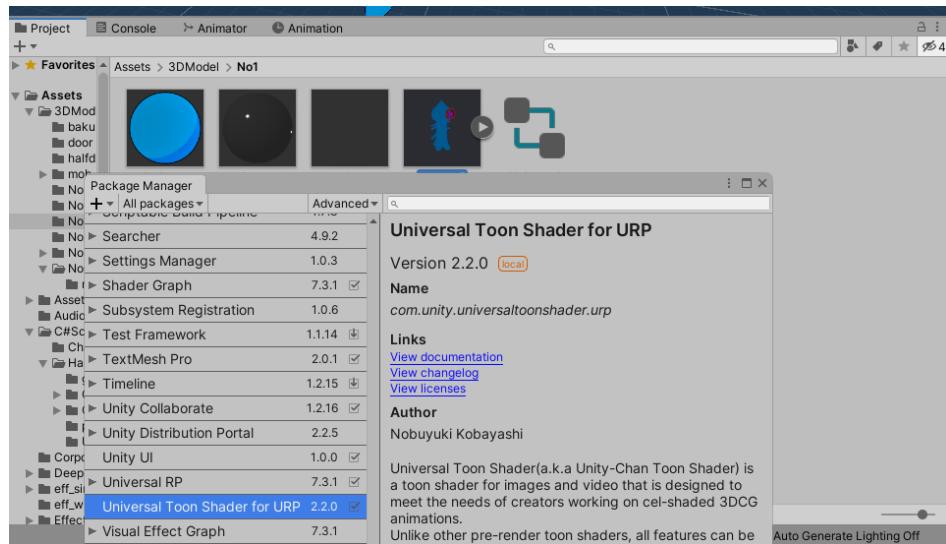


図 1 0 9 Toon Shader の導入

- ② 先ほどダウンロードした Toon Shader を使用してマテリアルを作成していく.Create→Material を選択してマテリアルを作成する.上部のタブから Universal Render Pipeline→Toon Shader を選択する.Toon Shader の設定を行えるようになるのでキャラクターに合わせて細かく値を決めていく. 基本的な色付けは BaseMap, 1st shademap, 2nd shade map の値を変更して色を決めていく. Base map がベースとなる色で, 光が当たっている部分の色になる. 1st shademap, 2nd shade map が 1 影カラー, 2 影カラーになっていて光の当たらない影の色を設定することができる.Base Color step では先ほど設定した色合いの中間点を調和することができる. 値を下げるときベースカラーと影の

境目がなくなり 3D モデルの色合いになる。今回はアニメのような 2 次元彩色にしたいので 0.5 より下げないようにする(キャラクターの材質によっては 0.5 より下げる)。

Blender の作成段階で細かくマテリアルを設定していると、その分 Unity 側でも変更することができる。

③ キャラクターによってはアウトラインをつけていく.outline settings から outline color の色を変更する。画面上のキャラクターを見ながら outline width の値を変更していく。サイズが大きいキャラクターほど outline width を大きくすることで迫力が出るようになる。

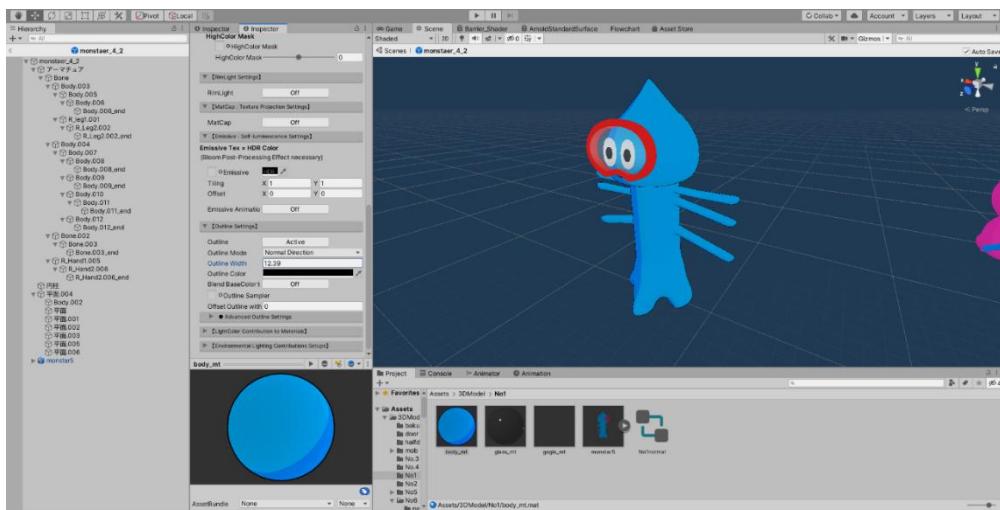


図 110 Toon Shader の設定

### 6.3.5 アニメーションの設定

Blender からインポートしたモデルを選択して下部にある Add Component を選択して Animator を選択する。Project 内に Animator Controller を作成する。先ほど設定した Animator と Animator Controller を連携し、Blender で作成したアニメーションをしようすることができる。

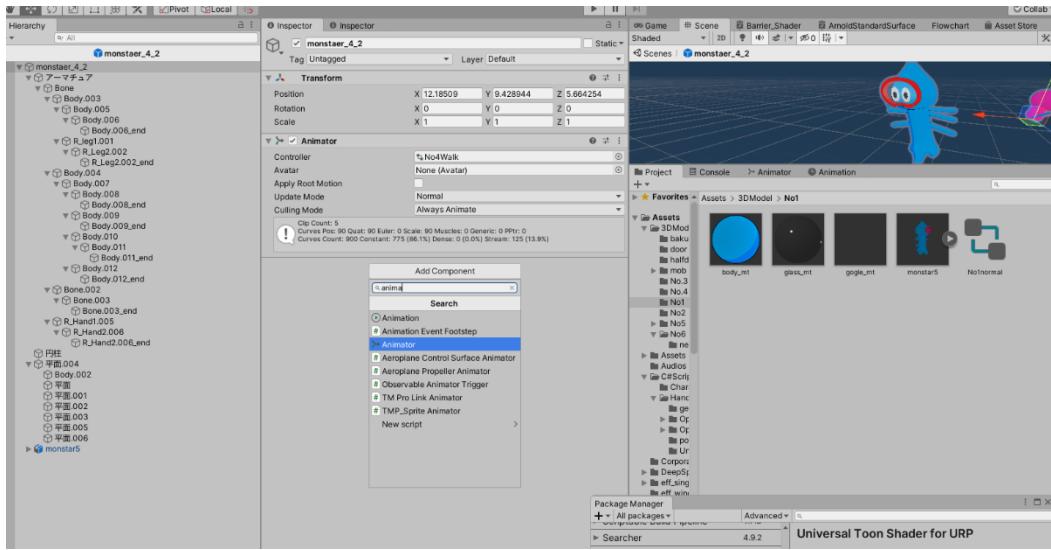


図 111 Animator の追加

### 6.3.6 Animator の設定

Animator を開き,Entry とアニメーションを連携させる.通常状態のアニメーションを用意しているので Entry と normal(通常状態)を連携させる.normal は設定で loop 状態にしておく.loop させることでキャラクターは棒立ち状態ではなくなり,ゲーム感を演出することができる.layers の中にアニメーションを一つずつ入れていく.create state で Empty を選択して,Motion にアニメーションを連携してする.アニメーションを連携させると名前が自動で気に変更されるので設定する必要はない.layers に出現させた攻撃 1,攻撃 2,攻撃ヒット,歩くアニメーションを normal に連携させる.

Normal を選択して Make Transition を選択する.Normal から矢印が出現するのでそれぞれのアニメーションに相互に繋げていく.Layers の隣にある Parameters の選択する.上部にある+で walk,attack1,attack2,hit と名前(任意のもの)をつける.矢印を選択すると inspector にアニメーションを設定するタブが現れる.下部にある Conditions のタブをアニメーションに合ったものに変更する.攻撃 1 の場合は Attack1 を選択し,false,もう一つの矢印を同じように Attack1 の true にする.このように設定することで false の場合はア

メーションが実行されず、プログラム上で Attack1 が true になったら攻撃 1 のアニメーションが実行される。攻撃アニメーションは loop になっていないので一度だけ再生される。

アニメーションを確認するには Parameters のそれぞれの名前の横にあるチェックボックスにチェックを入れるとアニメーションが実行される。

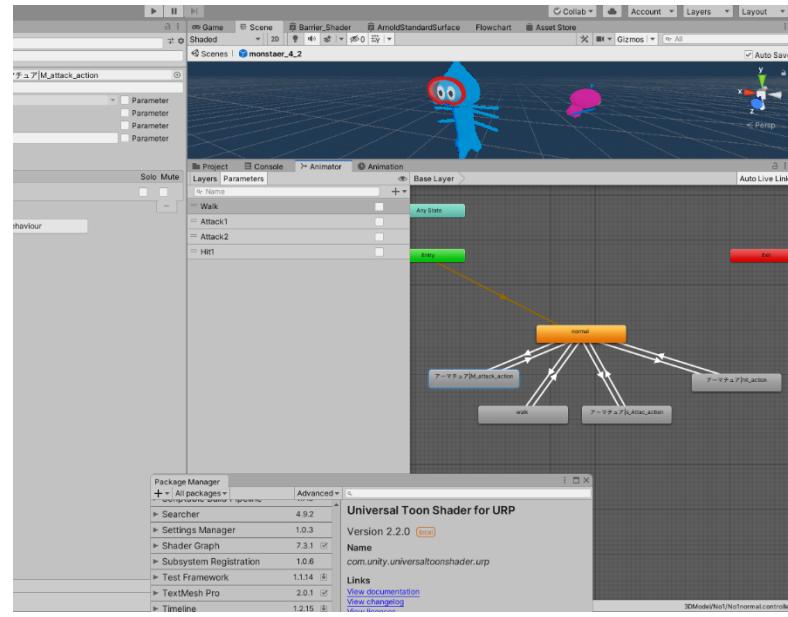


図 1 1 2 Layer の作り方

# 第7章 エフェクトの作成

## 7.1 使用技術

### 7.1.1 Blender

Blender ではパーティクル機能で使うメッシュ,テクスチャを作成する.

### 7.1.2 Unity

主にパーティクル機能を使用する.エフェクトによってはシェーダー,VEG を使用する.

## 7.2 制作過程

エフェクトを作成するにあたって Unity の既存機能であるパーティクルを使用していく.  
パーティクルは小さな粒子を飛ばす機能である.この粒子の飛ばす方向,時間,速さ,出現する  
までの時間,消える速さ,色の変わり方など細かい値を設定します.この粒子を Blender で作  
成したテクスチャをマテリアルにして置き換えることにより,ゲームでよく見るエフェクト  
を作ることができます.作業する手順はキャラクター作成より少ないが一つの手順に時間が  
かなりかかる場合がある.

## 7.3 エフェクトの作成方法

### 7.3.1 Blender でのテクスチャ作成

Blender の上部タブから Texture paint を選択する.画像から新規を選択し,縦横 256px ×  
256px(サイズは任意)で作成する.黒い画像が表示されるのでアルファ消去で透明になる  
ように全体を消す.エフェクトに使用する画像をペイントで作成する.拡散する光を演出す

るためにはペイントの減衰を滑らかにする。テクスチャは必要によって一枚から五種類ほど用意する。背景が透明化の状態で出力されるのでテクスチャはこれ以上設定する必要はない。

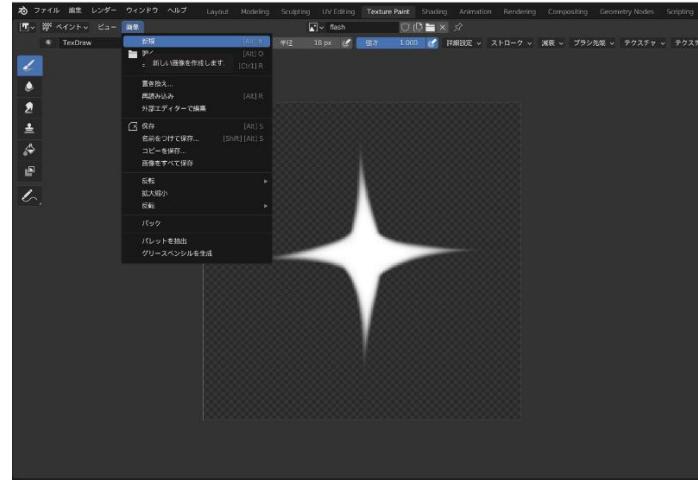


図113 テクスチャの作成

### 7.3.2 メッシュの作成

パーティクルでテクスチャの代わりにメッシュを使用する場合がある。メッシュもBlenderで作成します。キャラクターの作り方でインポートするとメッシュが機能しないので、Blenderで頂点の再設定をする必要がある。

### 7.3.3 パーティクルの設定

①先ほどBlenderで作成したテクスチャをインポートする。Create→Materialでマテリアルを作成する。作成したマテリアルにインポートしたテクスチャを連携させる。この作業をすることでテクスチャをマテリアルの粒子としてパーティクルが放出することができる。Inspectorの右側のSelectからテクスチャの変更ができる。

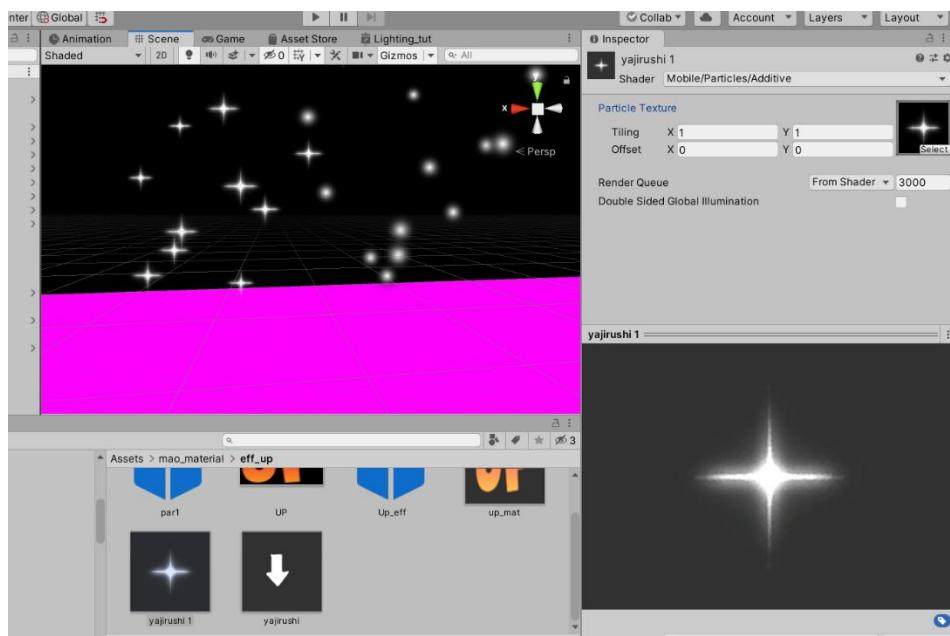


図114 パーティクルテクスチャの変更

使用するテクスチャがある分,上記のようなマテリアルを作成し,パーティクルを連携させていく.コピーで複製することもできる.

② ここからはパーティクルの設定を行っていく.設定する項目がかなり多いため,頻繁に使用するものを下記に記述していく.

1. Duration パーティクルの表示時間を指定する.
2. Looping アニメーションを繰り返すか否かのチェックボタン.チェックを入れると,繰り返しアニメーションが表示される.
3. Start Delay アニメーションが開始されるまでの待ち時間.設定した時間分,遅れてアニメーションが開始される.
4. Start Lifetime パーティクルの表示時間を指定する.
5. Start Speed パーティクルの表示速度を指定する.
6. Start Rotation パーティクルの開始時の向きを指定する.
7. Start Size 1つ1つのパーティクルのサイズを指定.値が大きくなるほど,パーティクルも

大きくなる。

8. Start Color パーティクルの色を指定。方形部分をクリックするとカラーピッカーが表示されるので、色を選ぶとパーティクルがその色に変わる。

9. Max Particles 画面内に表示するパーティクルの最大数。パーティクルが消えて数が減ると、新しいパーティクルが表示される。

パーティクルの数値変化方式 上記の設定の右側にある▼を選択すると下記のようなタブが出現する。

10. Constant デフォルトの設定で、定数に指定する。

11. Curve グラフを描き、そのグラフを元に値が変化する(詳しくは後述)。

12. Random Between Two Constants 2つの数値を入力し、その間の値がランダムに設定される。

13. Random Between Two Curves 2つのグラフを用意し、その間の値がランダムに設定される。

Curve と Random Between Two Constants はほとんどのエフェクトに使用している。

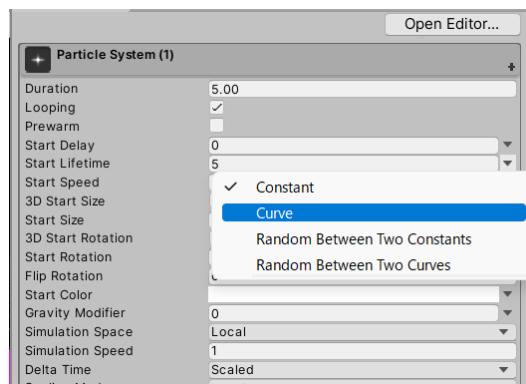


図 1 1 5 Curve の説明

Curve か Random Between Two Curves を選択すると、以下のようなグラフが表示される。

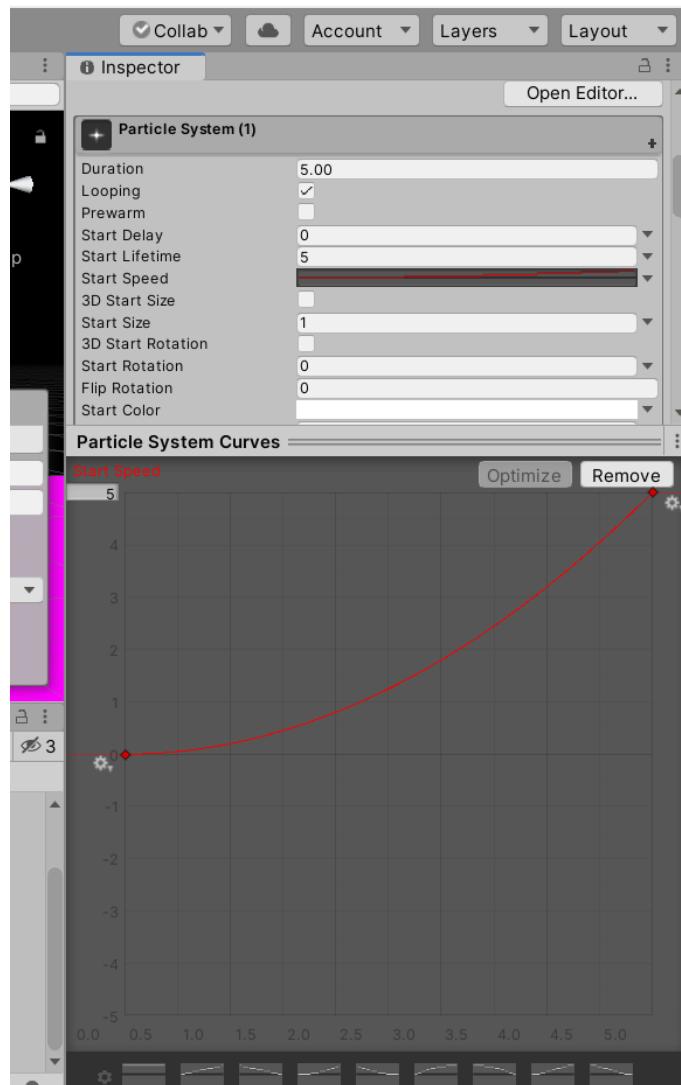


図 116 Curve の編集

グラフ赤線の左端がスタート地点,右端が終了地点になっている. Start Speed で例えると初めはゆっくり表示され,段々早く表示されていくグラフになっている. Start Size で例えると初めは小さな粒子が出現し,後半へ近づくにつれ大きな粒子が出現する.このように動きに緩急をつける,段々とゆっくり変更させたい場合に使用する.

赤線の端や間の点をドラッグ&ドロップで動かすことで,線を傾けたり曲線にしたりすることができ,数値がそのグラフのように変化する.下部にあるデフォルトを使用してもよい.

14. Emission パーティクル放射の頻度とタイミングを定義する. プラスからイベントの作成が可能で指定した時間に指定した量のパーティクルを放出することができる.

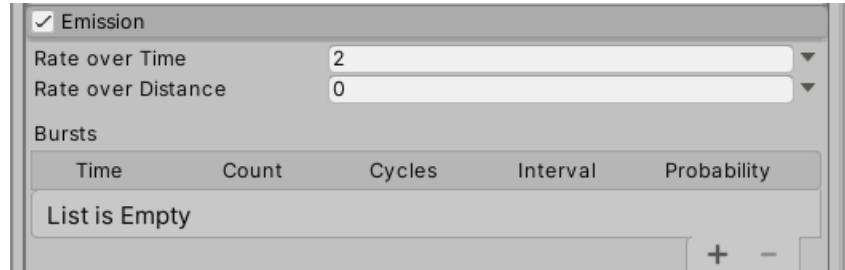


図 1 1 7 Emission の編集

15. Shape パーティクルを放出する体積や面, および開始速度の方向を定義する. Shape のタブから Sphere に設定するとパーティクルは 360 度すべての方向に放出される. Cone はパーティクルを放射状に放出し, Mesh は表面の法線の方向に放出される.

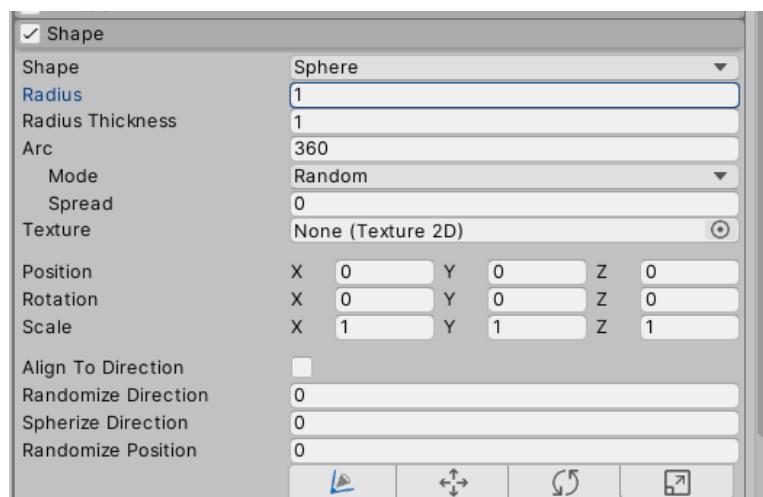


図 1 1 8 Shape の編集

16. Color Over Lifetime パーティクルの色と透明度が生存期間の時間経過によってどう変化するかを設定する.

17. Color 寿命の時間経過で変わっていくパーティクルのカラーグラデーション. グラデーションバーの左端はパーティクルの寿命の開始点を示す. 右端はパーティクルの寿命の終点を示す. 下の図ではパーティクルが透明の状態で出現し, 中心を境に時間がたつにつれ色が

透明になり,終了するときには見えなくなる.

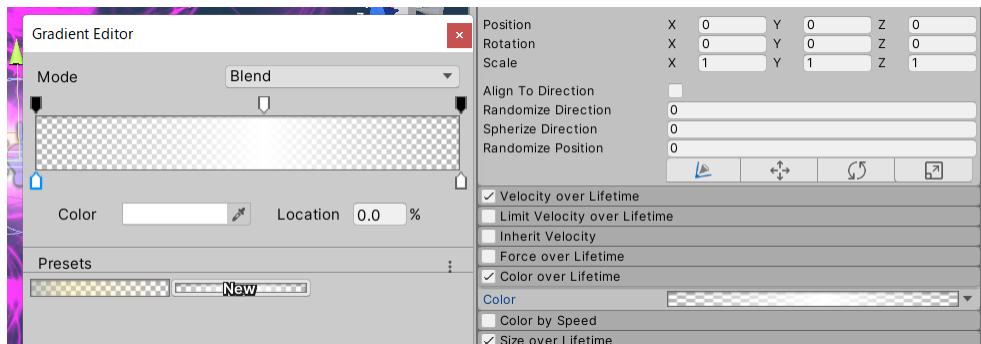


図 1 1 9 Color over Lifetime の編集

18. Size Over Lifetime Curve を使用し大きさを変える設定する.



図 1 2 0 Size Over Lifetime の編集

19. Noise パーティクルの動きに乱気流/ノイズを追加する.

20. Separate Axes 強さを制御し,各軸に別々に再マッピングする.

21. Strength 生存期間にわたりパーティクルにどれだけ強いノイズ効果を与えるかを定義するカーブ. 値が高いほど, パーティクルは速く遠くに移動する.

22. Frequency 値が低いとソフトで滑らかなノイズを作り, 値が高いと急速に変化するノイズを作る. パーティクルが移動するときに方向を変える頻度と, 方向転換の素早さを制御する.

23. Noise 使用することで不規則な動き(煙や火, 渦を巻く)をするエフェクトを作成することができる.

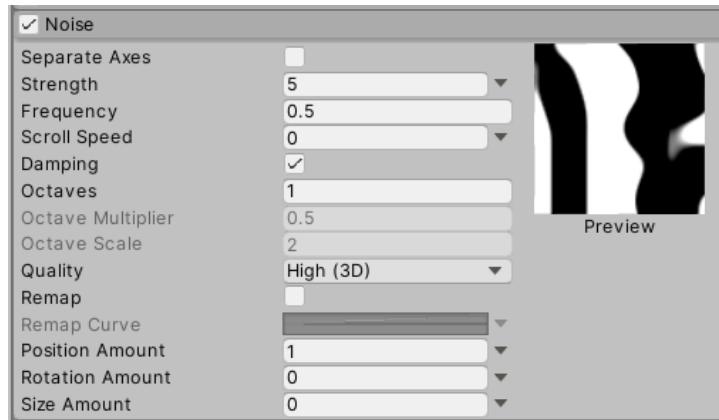


図 1 2 1 Noise の編集

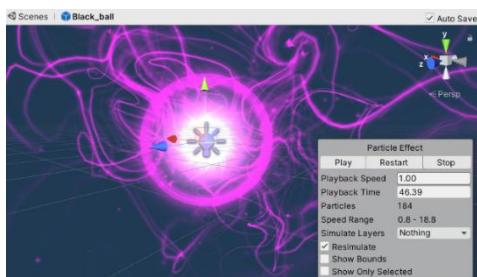


図 1 2 2 Noise on の状態

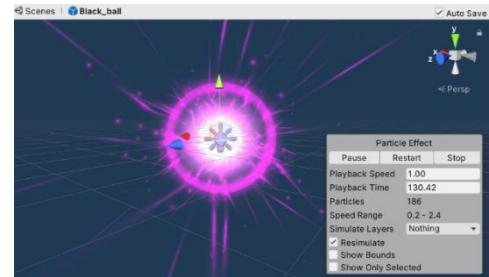


図 1 2 3 Noise off の状態

24. Trails パーティクルにトレイル機能(軌跡)を付与することができる.
25. Mode パーティクルシステムのトレイルの生成方法を選択する.Particle モードは,各パーティクルがその通り道に静止したトレイルを残す効果を作る.Ribbon モードは,各パーティクルの年齢に基づいてつながるリボン状のトレイルを作成する.
26. Ratio 0 から 1 の範囲の値で,トレイルが割り当てられたパーティクルの比率を指定する. トレイルはランダムに割り当てられるため,この値は確率を示している.
27. Lifetime トレイルの生存期間を示している. 属しているパーティクルの存続期間の乗数で指定.
28. Die With Particles 有効にすると,パーティクルが消滅するとすぐにトレイルが消える.

有効でない場合,残っているトレイルはその残りの生存期間に基づいて自然に消滅する.

29. Size affects Width 有効にすると,トレイル幅はパーティクルサイズによって乗算される.

30. Color over Lifetime このカーブで,タッチされているパーティクルの生存期間にわたり  
トレイル全体の色を制御する.

31. Color over Trail このカーブで,トレイル全長にわたる色を制御する.

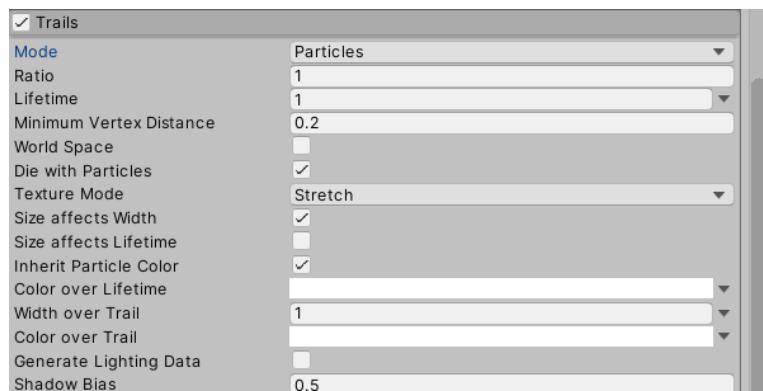


図 1 2 4 Trails の編集

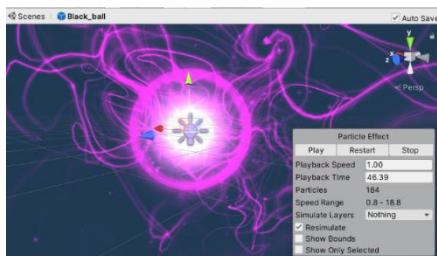


図 1 2 5 Trails on の状態

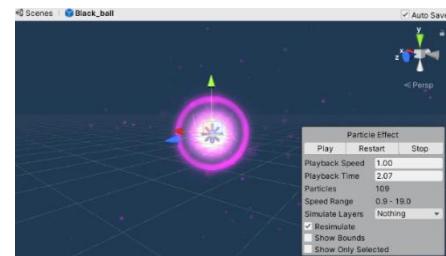


図 1 2 6 Trails off の状態

32. Renderer パーティクルのテクスチャやメッシュがどのように他のパーティクルによ  
つて変形,シェード,上書きされるかを決定します.

33. Render Mode グラフィックイメージ (またはメッシュ) から,画像をレンダリングする  
モード.

34. Billboard パーティクルが常にカメラの方を向く.

35. Sort Mode パーティクルを描画し(上に重なる) 順番.使用可能な値は By Distance (カ  
メラからの距離),Oldest in Front (古いものを表面に), Youngest in Front (新しいものを表

面に) がある.各パーティクルは,この設定によってソートされる.

36. Sorting Fudge パーティクルのソート順のバイアス.値が低いと他のパーティクルを含む透明なゲームオブジェクトの上に描画される可能性が増加する.

37. Min Particle Size 他の設定にかかわらず,もっとも小さなパーティクルのサイズ.ビューポートのサイズに対する割合で表示される.

38. Max Particle Size 他の設定にかかわらず,もっとも大きなパーティクルのサイズ.ビューポートのサイズに対する割合で表示される.

39. Material パーティクルのレンダリングに使用するマテリアル.

40. Trail Material パーティクルのトレイルのレンダリングに使用するマテリアル.Trails モジュールが使用可能な場合にのみ利用できる.

### 7.3.4 シェーダーグラフの作成

① グラフの作成を行う前に Shader Graph ファイルの作成を行うAssets→Create→Shader→Unlit Graph を選択.project にファイルが出現するので任意の名前に変更.Shader ファイルを適用するマテリアルを作るため,Create からマテリアルを作成する.Shader ファイルをドラッグ・アンド・ドロップすると自動的に適用される.

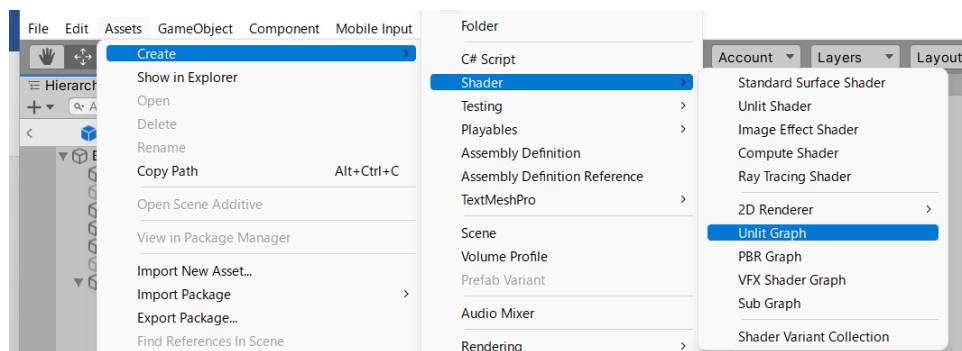


図 1 2 7 Shader Graph の作成

② Shader ファイルの Inspector を開き,Open Shader Editor を選択. Shader Graph の作成

ウィンドが表示されるShader Graph ウィンドは大きく4つの要素で構成されている。

Blackboard(ブラックボード) シェーダーのプロパティとキーワードのリスト。シェーダーで使用する値を変数として登録することができる。値を変更するエフェクトに使用する。

Graph Inspector(グラフィンスペクター) 現在選択中のノードやプロパティを編集するパネル。

Main Preview 現在のシェーダーを適用した時のプレビュー画面。編集する要素はかなり多いので割愛する。

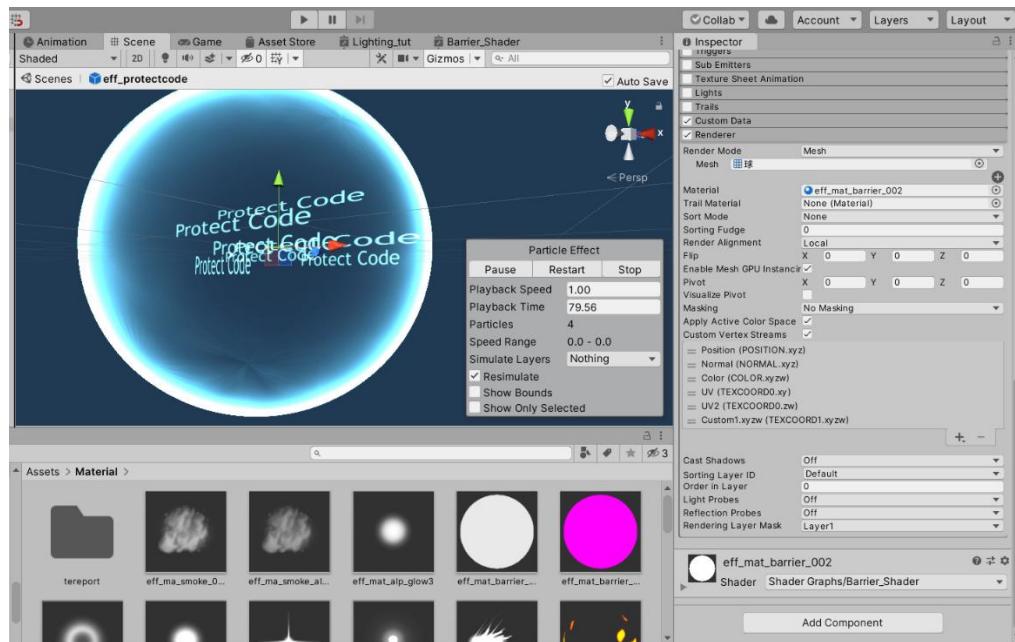


図128 Shader Graph で作成したエフェクト

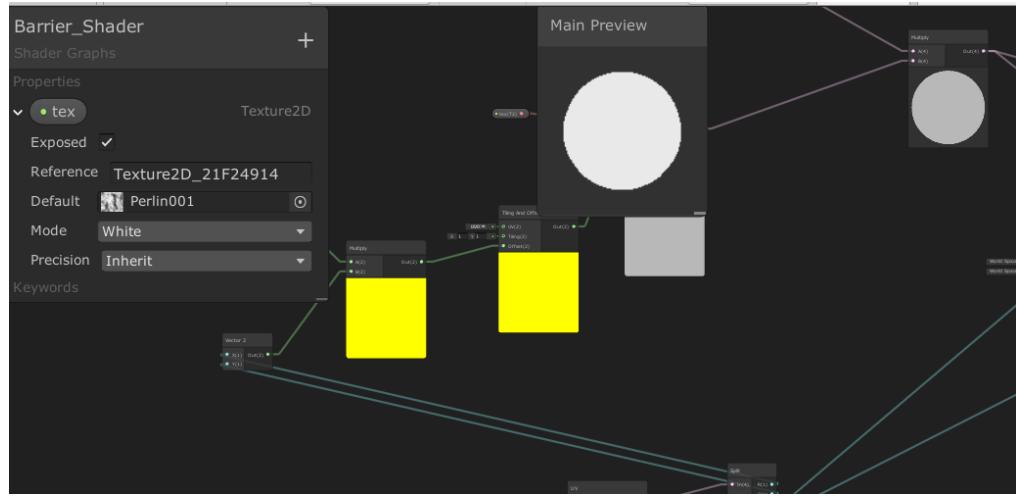


図 1 2 9 Shader Graph の編集画面

### 7.3.5 エフェクトの Prefab 化

任意のファイルを作成し,パーティクルをファイルにドラッグする.prefab(使用した設定をまとめたファイル)を作成し,unity package としてエクスポートすることができる.  
こちらで作成したエフェクトはゲームの戦闘シーン,探索時に使用.

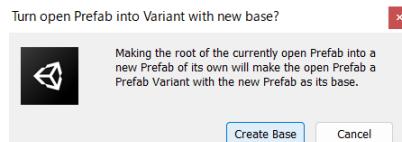


図 1 3 0 Prefab の作成



図 1 3 1 戰闘シーンで使用するエフェクト例

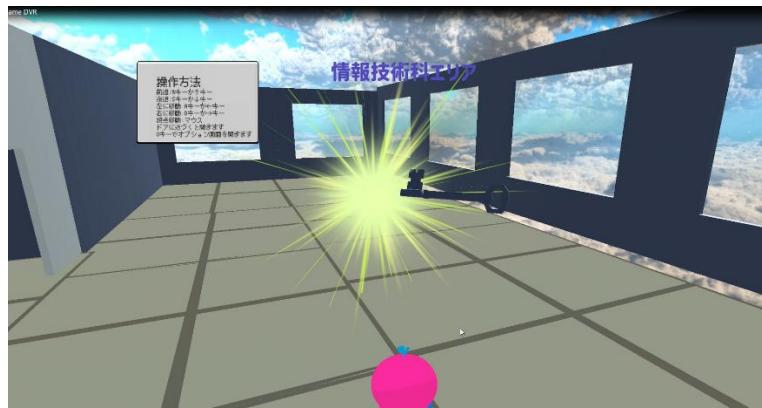


図 1 3 2 探索シーンで使用するエフェクト例

### 1.1.1 エフェクトの実装方法

Unity でエフェクトを実装する方法について記す。

Unity では、エフェクトは Particle System というコンポーネントに分類される。

ここでは例として本ゲームに登場する「アイティーくん」のスキルである「プロテクトコード」に使用したエフェクトを実装する。

エフェクトを実装した際の動作イメージは以下の通りである。

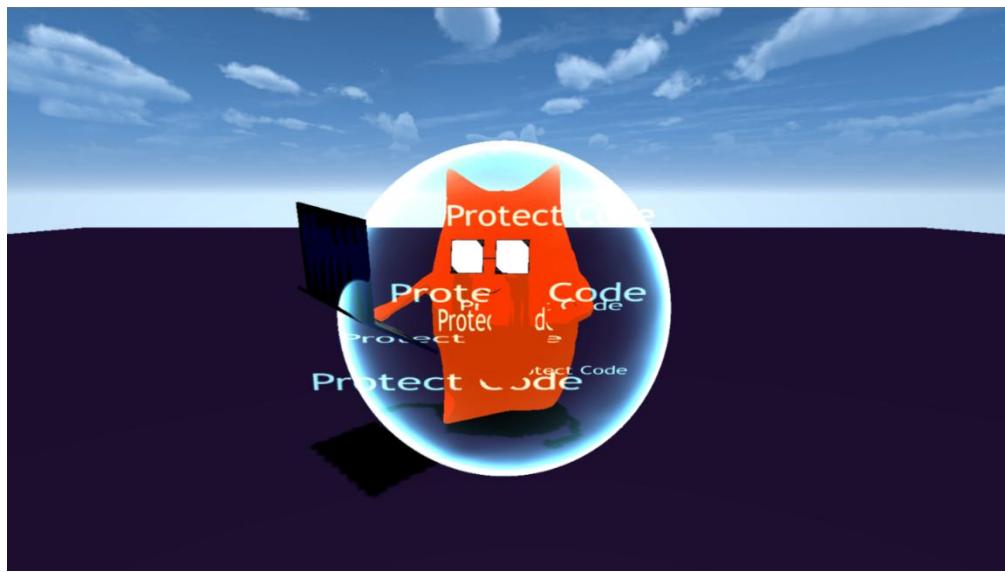


図 1 3 4 エフェクトの動作イメージ

アイティー君がエフェクトをまとうような形で実装するため、アイティー君を配置す

る.(図 2)

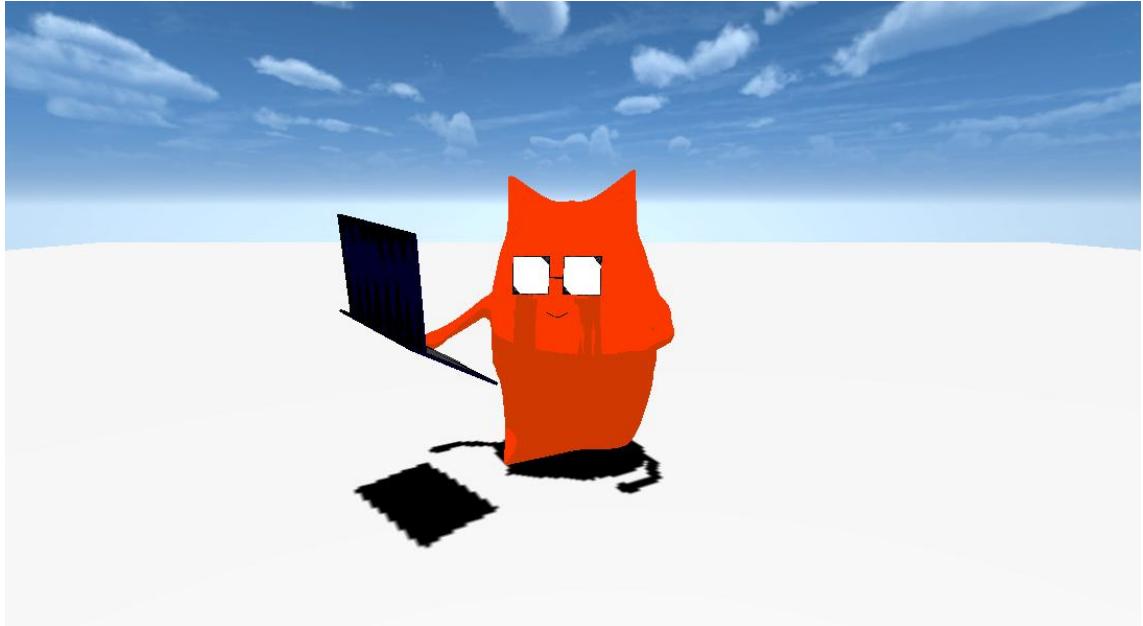


図 135 エフェクトの実装前

ここで使用するプロテクトコードのエフェクトは以下のようなものになる.(図 3)

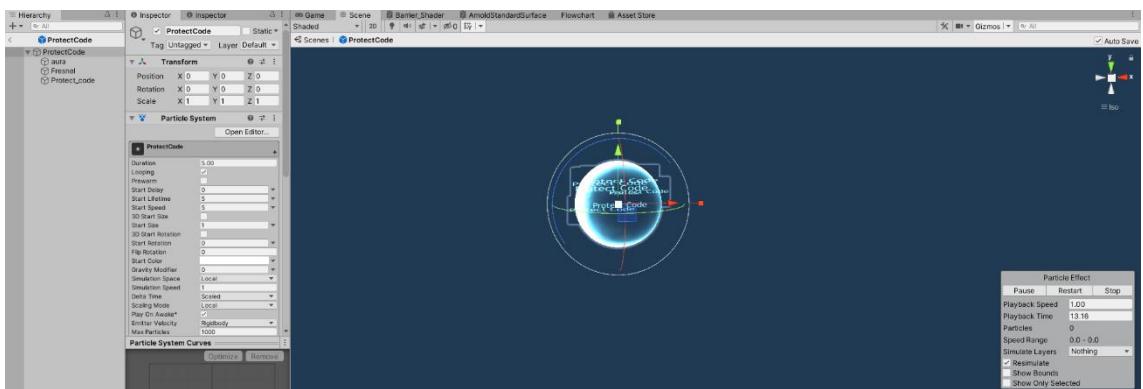


図 136 使用するエフェクト

本ゲームではスキル使用時など,何かをした際にエフェクトを使用するため,エフェクトの再生するタイミングを制御する必要がある.そのため,エフェクトのコンポーネントにある「Play On Awake」のチェックを外しておく.(Play On Awake にチェックが入っているとエフェクトがゲーム起動時から再生されるため.)(図 4)

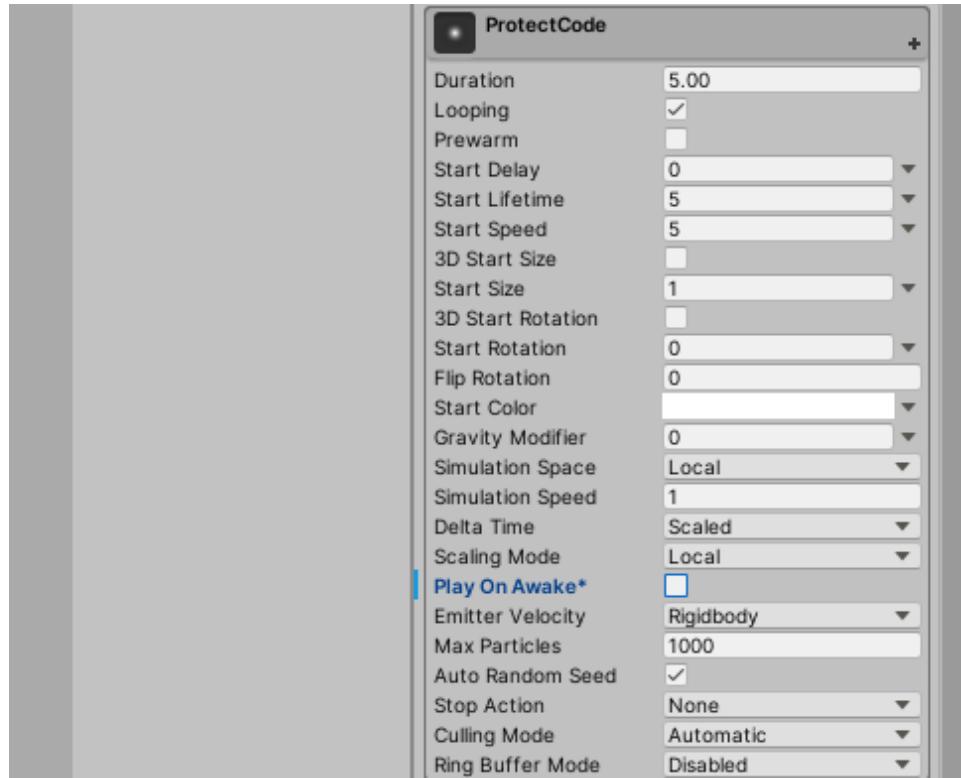


図 137 エフェクトのコンポーネント

上記のエフェクトをアイティー君がまとっているように見える位置に配置する。(図 5)

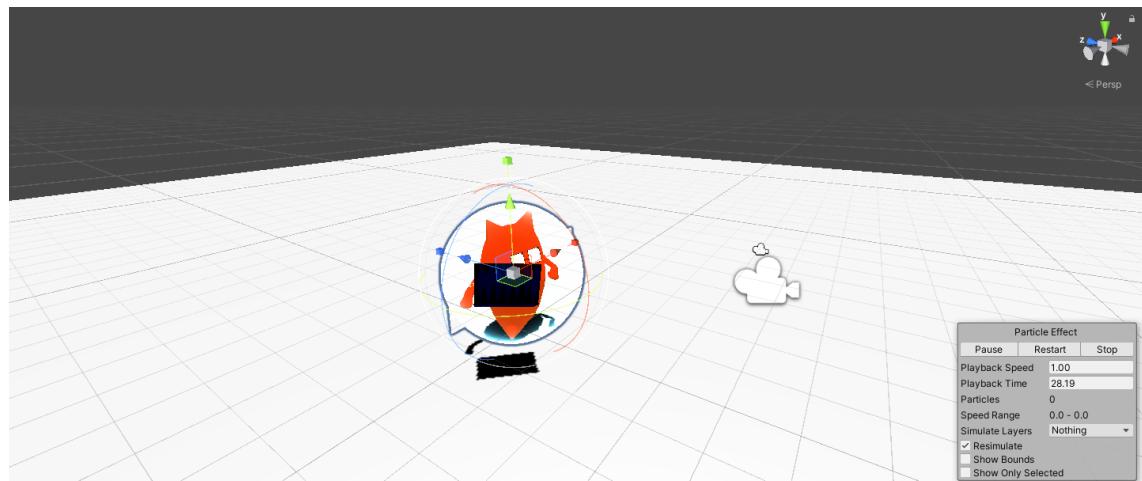


図 138 配置したエフェクト

次にエフェクトの再生するタイミングを制御するスクリプトを作成する。

作成したスクリプトは以下の通りである。

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Effect_test : MonoBehaviour
6  {
7      public ParticleSystem Effect;
8
9      // Start is called before the first frame update
10     void Start()
11     {
12     }
13
14
15     // Update is called once per frame
16     void Update()
17     {
18         if (Input.GetKeyDown(KeyCode.Space))
19         {
20             Effect.Play();
21         }
22     }
23 }
24
```

図139 エフェクトの制御を行うソースコード

エフェクトを Unity 側から指定できるようにするために、エフェクトを public 変数で定義する。(7行目)

スペースキーが押された場合にエフェクトを再生する処理を Update 関数で行う。

Update 関数はゲームが動いている間、常に呼ばれ続けるため、いつスペースキーを押されても反応することが可能。エフェクトの再生を停止する際には Stop 関数を呼ぶ。(16~22 行目)

本ゲームではこの処理をスキルを使用するタイミングで呼ぶことでエフェクトの再生を行っている。

作成したスクリプトを空のゲームオブジェクトなどにアタッチし,スクリプトの変数(Effect)にプロジェクト内のエフェクトを代入する.(図 7)

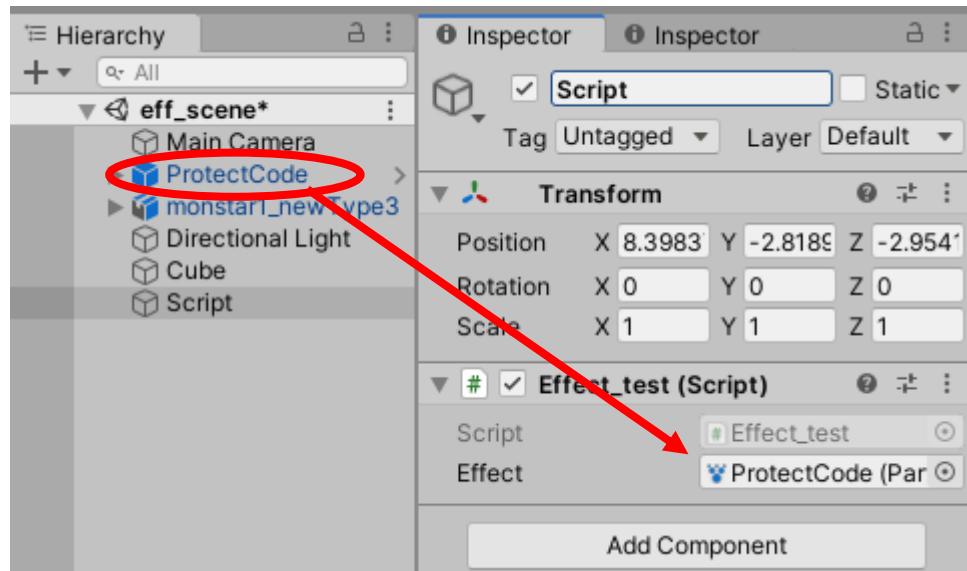


図 140 エフェクトの代入

完成したエフェクトの動作イメージは以下の通りである。

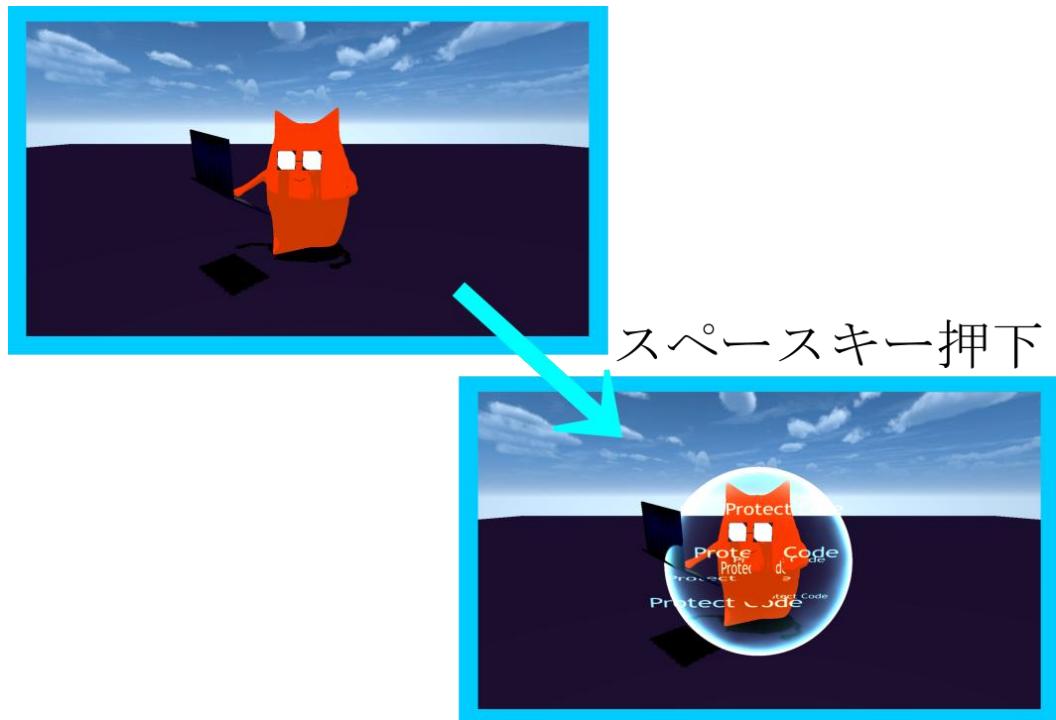


図 141 スクリプトの動作イメージ

# 第8章 ED,PV の作成

ストーリー性のあるゲームを作るにあたって ED ムービーが必要だと思い作成を行った。産技短展や学校見学で来校した人たちにプレイしなくてもゲームの雰囲気が伝わるように PV, 実況の作成も行った。

## 8.1 使用技術

ED ムービー, 紹介映像の作成にはフリー映像編集ソフト Aviutl を使用した。また, 映像の中で使用したグリーンバック素材, ブルーバック素材は Blender で作成した。

### 8.1.1 ED ムービーの作成

Xbox Game Bar でゲーム画面の録画を行う。実況の場合はマイクから音声を取得する。録画をした mp4 ファイルを作成する映像ソフトに取り込み編集を行う。フリーの音楽や Blender で作成したテクスチャを活用して作成した。

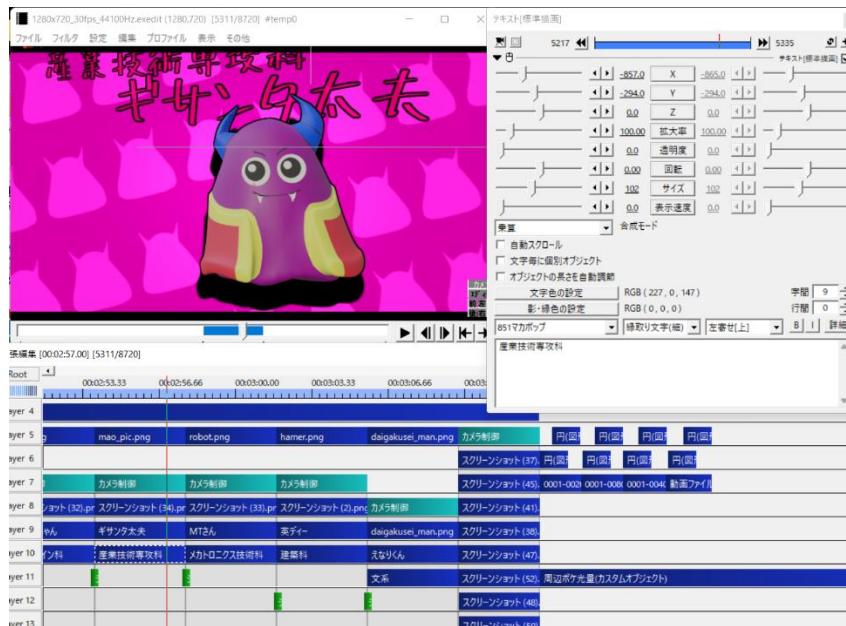


図 142 ED 作成の編集画面

### 8.1.2 紹介映像の作成

ED ムービーと同じ方法で作成した。

### 8.1.3 実況動画の作成

ED ムービーと同じ方法で作成した。

## 第9章 おわりに

今回の研究では,ハンドトラッキングによるプロジェクトの実装が予定より早く進めることができ,ゲーム開発に費やす時間を増やすことができた。

結果として,ただの学科紹介ではなく,誰もが楽しめる,ユニークでゲーム性の高いロールプレイングゲームが完成した。

最後になりますがキャラクターを作成し、使用を快く承諾していただいた産業デザイン科千葉未悠さんにお礼を申し上げます。誠にありがとうございました。

## 参考文献

KTK\_kumamoto. ゲーム開発奮闘記. Hatena Blog.

2022-02,<https://ktk-kumamoto.hatenablog.com/>,(参照 2022-12).

Kazuma\_f. [Unity] 3D オブジェクトがピンク色になった時の解決方法. Qiita.

2022-10,[https://qiita.com/kazuma\\_f/items/4d265c5615f1d441ffe9](https://qiita.com/kazuma_f/items/4d265c5615f1d441ffe9),(参照 2022-12).

はこねは.URP にしたら UnityChanToonShader を使ったモデルがピンクになって困った話. Hantena Blog. 2021-11, <https://hakonebox.hatenablog.com/entry/2021/11/05/000230>,(参照 2022-12).

GabrielAguiarProd .Unity VFX Graph - Hits and Impact Effects Tutorial .Youtube.

2020-12, <https://www.youtube.com/watch?v=jSIan1cEYTI>,2023-01.

DOVA-SYNDROME .FREE BGM. Youtube.

2017-12,[https://www.youtube.com/watch?v=6vcSHr\\_3PNY&list=LLop3Ht5lh2aHgIJIUUSWAWyQ&index=8](https://www.youtube.com/watch?v=6vcSHr_3PNY&list=LLop3Ht5lh2aHgIJIUUSWAWyQ&index=8),2023-2.

2022-10,[https://www.youtube.com/watch?v=Yjju\\_UOtsSA&list=LLop3Ht5lh2aHgIJIUUSWAWyQ&index=24](https://www.youtube.com/watch?v=Yjju_UOtsSA&list=LLop3Ht5lh2aHgIJIUUSWAWyQ&index=24),2022-12.

2022-12,<https://www.youtube.com/watch?v=F3ryAoSoauk&list=LLop3Ht5lh2aHgIJIUUSWAWyQ&index=31>,2022-12.

加藤秀樹.【決定版】元カプコンのプロデューサーが教える企画書の作り方.2022/03/19.

<https://game.creators-guild.com/g4c/work-study-20220319/>.参照

2022/10/20

山村達彦.【Unity 道場仙台スペシャル 2】Unity でステージをつくるのじゃ. 2018/02/07. <https://www.youtube.com/watch?v=m6AR5Lysv00>,参照 2022/11/30

sasanon. “Unity で 3D 空間内を TPS っぽく WASD 移動するキャラをつくる”. ささ

み雑記帳. 2017-09-17. <https://sasanon.hatenablog.jp/entry/2017/09/17/041612>. (参照 2022-11-22).

sensiki. “Unity のアセットストアから消えた「Fungus」をインストールする方法”. 好奇心俱楽部. 2021-12-11. <https://trend-tracer.com/fungas-install/>.(参照 2022-1-20).

Kirikabu\_ueda. “【Unity】Fungus でゲーム制作(1)会話シーンを作る”. Qiita. 2020-10-30. [https://qiita.com/Kirikabu\\_ueda/items/bfc6e086d408b1cba34b](https://qiita.com/Kirikabu_ueda/items/bfc6e086d408b1cba34b).(参照 2022-1-20).

エフェクト参考 [GabrielAguiarProd](#)

使用音楽 [DOVA-SYNDROME](#)