

# KILL SHOT THE BEAT

キルショット・ザ・ビート

MARKET SEL

IMPLANT TECH

ROBO CORP

DISCOVER NEW WORLD

MNG INSURANCE

HONEY'S

BARBECUE

WORKSH

OP

EX

RE

SECTOR 36

ID:ASDHJ#M



READER

リーダー / プランナー

新倉 混祐

DESIGNER

デザイナー

村上 慧都 城脇 礼奈

PROGRAMMER

プログラマー

盛田 翔太 小林 勇樹

菅原 渉 大長 ルミナ

山川 恵斗 磯部 功太

# KILL SHOT THE BEAT

# KILL SHOT THE BEAT

コンセプト  
音ゲー+音ハメ+FPS

プレイ人数 / 1人  
プラットフォーム / PC  
開発環境 / Unity  
ゲームルール / Music Game  
コントローラー / キーボード・マウス  
プレイ時間 / 4~5分

スーパーゲームクリエイター専攻3年 新倉滉祐

## このゲームの面白いところ

- リズムに乗って銃を撃ち、敵を倒したときの爽快感

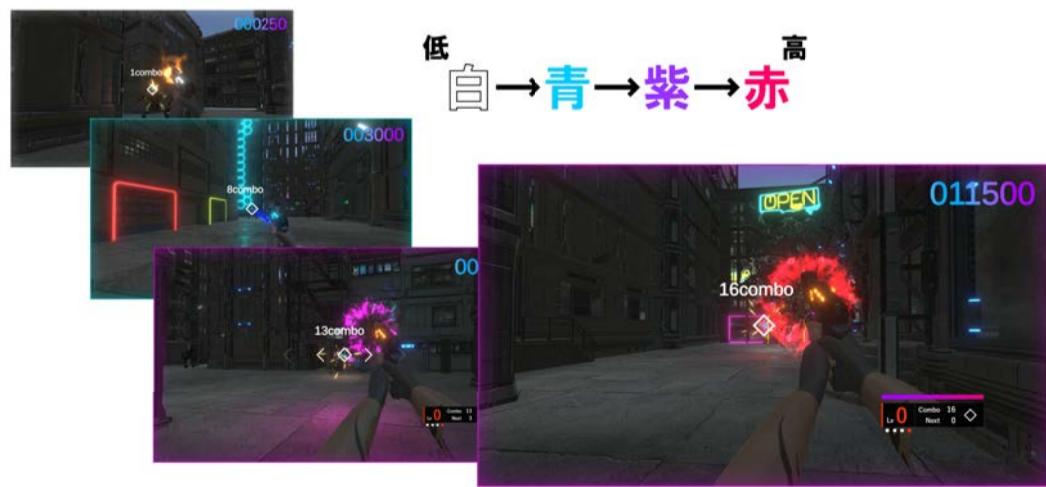


- Dose (状態) で増す爽快感



## dose

上がれば上がるほど派手になるエフェクト



色を上げて無双を楽しもう

企画書

新倉 滉祐

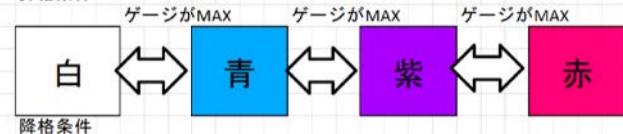
#### 4-5-1 状態の種類

説明 種類は色で判別します。UI・エフェクト・サウンドで表現する予定です。



#### 状態の昇格・降格条件(白が初期状態です)

昇格条件



降格条件 ゲージがMAX ゲージがMAX ゲージがMAX  
ゲージが無くなる ゲージが無くなる ゲージが無くなる

#### 4-5-3 状態ごとで弾を当てたときのスコア

色	白	青	紫	赤
Good	150	250	350	750
Great	250	350	750	1000
Perfect	300	500	1000	1500

#### 4-5-4 COMBOによるスコア加算

説明 プラスアルファで入るスコアです。

0~5COMBO 5~9COMBO 10~14COMBO 15~20COMBO

#### 6-3. エネミー1

説明 エネミー1の仕様です

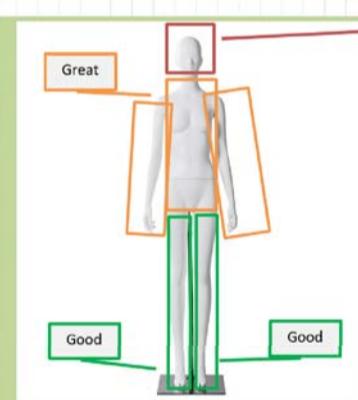


予定 アセットを使用。攻撃モーションも入れたい。弾が当たっただけ反らせたい。アセットできる限り早めに用意します、、、

使用アセット名 Etasphera09

倒れたとき Plasma Grenade

#### 当たり判定



#### ステータス

#	項目	条件・数値
1	HP	100
2	攻撃条件	攻撃範囲に入ったら攻撃
3	攻撃クールタイム	2秒
4	移動速度	120
5	与ダメージ	スコアを-1500
6	倒したときに貰えるスコア	3000
7	ダメージのリアクション	アリ
8	スปーンの頻度	2秒ごとに4体
9	硬直時間	1秒

プレイヤーを100に仮定

コンボモリセット

値は調整予定

スปーン条件が変わったため消去

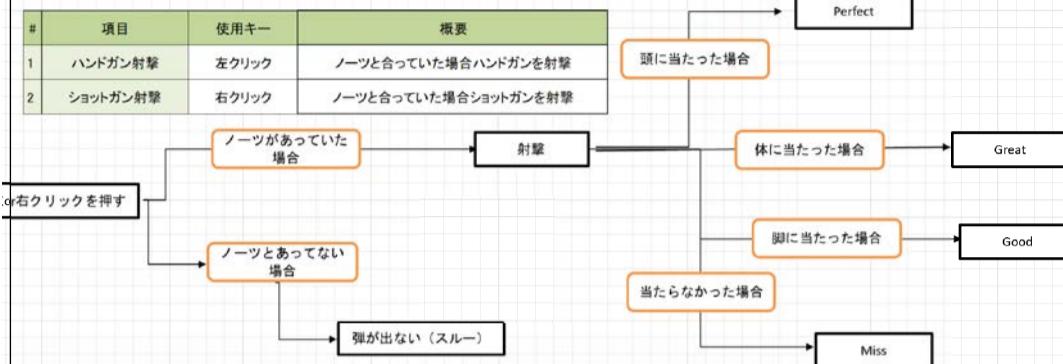
## 仕様書

## 新倉 混祐

企画立案、メインプランナーを担当しました。

仕様書ではプログラマーが作業しやすいように書くことを心掛け、コミュニケーションも短い期間で行いました。リーダーとしてゲームの方針を固めるときデザイナー、プログラマーと衝突することも複数回ありましたが、メンバーの熱意に助けられ最高の作品に仕上げることができました。

#### 4-3.プレイヤー攻撃操作



#### 4-4.その他操作

#	項目	使用キー	概要
1	メニューを開く	ESCキー	ゲームを中断しオプション画面を開く

#### 4-5.攻撃(射撃)

12/20仕様変更

#### 6-3-2.通常(右)ノーツ

見た目	説明
	この作品の基本ノーツの一つ。 画面右側から流れてくる。 右クリックで判定を取り、 成功ならショットガンを射撃、失敗なら射撃できない。

#### 6-3-2.ロングノーツ(チャージ)

見た目 長かったので切り取りで合成したものです。本物はGoogleDriveに入っているので確認してください。  
[https://drive.google.com/file/d/1SlieNR3fQk\\_dsUiD9v7QYCXQF-l5Xwz9U/view?usp=sharing](https://drive.google.com/file/d/1SlieNR3fQk_dsUiD9v7QYCXQF-l5Xwz9U/view?usp=sharing)



#### 説明

この作品の特殊ノーツ。クリック長押しで最初に判定を取り長押し、話したときにも判定がある。  
成功するとチャージショットが発射される。(押したとき、離したときの両方を成功させないと成功判定にならない)

SE	タイトル	クリック時	SE	どうーん的な	△				
	曲選択	曲切り替え時	SE	シュンツルカタッ的な	△				
		曲決定時	SE	ベースの低い音的な	△				
メインゲーム	射撃	SE	射撃音	△	メインゲーム	歩いたとき	SE	歩く音(走る感じ)	×
	敵にHIT(白)	SE	べちっ的な	×	エネミー	攻撃時	SE	ぶおん	×
	敵にHIT(青)	SE	べちっ的な	×	エネミー2	攻撃時	SE	射撃音	×
	敵にHIT(緑)	SE	べちっ的な	×					
	敵にHIT(赤)	SE	べちっ的な	×					
	エネミー破壊	SE	低音	△					
	エネミー破壊(赤)	SE	重低音	△					
リザルト	Nextを押したとき	SE	ベースの低い音的な	△					
ランキング	Nextを押したとき	SE	ベースの低い音的な	△					
END	切り替わった瞬間	SE	読み上げてきな	×					
オプション	オプションを開いたとき	SE	シュインっ的な	×					
	デザインを変更したとき	SE	カタツ的な	△					
	ゲージを左右に動かしたとき	SE	カチカチツ的な	△					
BGM	全体	常時	BGM	曲再生	△	計5			
メインゲーム	スタート時	BGM	曲再生	△					
リザルト	常時	BGM	適当な曲	△					
ランキング	常時	BGM	適当な曲	△					
曲選択	プレビュー	BGM	曲再生	△					
EFFECT	メインゲーム	射撃	Effect	白いマズルフラッシュ	△				
	敵にHIT(白)	Effect	当たった場所に白い火花的な	△					
	敵にHIT(青)	Effect	当たった場所に青い火花的な	△					
	敵にHIT(緑)	Effect	青いマズルフラッシュ	△					
	敵にHIT(赤)	Effect	当たった場所に赤い火花的な	△					
	エネミー破壊	Effect	紫のマズルフラッシュ	△					
	エネミー破壊(赤)	Effect	赤のマズルフラッシュ	△					
		Effect	ボカンできな	△					
		Effect	なんかかっこいいの	△					



PLANNER

ゲームタイトルロゴ / UI デザイン /  
XD 遷移演出デザイン / エフェクトアニメーション /

村上 慧都

ゲームタイトルロゴ

Adobe Photoshop/Illustrator

サイバーパンクな世界感を意識し、ネオン管をイメージしたデザインに決定しました。複数のレイアウト案を出し、その中から一番まとまりのある形に決めました。

KILL SHOT THE BEAT

KILL SHOT THE BEAT

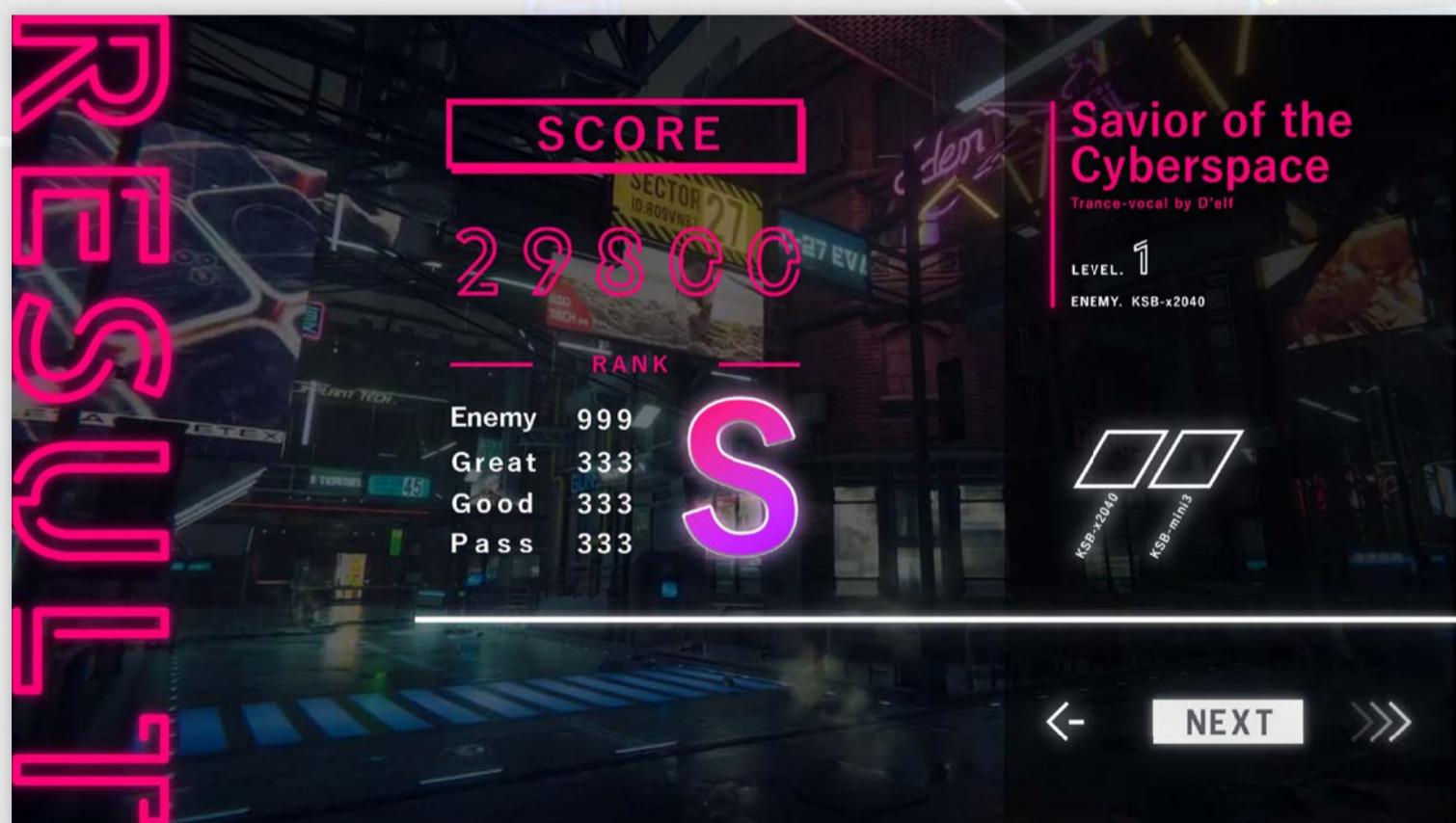


KILL SHOT  
THE BEAT

KILL SHOT  
THE BEAT

KILL SHOT  
THE BEAT  
キルショット・ザ・ビート

DESIGNER

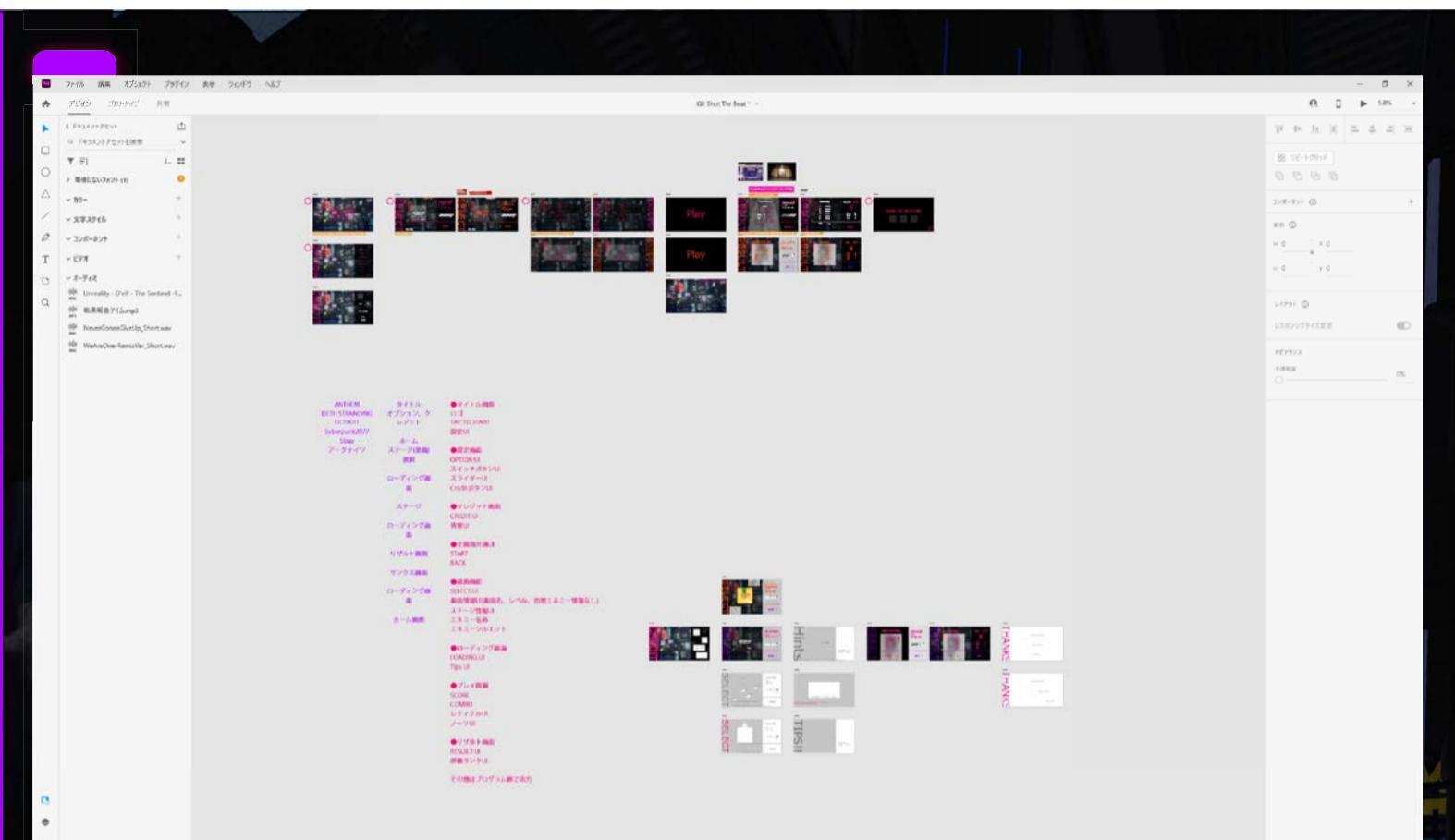


S A B C

UI デザイン  
Adobe Photoshop

一つずつコンセプトデザインを作成し、プログラマーへ共有しました。  
完成形を先に視覚的に提示し、そこからプログラムで出来る部分と  
デザイナーが作る部分に分けて作成しました。

DESIGNER



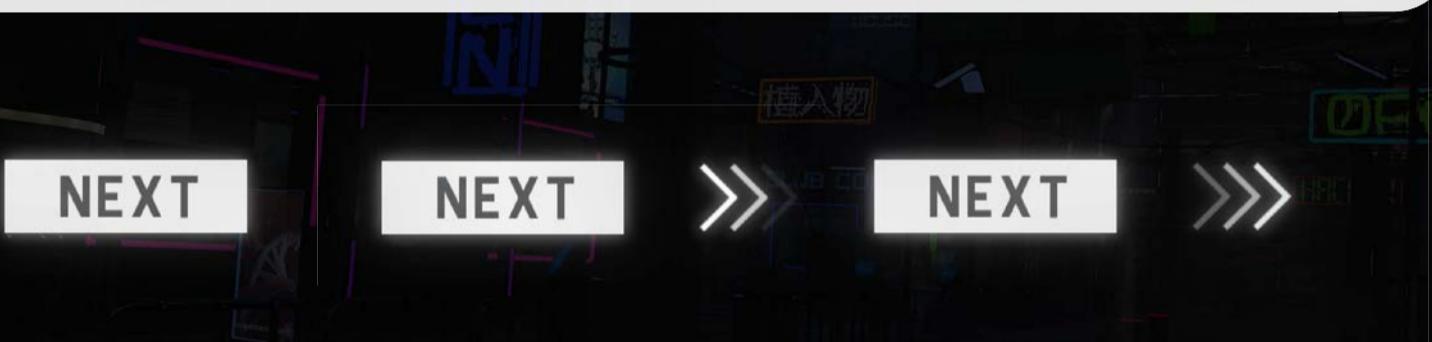
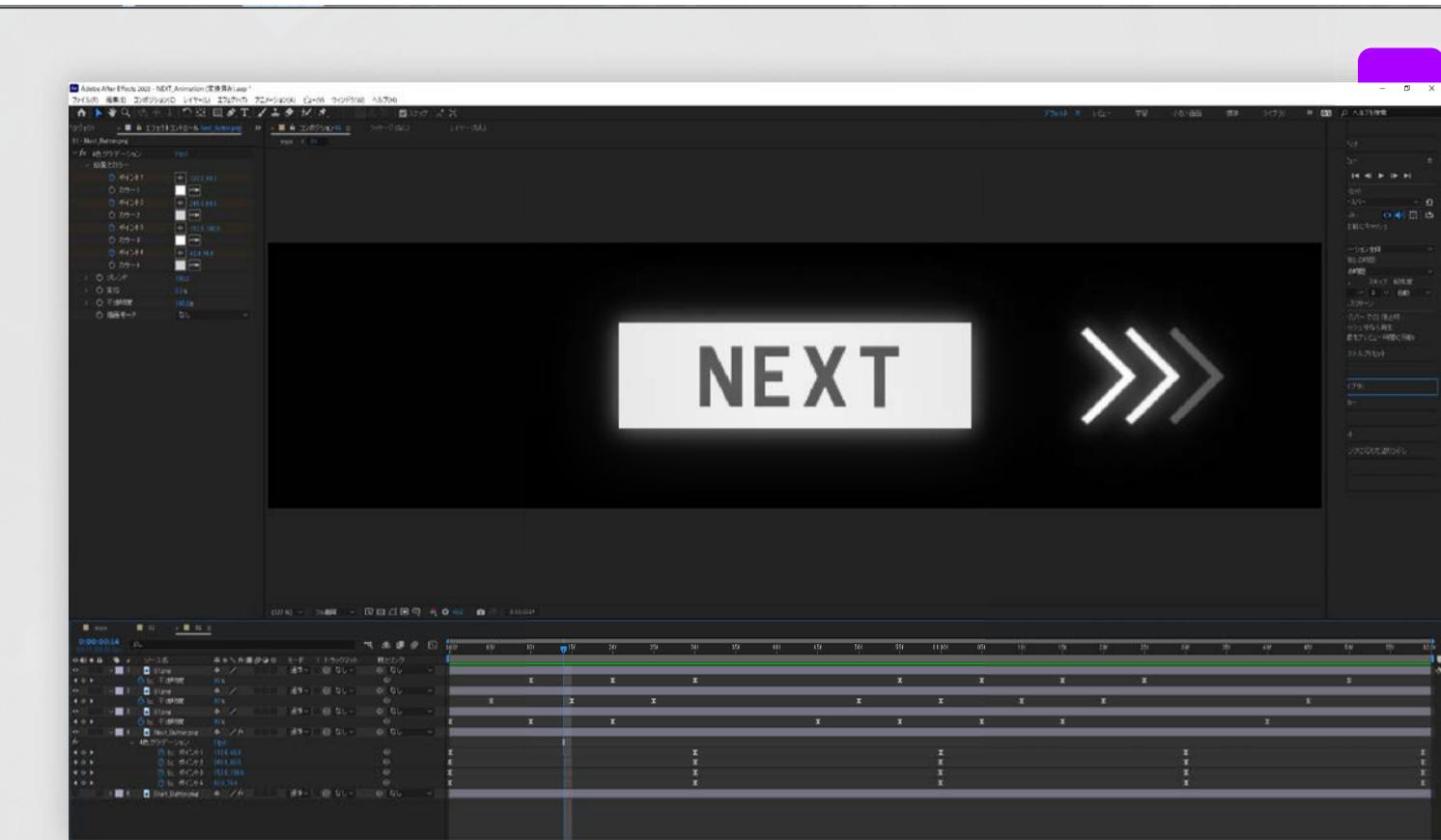
## XD 遷移演出デザイン

Adobe XD

大まかな UI をワイヤーフレームで作成し、Photoshop や AfterEffects で画面を作成していきました。SE やアニメーションなども実装し XD 上で実機のように確認可能にしました。



DESIGNER



**エフェクトアニメーション**  
Adobe AfterEffects/photoshop

アニメーションを AfterEffects で作成しました。単純動作のものは作成した Ae を元にプログラマーが、グリッヂなどの複雑な物は動画でそのまま実装しています。

**SCORE**



DESIGNER



曲ジャケット

CLIP STUDIO PAINT PRO

城脇 礼奈

楽曲選択画面で表示されるジャケットイラスト2枚を担当しました。

ゲームの世界観にマッチするようネオン街をイメージしたビビッドカラーを基調に、路地裏のグラフィティやステンシルアート風に仕上げました。



DESIGNER

# WE ARE ONE

# WE ARE ONE

# WE ARE ONE



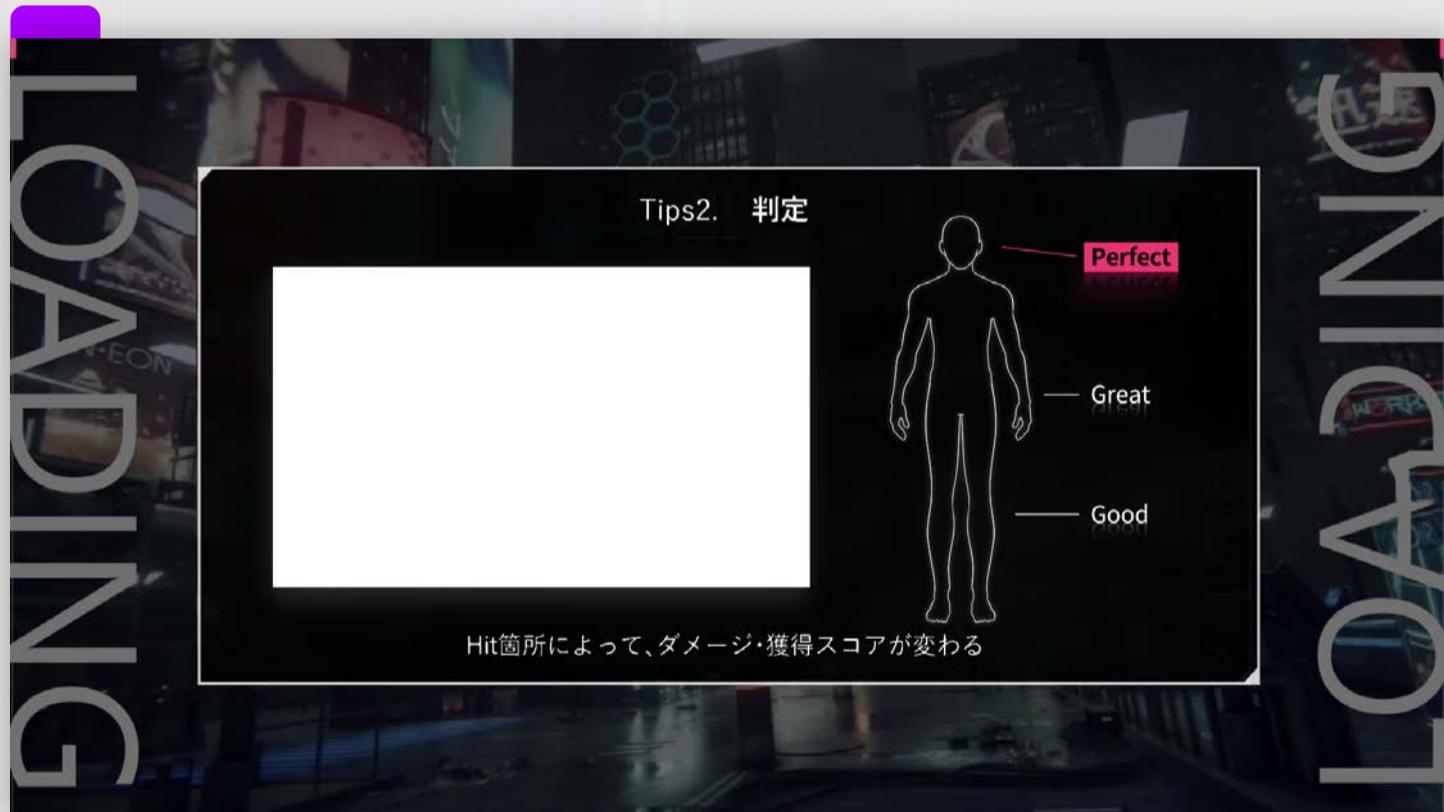
# SAVIOR OF THE CYBER SPACE

# SAVIOR OF THE CYBER SPACE

Savior of the Cyberspace



DESIGNER



## UI デザイン

城脇 礼奈

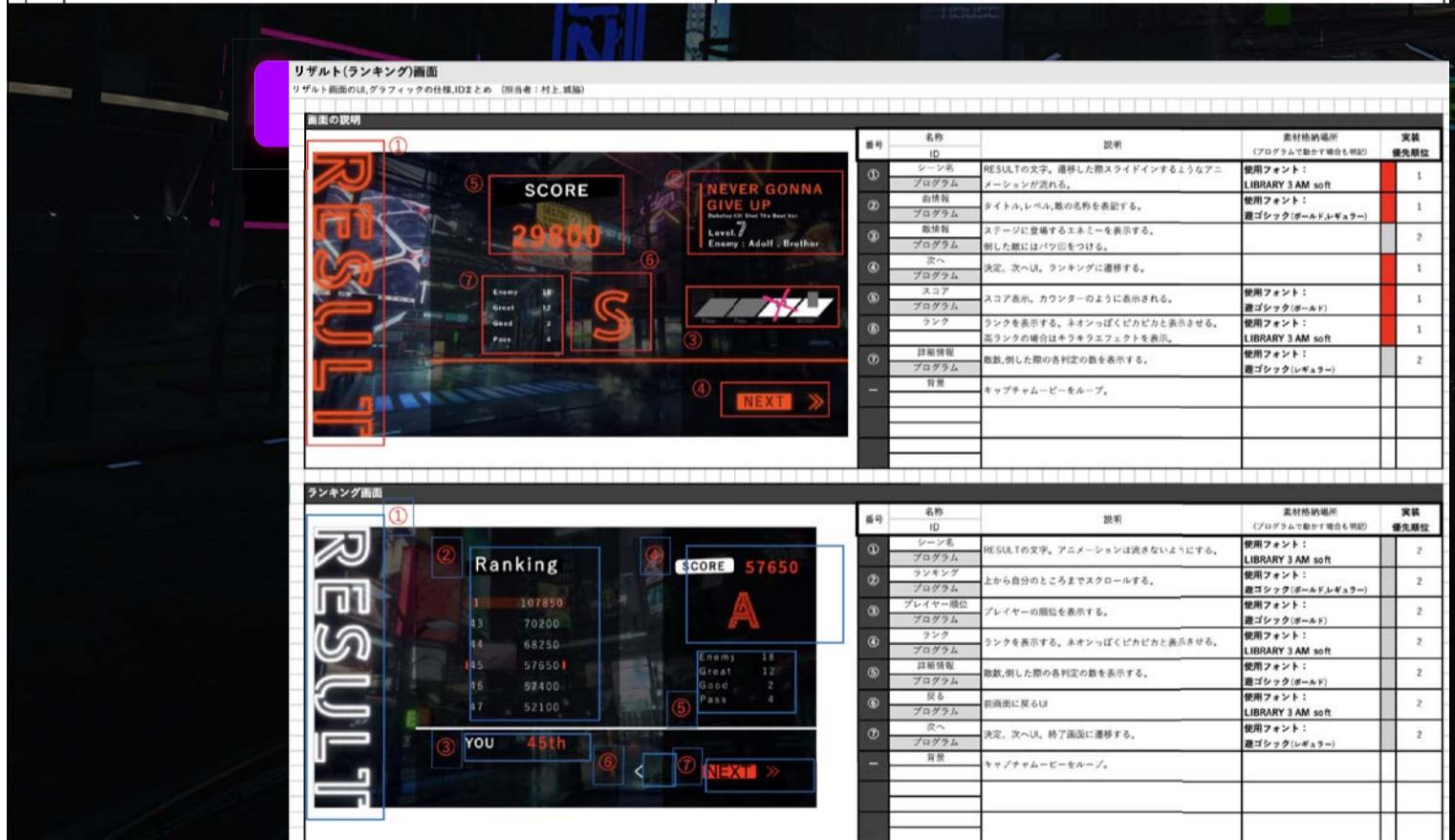
メインデザイナーの補助として、UI デザインの諸所で制作に関わりました。  
オプション画面 UI やローディング中に表示される Tips のグラフィック制作、  
各シーンのバランス調整やフォントの選定などを行いました。

DESIGNER



**アニメーションが必要な部分**

必要箇所	備考(アニメーションのイメージ図,詳細説明)
① シーン名	遷移した際、上からスライドインするようなアニメーションを入れる。
⑥ ジャケット選択移動時のアニメーション	斜め上下に動く。ジャケット横の矢印型UIがやや外側に動いて戻る。
- 背景	3Dメッシュマップが表示される。



## グラフィック仕様書

城脇 礼奈

デザイナーが少ないチームなので、早期に仕様の整理やタスクの洗い出しが必要だと考え、メインデザイナーと相談しながら実装可能なものを取捨選択し優先順位を書き出してまとめました。各工程の進捗管理や仕様の確認に役立てたかなと思います。

PROGRAMMER  
盛田 翔太



```
23 個の参照
public static ActionManager instance [ private set; set; ] = null;
2 個の参照
public bool IsSleen [ get; private set; ]
[Serializable]
private GameObject MenuOBJ;
private ReactiveProperty<float> op_BGM = new ReactiveProperty<float>();
4 個の参照
public ReadOnlyReactiveProperty<float> OP_BGM => op_BGM.ToReadOnlyReactiveProperty();

private ReactiveProperty<float> op_SE = new ReactiveProperty<float>();
3 個の参照
public ReadOnlyReactiveProperty<float> OP_SE => op_SE.ToReadOnlyReactiveProperty();

private ReactiveProperty<float> op_NotesSpeed = new ReactiveProperty<float>();
1 個の参照
public ReadOnlyReactiveProperty<float> OP_NotesSpeed => op_NotesSpeed.ToReadOnlyReactiveProperty();

private ReactiveProperty<float> op_Timing = new ReactiveProperty<float>();
1 個の参照
public ReadOnlyReactiveProperty<float> OP_Timing => op_Timing.ToReadOnlyReactiveProperty();

private ReactiveProperty<float> op_Mouse = new ReactiveProperty<float>();
2 個の参照
public ReadOnlyReactiveProperty<float> OP_Mouse => op_Mouse.ToReadOnlyReactiveProperty();
```

PROGRAMMER  
菅原 涉



PROGRAMMER  
山川 恵斗



```
KILL SHOT CYBER VEIL
TT GASH CRASH D-ELF
Savior of the
Cyber Space
Trance-vocal by D'eff
LEVEL. 1
ENEMY. KSB-x2040
KSB-x2040
WE D-ELF
```

## PROGRAMMER

小林 勇樹



```
[SerializeField] private TextAsset textAsset;
[System.Serializable]
5個の参照
public class GunDamageClass
{
    public string color;
    public int Legdamage;
    public int Bodydamage;
    public int Headdamage;
}

public GunDamageClass[] gunDamage;

@Unity メッセージ10 個の参照
public void Start()
{
    ReadCSV();
}

1 個の参照
public void ReadCSV()
{
    string[] data = textAsset.text.Split(new string[] { "\r", "\n" }, System.StringSplitOptions.None); //カンマ区切りにしてdataに代入
    int tableSize = data.Length / 4 - 1; //dataのlengthを4-1をして範囲指定
    gunDamage = new GunDamageClass[tableSize]; //gunDamageにサイズをぶち込む
    for (int i = 0; i < tableSize; i++)
    {
        gunDamage[i] = new GunDamageClass();
        gunDamage[i].color = data[4 * (i + 1)];
        gunDamage[i].Legdamage = int.Parse(data[4 * (i + 1) + 1]);
        gunDamage[i].Bodydamage = int.Parse(data[4 * (i + 1) + 2]);
        gunDamage[i].Headdamage = int.Parse(data[4 * (i + 1) + 3]);
    }
}
```

Gun Damage	
=▼ White	Color White
	Legdamage 30
	Bodydamage 45
	Headdamage 80
=▼ Blue	Color Blue
	Legdamage 45
	Bodydamage 60
	Headdamage 100
=▼ Purple	Color Purple
	Legdamage 60
	Bodydamage 100
	Headdamage 100
=▼ Red	Color Red
	Legdamage 100
	Bodydamage 100
	Headdamage 100

## PROGRAMMER

大長 ルミナ

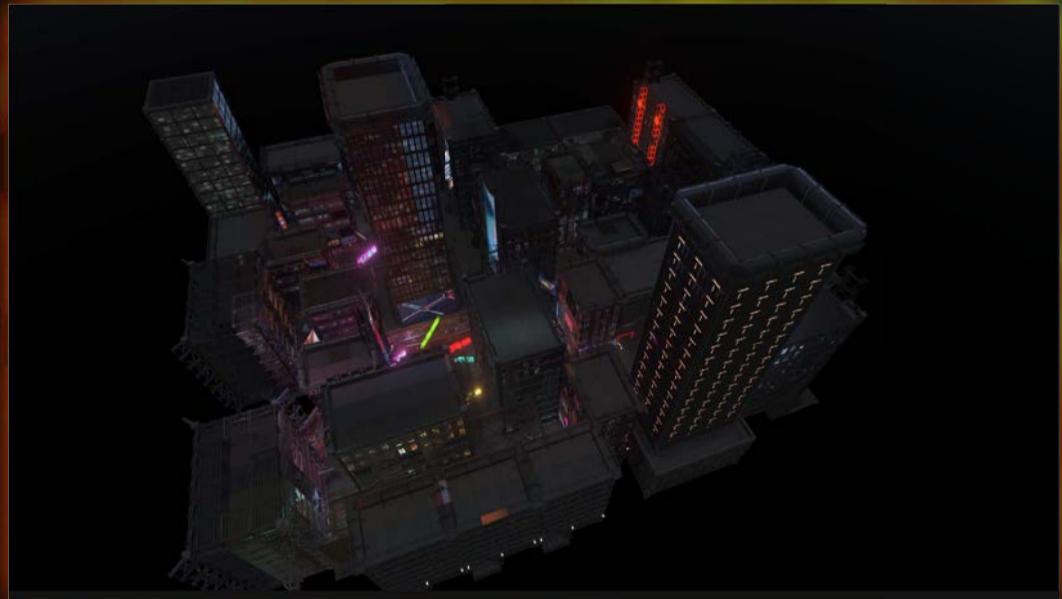


```
/// AIの行動を制御する ...
@I usage ▲ MS School +1
public void situationAI(Situation situation)
{
    //死んだ後に状態が変化した場合の予防処理
    if(gameObject.activeSelf) return;
    switch (situation)
    {
        case Situation.Idle://待機
            NavMeshAgent.speed = 0;
            Animator.SetInt(cullentAnimStatus,value:(int)MoveType.Idle);
            StartCoroutine(routine: Ai.Idle());
            break;
        case Situation.Return://元の位置に移動
            StartCoroutine(routine: Ai.Return());
            break;
        case Situation.LockOn://追尾
            StartCoroutine(routine: Ai.LockOn());
            break;
        case Situation.Roaming://徘徊
            StartCoroutine(routine: Ai.Roaming());
            break;
        case Situation.Attack://攻撃
            StartCoroutine(routine: Ai.Attack());
            break;
        case Situation.KnockBack:
            StartCoroutine(routine: Ai.KnockBack());
            break;
        case Situation.Die:
            StartCoroutine(routine: Ai.Die());
            break;
    }
}
```

Enemy Status List	
=▼ Robot	Robot
Name	Robot
Prefab	Robot
Range	Short
Hp	100
Attack Power	1500
Cool Time	2
Speed	50
Score	3000
=▼ Drone	Drone
Name	Drone
Prefab	None (Game Object)
Range	Long
Hp	70
Attack Power	1000
Cool Time	7
Speed	50
Score	2000
=▼ New Enemy	New Enemy
Name	New Enemy
Prefab	None (Game Object)
Range	Long
Hp	0
Attack Power	0
Cool Time	0
Speed	0
Score	0

## PROGRAMMER

磯部 功太



GameManager/ リザルトシーン /  
オプション画面 / インスペクタ拡張 / クラス拡張 /

盛田 翔太



Unity スクリプト 135 個の参照

```
public class GameManager : MonoBehaviour
{
    [SerializeField]
    private List<TitleSelect> canPauseScene = new List<TitleSelect>();

    36 個の参照
    public static GameManager instance { private set; get; }

    6 個の参照
    public bool IsPause { get; private set; } = false;

    public int Score = 0;

    4 個の参照
    public int KillEnemy { get; private set; } = 0;

    4 個の参照
    public int HeadHit { get; private set; } = 0;

    4 個の参照
    public int BodyHit { get; private set; } = 0;

    4 個の参照
    public int LegHit { get; private set; } = 0;

    13 個の参照
    public SongData Song { get; private set; }

    ④ Unity メッセージ 10 個の参照
    private void Awake()
    {
        if (!instance)
        {
            instance = this;
            DontDestroyOnLoad(this.gameObject);
        }
        else
        {
            Destroy(this.gameObject);
        }
    }

    ④ Unity メッセージ 10 個の参照
    private void Start()
    {
        //Musicセレクトシーンに移行したら、リザルトのステータス全部リセット
        LoadSceneManager.instance.CurrentScene.Where(i => i == TitleSelect.Music_Select).Subscribe(x => ResetResult()).AddTo(this);
        //MainMapに移行したら、カーソル非表示
        LoadSceneManager.instance.CurrentScene.Where(i => i == TitleSelect.MainMap).Subscribe(x => ViewCursor(false)).AddTo(this);
    }

    public void Pause()
    {
        StartCoroutine(_Pause());
    }

    1 個の参照
    private IEnumerator _Pause()
    {
        if (IsPause)
        {
            //再生
            IsPause = false;
            Time.timeScale = 1;
            yield return null;
            OptionManager.instance.ClauseMenu();
            PlayMusic_Player.instance.Set_BGM_Volume();
            PlayMusic_Player.instance.Resume_PlayMusic_Player();
            ViewCursor(false);
        }
        else
        {
            //ポーズ
            IsPause = true;
            Time.timeScale = 0;
            yield return null;
            PlayMusic_Player.instance.Pause_PlayMusic_Player();
            OptionManager.instance.OpenMenu();
            ViewCursor(true);
        }
    }
}
```

## GameManager

GameManager では、ほぼほぼデータを保持することしかやっていません。  
当時ポーズ処理を追加したいのですが、新しくスクリプトを用意するのも...?  
と思い GamaManager に実装しました。リザルトで使うデータなどを初期化  
する処理も、ミュージックセレクト画面に移行したら初期化するようになって  
います。ここでは Unixx を使用しイベントを登録しています。



## SCORE

Enemy	0
Head	0
Body	0
Leg	0

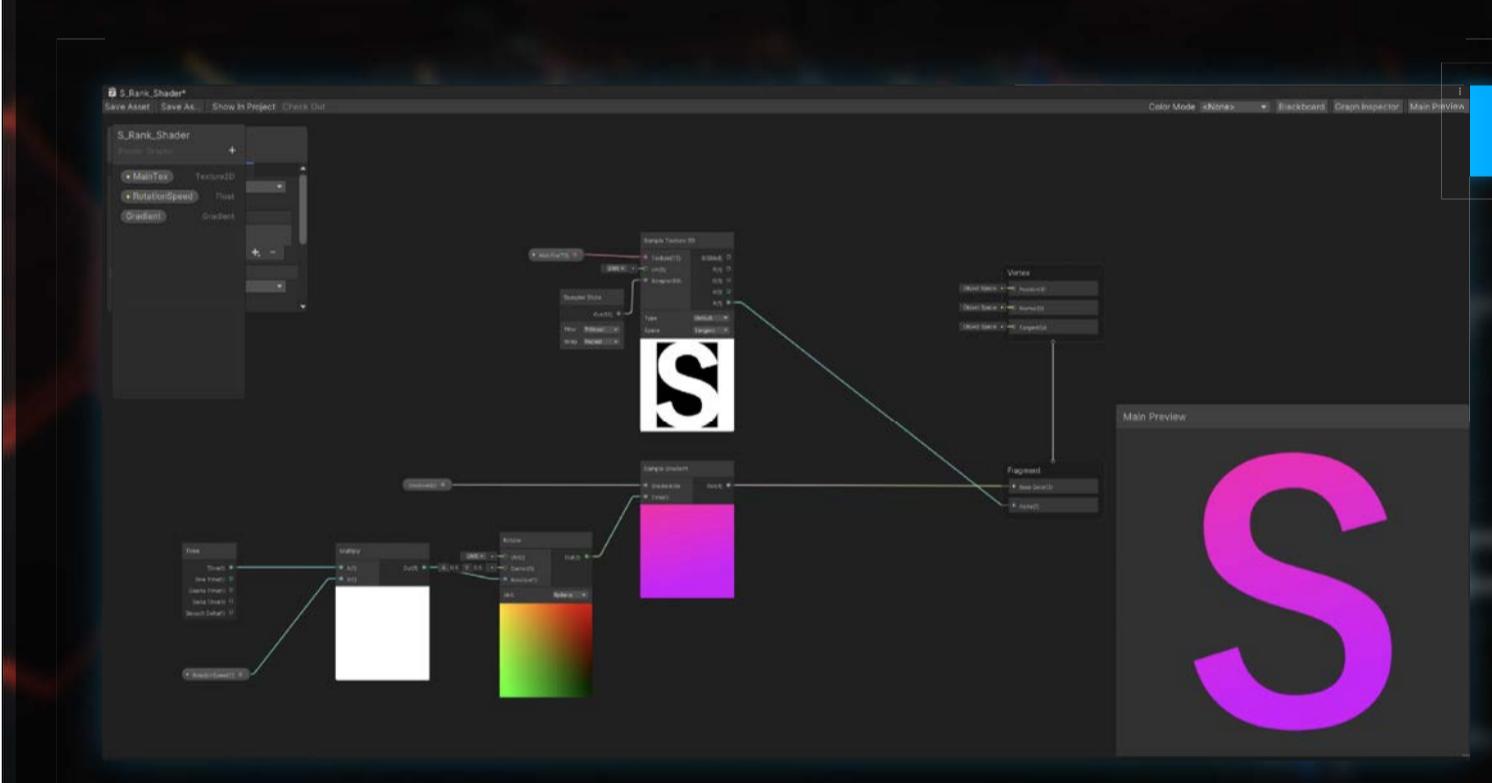
WE ARE ONE

Dubstep Kill Shot The Beat Ver. by D'elf

LEVEL 5

ENEMY, KSB-x2040, KSB-mini3

**NEXT** >>>



数値の部分は `ResultManager` クラスが変更をしているだけなので、特に目立った部分はありません。Shader Graph で書いた部分を紹介します。デザイナーから「中の色の部分を 2 色のグラデーションで回転してほしい」と要望をいただきました。処理としてはすごくコンパクトにできたと思います。

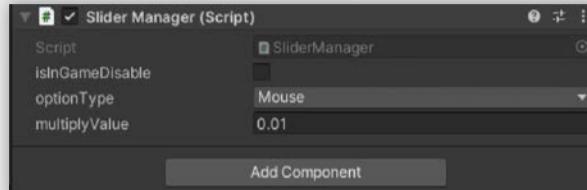
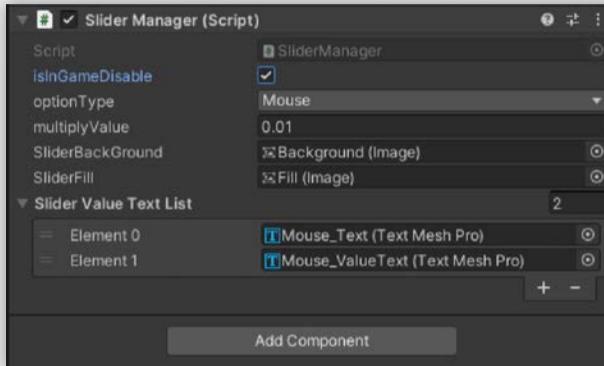
## リザルトシーン

PROGRAMMER



## オプション画面

Option 画面では、各パラメータを調整できるように、Slider で実装しました。ゲーム中はタイミングを変更できないように設定されています。左の文字は、曲のイメージカラーに合わせて色が変わり、値は GameManager から取得できます。パラメータの変数はすべて Unirx の ReactiveProperty で定義し、変更がかかった場合サウンド関係にイベント発行するようになっています。



```
[CustomEditor(typeof(SliderManager))]
@Unity_スクリプト10 個の参照
public class Cl_SliderManager : Editor
{
    0 個の参照
    public override void OnInspectorGUI()
    {
        EditorGUILayout.BeginDisabledGroup(true);
        EditorGUILayout.ObjectField("Script", MonoScript.FromMonoBehaviour((MonoBehaviour)target), typeof(MonoScript), false);
        EditorGUILayout.EndDisabledGroup();
        var slider = target as SliderManager;
        slider.isInGameDisable = EditorGUILayout.Toggle("isInGameDisable", slider.isInGameDisable);
        slider.optionType = (OptionType)EditorGUILayout.EnumPopup("optionType", slider.optionType);
        serializedObject.FindProperty("multiplyValue").floatValue = EditorGUILayout.FloatField("multiplyValue", serializedObject.FindProperty("multiplyValue").floatValue);
        if (slider.isInGameDisable)
        {
            slider.SliderBackGround = (Image)EditorGUILayout.ObjectField("SliderBackGround", slider.SliderBackGround, typeof(Image), true);
            slider.Sliderfill = (Image)EditorGUILayout.ObjectField("Sliderfill", slider.Sliderfill, typeof(Image), true);
            //SliderValueTextList の表示部分
            EditorGUILayout.PropertyField(serializedObject.FindProperty("SliderValueTextList")); //paramsでoptionを書くことも可能
        }
        serializedObject.ApplyModifiedProperties();
    }
}
```

## インスペクタ拡張

スイッチ一つの ON/OFF で要らない設定を消したり、表示させたりすることができます。将来的にはもう少し勉強を重ね、便利機能をインスペクタ上に実装出来たらいいなと考えています。

```
public static T MS_FadeOut<T>(this T t, float time, float endValue, UnityAction action) where T : Image
{
    IsFade = true;
    if (t == null)
    [
        return t;
    ]
    StaticMono.StartCoroutine(FadeOut(t, time, endValue, action));
    return t;
}
private static IEnumerator FadeIn(Image image, float time, float endValue, UnityAction action)
{
    if (time > 0)
    [
        var c = image.color;
        c.a = 0;
        while (image.color.a != endValue)
        [
            c.a += Time.deltaTime / time;
            if (c.a > endValue)
            [
                c.a = endValue;
            ]
            image.color = c;
            yield return null;
        ]
        IsFade = false;
        action.Invoke();
    ]
    else
    [
        Debug.LogError("FadeInではtimeを0以上の値にしてください");
        var c = image.color;
        c.a = endValue;
        image.color = c;
        yield return null;
    ]
}
```

## クラス拡張

便利機能 ( フェイド機能 ) を自作で作ってみたかったので入れてみました。使っていて問題はないのですが、現在がフェイドかどうかは不明なのと、外からは void 関数で呼んでいるため、コルーチンで待つことができないところが不便でした。結果としてはあまり使っていないですがクラスに機能追加できた点では大成功です！今後はこの便利機能をより良くしていきたいと思っています。

## CSV から数値を読み込み List 化 / プレイヤーの移動 / プレイヤーステータスのフラグ管理 /

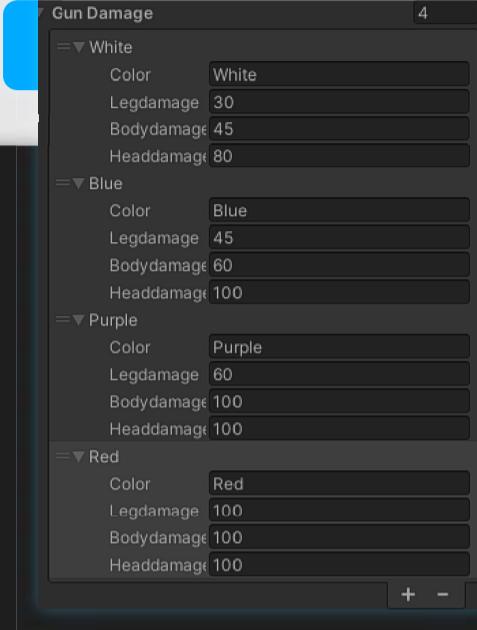
小林 勇樹

```
[SerializeField] private TextAsset textAsset;
[System.Serializable]
5 個の参照
public class GunDamageClass
{
    public string color;
    public int Legdamage;
    public int Bodydamage;
    public int Headdamage;
}

public GunDamageClass[] gunDamage;

❷Unity メッセージ10 個の参照
public void Start()
{
    ReadCSV();
}

1 個の参照
public void ReadCSV()
{
    string[] data = textAsset.text.Split(new string[] [",", "\n"], System.StringSplitOptions.None); //カンマ区切りにしてdataに代入
    int tableSize = data.Length / 4 - 1; //dataのlengthを4-1をして範囲指定
    gunDamage = new GunDamageClass[tableSize]; //gundamageにサイズをぶち込む
    for (int i = 0; i < tableSize; i++)
    {
        gunDamage[i] = new GunDamageClass();
        gunDamage[i].color = data[4 * (i + 1)];
        gunDamage[i].Legdamage = int.Parse(data[4 * (i + 1) + 1]);
        gunDamage[i].Bodydamage = int.Parse(data[4 * (i + 1) + 2]);
        gunDamage[i].Headdamage = int.Parse(data[4 * (i + 1) + 3]);
    }
}
```



### CSV から数値を 読み込み List 化

プランナーが数値を変えやすいように CSV ファイルから List にして攻撃力を変動させました。

```
if (Input.GetAxisRaw("Horizontal") != 0 || Input.GetAxisRaw("Vertical") != 0)
{
    AddWalkFlag();
}
else
{
    playerScript._rg.velocity = new Vector3(0, playerScript._rg.velocity.y, 0);
}

// カメラの方向から、X-Z平面の単位ベクトルを取得
Vector3 cameraForward = Vector3.Scale(Camera.main.transform.forward, new Vector3(1, 0, 1)).normalized;

// 方向キーの入力値とカメラの向きから、移動方向を決定
Vector3 moveForward = cameraForward * inputVertical + Camera.main.transform.right * inputHorizontal;

// 移動方向にスピードを掛ける。ジャンプや落下がある場合は、別途Y軸方向の速度ベクトルを足す。
_rg.velocity = moveForward.normalized * _currentSpeed + new Vector3(0, _rg.velocity.y, 0);
```

### プレイヤーの移動

関数に float の引数を付けることでコードをきれいに書く意識をしました。  
カメラの forward によって、w で前に移動するようにしました。

```
1 個の参照
public void AddWalkFlag()
{
    //今現在のステータス と walk の 0001 のステータスが or で掛けてフラグを立てる
    //どちらかが1立っていたら1を立てる
    //0000 | 0001 = 0001
    //左から一個づつ比較している
    _playerStatus |= PlayerStatus.Walk; //フラグの追加
}

1 個の参照
public void DelWalkFlag()
{
    //今現在のステータスを walk の 1110 & をかけてフラグを消してる
    //ANDは両方に1が立ったら結果に1が立つ
    //0001 & 1111 = 1111
    _playerStatus &= ~PlayerStatus.Walk;
}

1 個の参照
public void AddJumpFlag()
{
    _playerStatus |= PlayerStatus.Jump;
}

2 個の参照
public void DelJumpFlag()
{
    _playerStatus &= ~PlayerStatus.Jump;
}
```

```
#region Flag
[Flags]
10 個の参照
enum PlayerStatus
{
    Idle = 0x000,    //待ち状態
    Walk = 0x001,    //歩く
    Jump = 0x002,    //ジャンプ
    Attack = 0x004,  //アタック
}
#endregion
```

## プレイヤーステータスの フラグ管理

プレイヤーのフラグを 16 進数で管理してフラグの add や del をすることで、複数のステータスを管理できるようになりました。



CRIWARE / ノーツのオブジェクトプール処理 /  
ノーツ判定スクリプト /

菅原 渉



```
//TapNoteを貸し出す処理
2 個の参照
public NotesScript Launch_Tap()
{
    //QueueからNotesを一つ取り出す
    NotesScript tmpBullet = tapNotesQueue.Dequeue();
    //弾を表示する
    tmpBullet.SetNotes_InitialValue();
    tmpBullet.NoteFadeIn();
    //呼び出し元に渡す
    return tmpBullet;
}

//TapNoteの回収処理
3 個の参照
public void Collect_TapNote(NotesScript _bullet)
{
    //Queueに格納
    tapNotesQueue.Enqueue(_bullet);
}
```

## ノーツの オブジェクトプール処理

今回のゲームはメインゲーム部分が重くなる可能性があったので、ノーツの部分でも出来るだけ軽くできるように、オブジェクトプールを利用しました。

PROGRAMMER

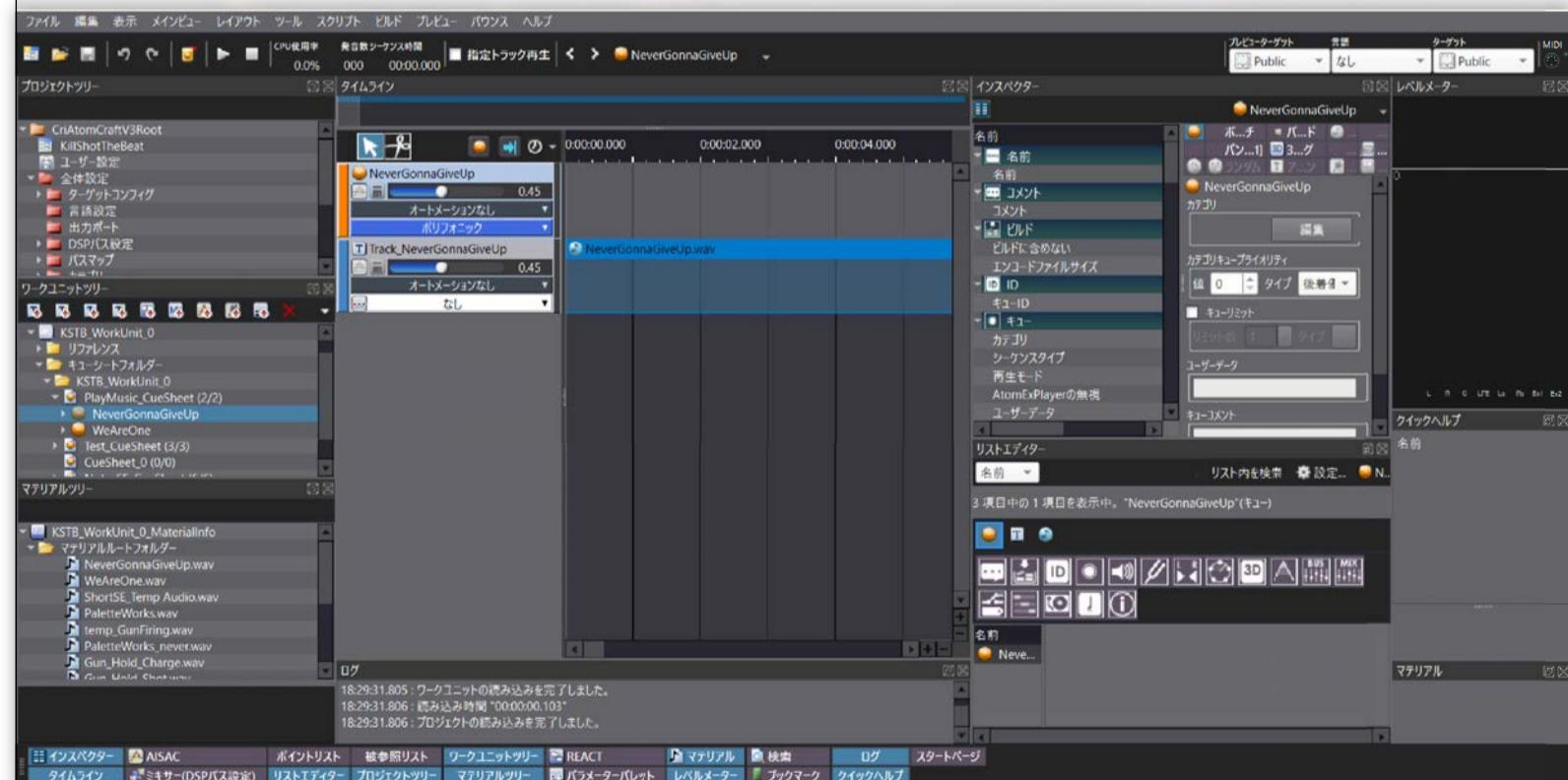
```

136
137     /// <summary>
138     /// 判定するNoteのTypeをチェックして、適切な判定を読み込む
139     /// </summary>
140     /// <returns></returns>
141     3 個の参照
142     public NoteJugeIndex NowNotes_Judge()
143     {
144         // 戻り値用に宣言
145         NoteJugeIndex jugeIndex = new NoteJugeIndex();
146
147         if (noteDataLists.Count > noteJudgeCount)
148         {
149             // TapNoteだったら
150             if ((int)NoteType.Tap == ReturnNoteType())
151             {
152                 // 出た判定を戻り値で返す
153                 return TapNoteJudge(jugeIndex);
154             }
155             // それ以外 (HoldNote) だったら
156             else
157             {
158                 // 出た判定を戻り値で返す
159                 return HoldNoteJudge(jugeIndex);
160             }
161         }
162         return null;
163     }

```

## ノーツ 判定スクリプト

プレイヤーがクリックしたときに読み込まれるノーツ判定のスクリプトです。  
ノーツの種類で判定の仕方が異なるため、まず初めに種類の検知をしています。



**CRIWARE**

BGM・SE の管理で、便利かつ UnityEngine聞いたため、使用いたしました。

PROGRAMMER

## エネミーのAI / オブジェクトプール / スクリプタブルオブジェクト /

大長 ルミナ



```
/// AIの挙動を制御する ...
[1 usage 2 MS School +1]
public void SituationAI(Situation situation)
{
    //死んだ後に状態が変化した場合の予防処理
    if(!gameObject.activeSelf) return;
    switch (situation)
    {
        case Situation.Idle://待機
            NavMeshAgent.speed = 0;
            Animator.SetInteger(id:CullentAnimStatus, value:(int)MoveType.Idle);
            StartCoroutine(routine: Ai.Idle());
            break;
        case Situation.Return://元の位置に移動
            StartCoroutine(routine: Ai.Return());
            break;
        case Situation.LockOn://追尾
            StartCoroutine(routine: Ai.LockOn());
            break;
        case Situation.Roaming://徘徊
            StartCoroutine(routine: Ai.Roaming());
            break;
        case Situation.Attack://攻撃
            StartCoroutine(routine: Ai.Attack());
            break;
        case Situation.KnockBack:
            StartCoroutine(routine: Ai.KnockBack());
            break;
        case Situation.Die:
            StartCoroutine(routine: Ai.Die());
            break;
    }
}
```

```
public ReactiveProperty<Situation> CullentSituation = new ReactiveProperty<Situation>(); Serializable
public readonly int CullentAnimStatus = Animator.StringToHash(name: "NowStatus");
```

**エネミーのAI**

今回初めてのAI挙動にチャレンジしました。

AIが自身で考えてから次の行動に移すようにランダム性を高めて作成しました。  
またUniRxを新たに学びイベント登録することで無駄な処理を減らし、  
複数居ても軽く動くように気を配りながら作りました。

```

/// 使われていない敵をステージに配置する関数 ...
⌚ Frequently called ⌚ 1 usage 2 RMN0605
public GameObject OrderEnemy(Vector3 summonArea)
{
    bool isOrderedFound = false;
    GameObject summonEnemy = null;

    // プレハブ探しの旅スタート
    foreach (var poolClass : value.index) in EnemyPoolList.Select((value: PoolClass, index: int) => new { value, index })
    {
        // 使われていないプレハブを見つけ方はこちらへ
        if (!poolClass.value.IsUsedPrefab)
        {
            summonEnemy = poolClass.value.Prefab;
            poolClass.value.IsUsedPrefab = true;
            isOrderedFound = true;
            summonEnemy.GetComponent<Enemy>().MyPoolNum = poolClass.index;
            break;
        }
    }

    // プレハブを見つけられなかつの方はこちらへ
    if (!isOrderedFound)
    {
        summonEnemy = AreaSummonEnemies[0].gameObject;
        summonEnemy.transform.position = summonArea;
        AreaManager.Instance.AreaEnemyList.Add(summonEnemy);
    }
}

// 生成時にオブジェクトが一方向向いていると不自然なためランダムに回転させる
summonEnemy.transform.Rotate(xAngle: 0, yAngle: Random.Range(0, 360), zAngle: 0);
summonEnemy.transform.position = summonArea;
AreaManager.Instance.AreaEnemyList.Add(summonEnemy);

return summonEnemy;
}

```

## オブジェクトプール

生成時におこる過度な処理を軽減するため、オブジェクトプールを作成しました。  
直接的に関係するプログラムではないですが、これを使用するだけでプレイ中のエネミー生成に対する重さを大きく変えることができました。

```

public virtual void Awake()
{
    // 必要なデータを集める
    var DataBase : EnemyStatusInformation = Resources.Load<EnemyDataBase> (path: "EnemyDataBase").EnemyStatusList[(int)Id];

    // 代入
    MaxHp = DataBase.Hp;
    _currentHp = MaxHp;
    AttackPower = DataBase.AttackPower;
    _attackCoolTime = DataBase.CoolTime;
    _currentSpeed = DataBase.Speed;
    _getScore = DataBase.Score;
}

```

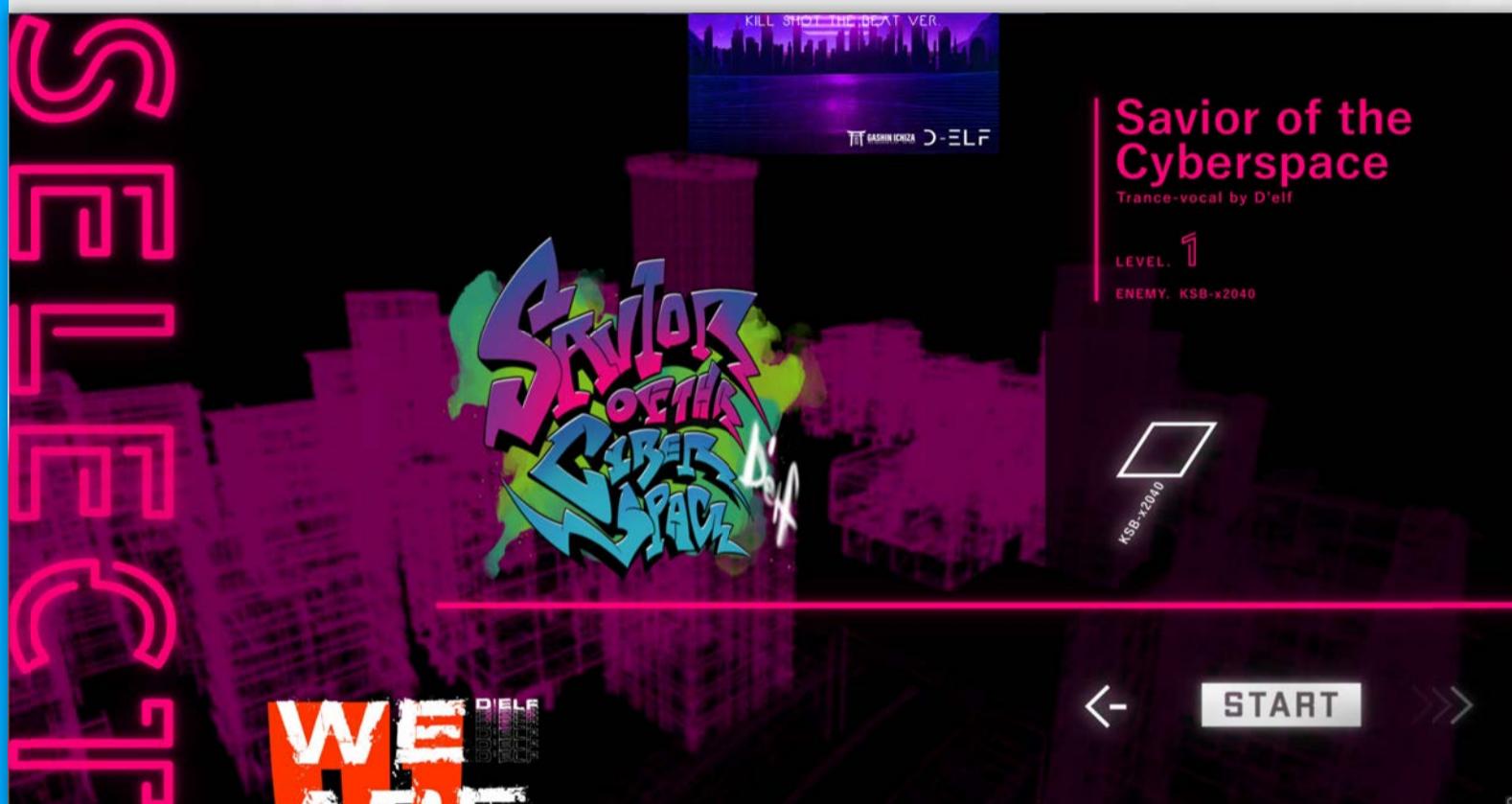
Enemy Status List	
Robot	Robot
Name	Robot
Prefab	Robot
Range	Short
Hp	100
Attack Power	1500
Cool Time	2
Speed	50
Score	3000
Drone	Drone
Name	Drone
Prefab	None (Game Object)
Range	Long
Hp	70
Attack Power	1000
Cool Time	7
Speed	50
Score	2000
New Enemy	New Enemy
Name	New Enemy
Prefab	None (Game Object)
Range	Long
Hp	0
Attack Power	0
Cool Time	0
Speed	0
Score	0

## スクリプタブルオブジェクト

エネミーの新規追加の際プランナー側でステータスを細かく  
いじれるように作りました。  
新たな敵を作る際は、モデルとステータスの情報を与えるだけで  
ステージで使えるようにしました。

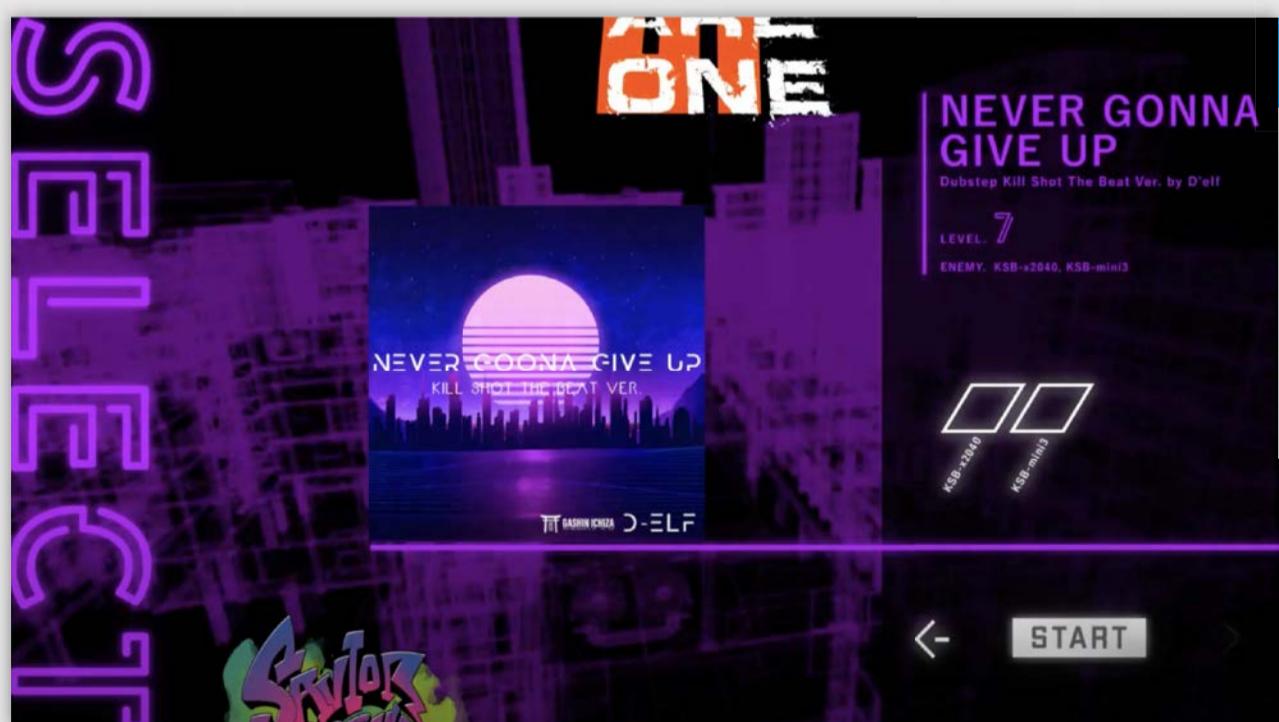
select 画面 / メイン画面 /  
曲管理のスクリプト /

山川 恵斗



select 画面

select 画面は見栄えも重視しつつ軽くすることに力を入れています。  
Google drive に動画も上がっています。



PROGRAMMER



## メイン画面

周りの effect で現在の DoseLevel の状況が分かるようにしています。  
また、HUD はポジションに対して遅れて追従するようになっています。

```
delayTime += Time.deltaTime;
delayTime1 += Time.deltaTime;
mousewheelPoint = Input.GetAxis("Mouse ScrollWheel");

if (Select_check == true) {
    if (mousewheelPoint < 0 && delayTime > 0.8f && delayTime1 > 1f && !scroll) //下方向入力時
    {
        scroll = true;
        delayTime = 0;
        if (SelectSongNum > 0)
        {
            SelectSongNum--;
            GameManager.instance.ChangeSongData(songData[SelectSongNum]);
            buttonKeep.ChangeHighLightColor();
            buckButtonKeep.ChangeHighLightColor();
            SelectScenePreview.instance.DoseSetAndPlay(SelectSongNum);
        }
        else
        {
            SelectSongNum = jacketSprites.Count - 1;
            GameManager.instance.ChangeSongData(songData[SelectSongNum]);
            buttonKeep.ChangeHighLightColor();
            buckButtonKeep.ChangeHighLightColor();
            SelectScenePreview.instance.DoseSetAndPlay(SelectSongNum);
        }
        UpSetJacket();
        loopImageCenterNum = (loopImageCenterNum + 1) % transforms.Count;
        MoveJacket();
        colorChange.Color();
    }
}
```

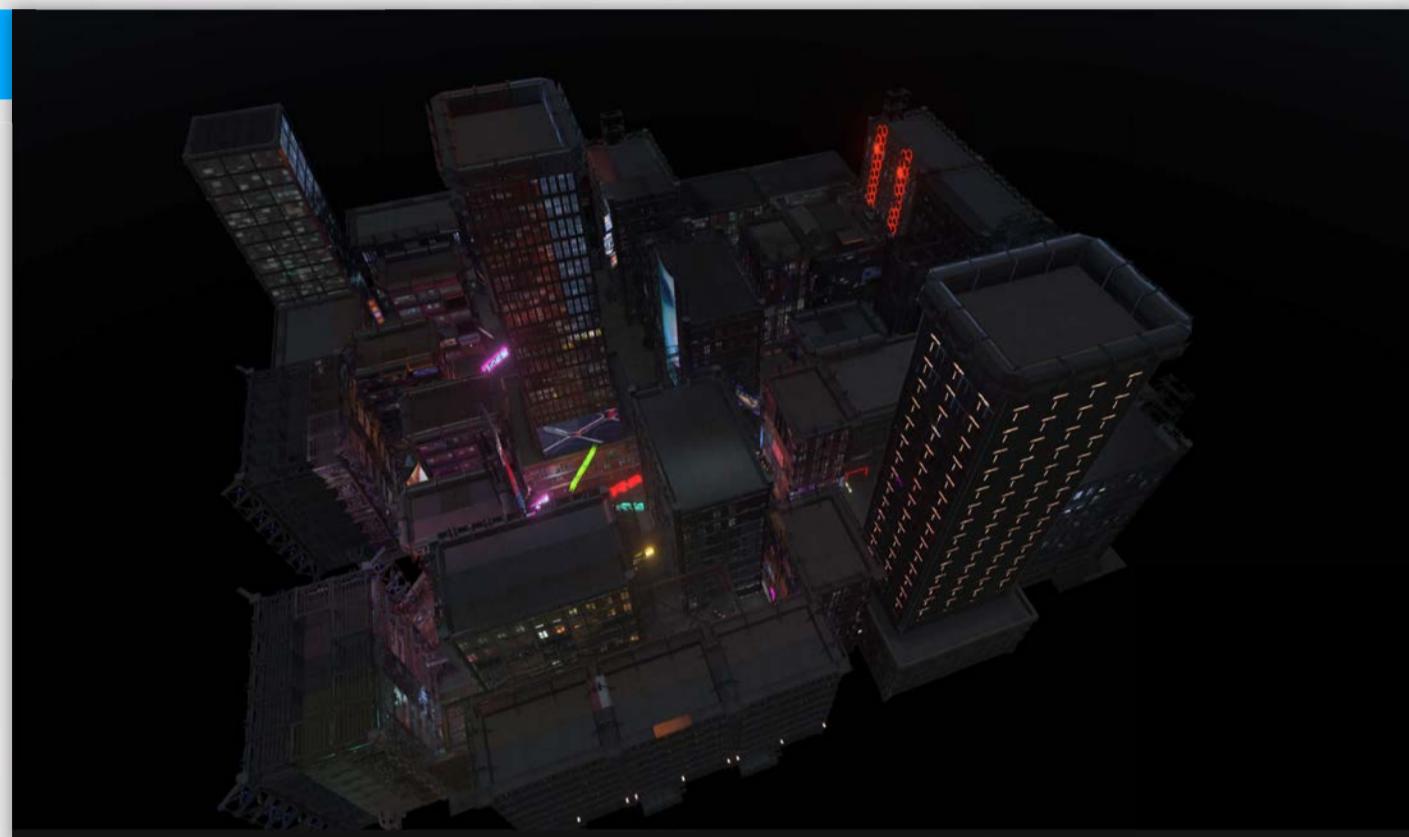
## 曲管理のスクリプト

曲の管理を行っている script になります  
楽曲の選択や UI 関係もこの番号をもらって動いています。

PROGRAMMER

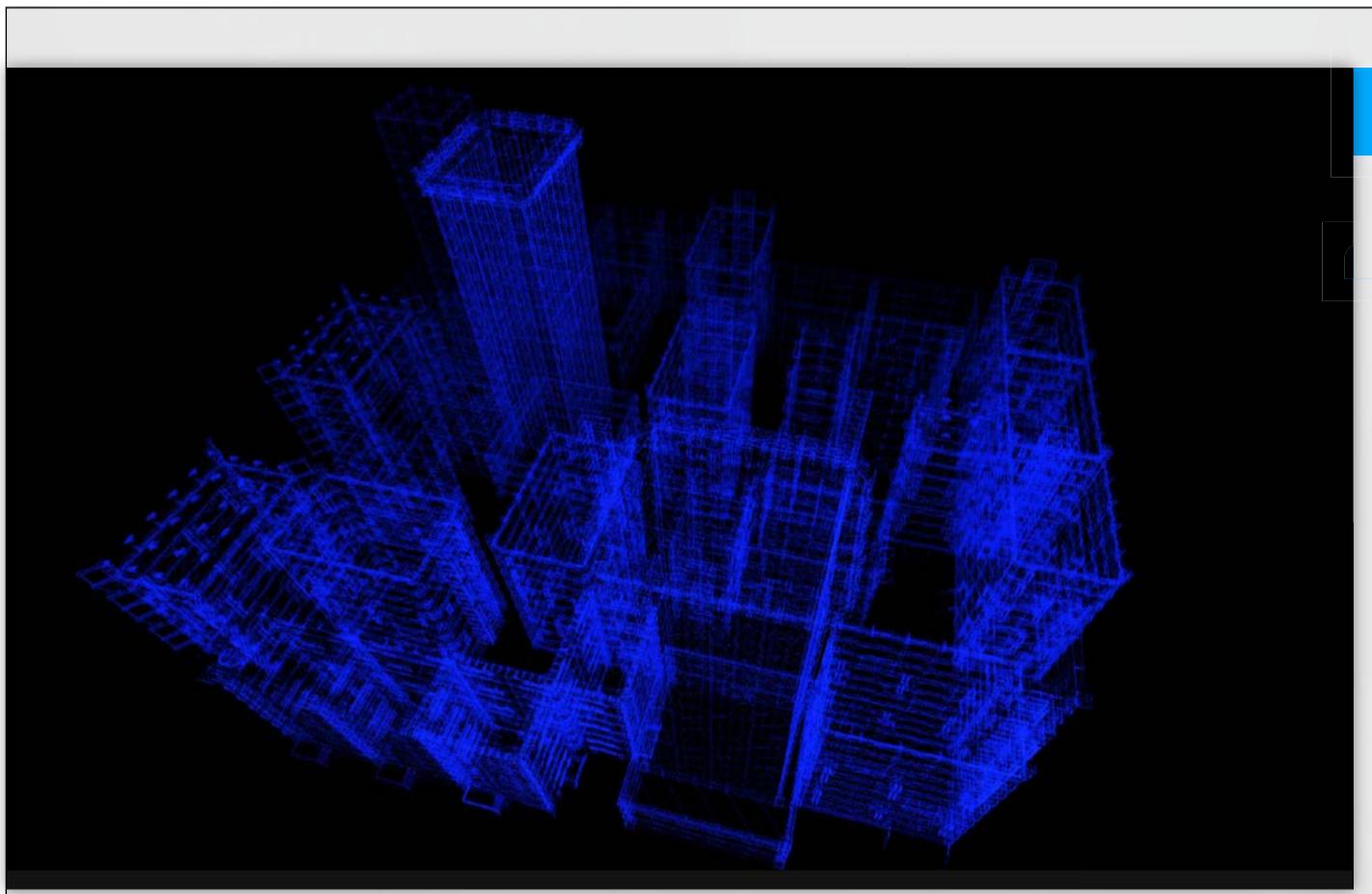
マップ制作 /  
マップのワイヤーフレーム化 /

磯部 功太



### マップ制作

マップの形を作ったり小物などを置くのに  
時間がかかりました。(約 30 時間)  
マップライトバイクなど(約 15 時間)は、  
マップのライトなどの設定に少し手こずり  
ました。



```
WireframeManager.cs
```

```
Assembly-CSharp
1 用 System.Collections;
2 用 System.Collections.Generic;
3 用 UnityEngine;
4 公共 enum Controller
5 {
6     Lines,
7     Points
8 }
9 //Unity スクリプト(0 個の参照
10 公共 class Wireframe : MonoBehaviour
11 {
12     [SerializeField]
13     Controller controller;
14     [SerializeField]
15     public Material material;
16     [SerializeField]
17     Material material2;
18     //Unity メッセージ(0 個の参照
19     void Start()
20     {
21         if (this.gameObject.GetComponent<Wireframe>() != null)
22             while (true)
23             {
24                 if (obj.gameObject.GetComponent<Wireframe>() != null)
25                 {
26                     Destroy(obj.gameObject.GetComponent<Wireframe>());
27                 }
28                 else
29                 {
30                     break;
31                 }
32             }
33     }
34 }
```

```
Wireframe.cs
```

```
Assembly-CSharp
1 用 System.Collections;
2 用 System.Collections.Generic;
3 用 UnityEngine;
4 公共 class WireframeManager : MonoBehaviour
5 {
6     [SerializeField]
7     Object[] allGameObject;
8     [SerializeField]
9     Material material;
10    //Unity メッセージ(0 個の参照
11    private void Start()
12    {
13        //ピエラルキー上のすべてのオブジェクトを取得
14        allGameObject = Resources.FindObjectsOfTypeAll(typeof(GameObject));
15        //取得したオブジェクトの名前を表示
16        foreach (GameObject obj in allGameObject)
17        {
18            if (obj.gameObject.GetComponent<Wireframe>() != null)
19                while (true)
20                {
21                    if (obj.gameObject.GetComponent<Wireframe>() != null)
22                    {
23                        Destroy(obj.gameObject.GetComponent<Wireframe>());
24                    }
25                    else
26                    {
27                        break;
28                    }
29                }
30        }
31    }
32 }
```

## マップの ワイヤーフレーム化

ワイヤーフレームマネージャ（約 7 時間）は、すべてのオブジェクトにスクリプトをつけてそこに対応させるのがとても大変でした。  
ワイヤーフレーム（4 時間）はマテリアルなどを使うのに時間がかかりました。

# KILL SHOT THE BEAT

キルショット・ザ・ビート

 GASHIN ICHIZA  
This impression to all. Est. 2022

MARKET SECTOR 36  
ID:ASDHJ#M