2020年度10月START Unity講座

12回目

講師：幸田 将伍 (@MagurodonDev)

今回の講義の目的

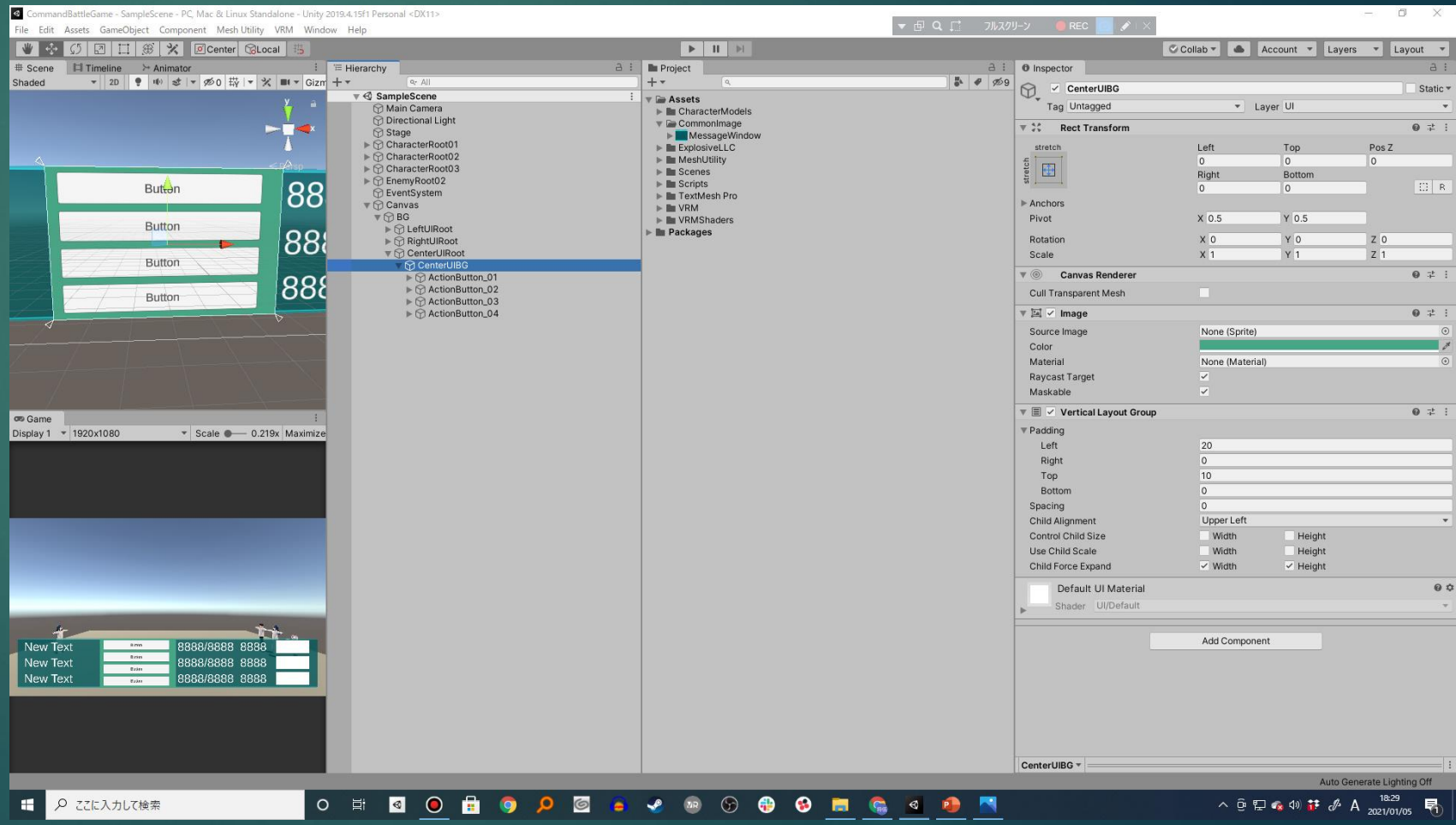
- プログラムを自分で読めるようになる
- Unityを使って自分が実現したいことをできるようになる
- 自分一人でもゲームを作成できるレベルになる
- Unityの活用事例を学び、自分の進路に役立てる
- 実際のエンジニアがどういった仕事の進め方をしているかを知る
- ゲーム会社のクライアントエンジニアとして就職できるレベルになる

一緒にレベルアップして行きましょう！

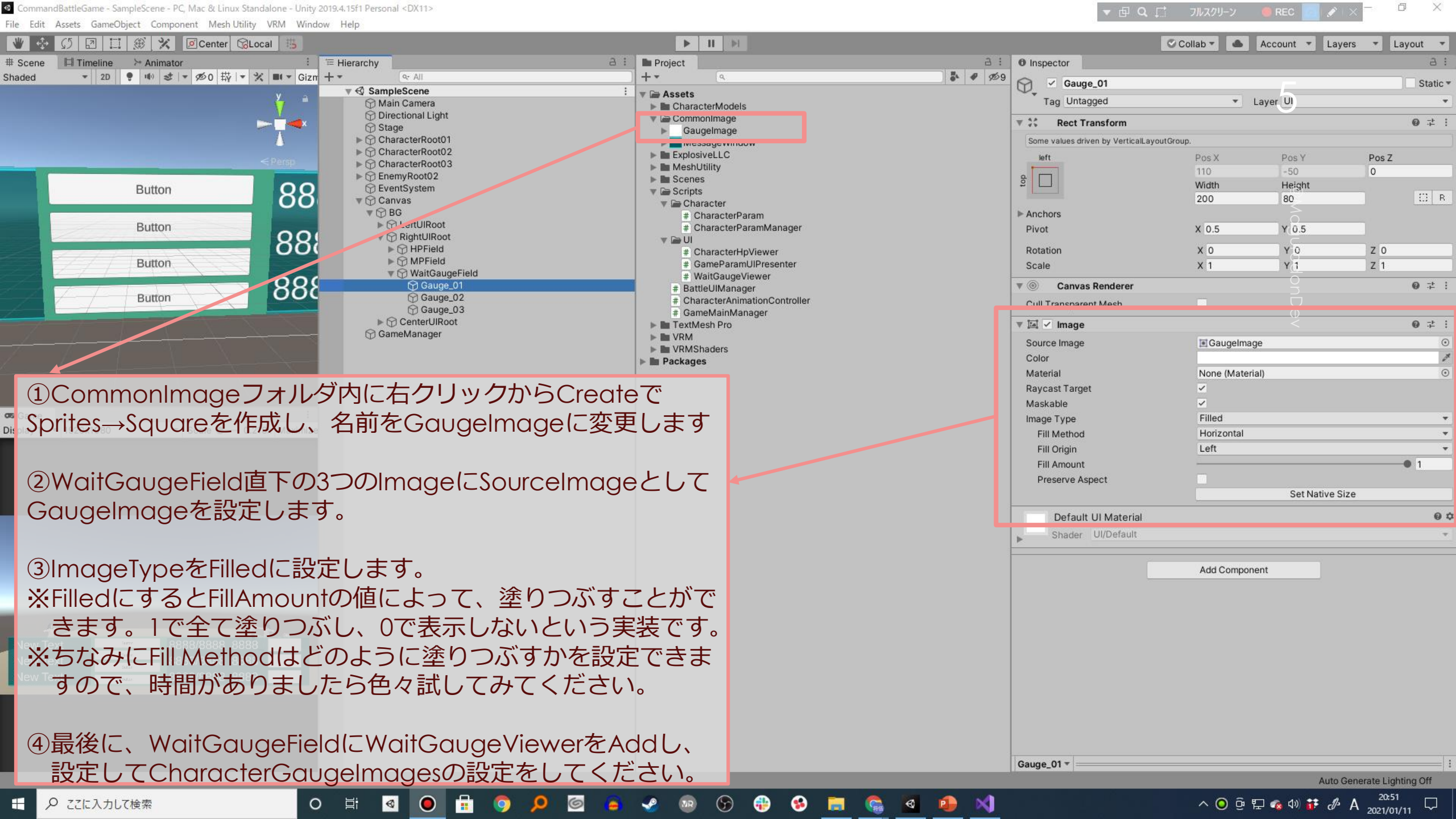
Unity
コマンドバトルゲーム

- 先週はUIの設定と、キャラクターのパラメーターの設定を行いました。

- 本日は、
- ①行動開始に使用するゲージの表現方法
- ②ゲージが溜まったら選択肢のパネルを表示する実装
- ③キャラクターの行動を決めるボタンによって、行動を起こす実装
- を行っていきます。
- Scripts/UIフォルダ直下にWaitGaugeViewer.csを作成しましょう。



```
using UnityEngine;
using UnityEngine.UI;

Unity スクリプト | 1 個の参照
public class WaitGaugeViewer : MonoBehaviour
{
    public Image[] CharacterGaugeImages = new Image[3];
    public float[] CharacterSpeeds = new float[3];

    0 個の参照
    public void Init()
    {
        for (int i = 0; i < 3; i++)
        {
            CharacterGaugeImages[i].fillAmount = 0;
        }
    }

    1 個の参照
    public void WaitGaugeUpDate()
    {
        for (int i = 0; i < 3; i++)
        {
            CharacterGaugeImages[i].fillAmount += CharacterSpeeds[i]/10000f;
        }
    }

    // Updateで計算をした後にSpeedの反映が入るようにLateUpdateを使用する
    Unity メッセージ | 0 個の参照
    private void LateUpdate()
    {
        WaitGaugeUpDate();
    }
}
```


①CommonImageフォルダ内に右クリックからCreateで
Sprites→Squareを作成し、名前をGaugeImageに変更します

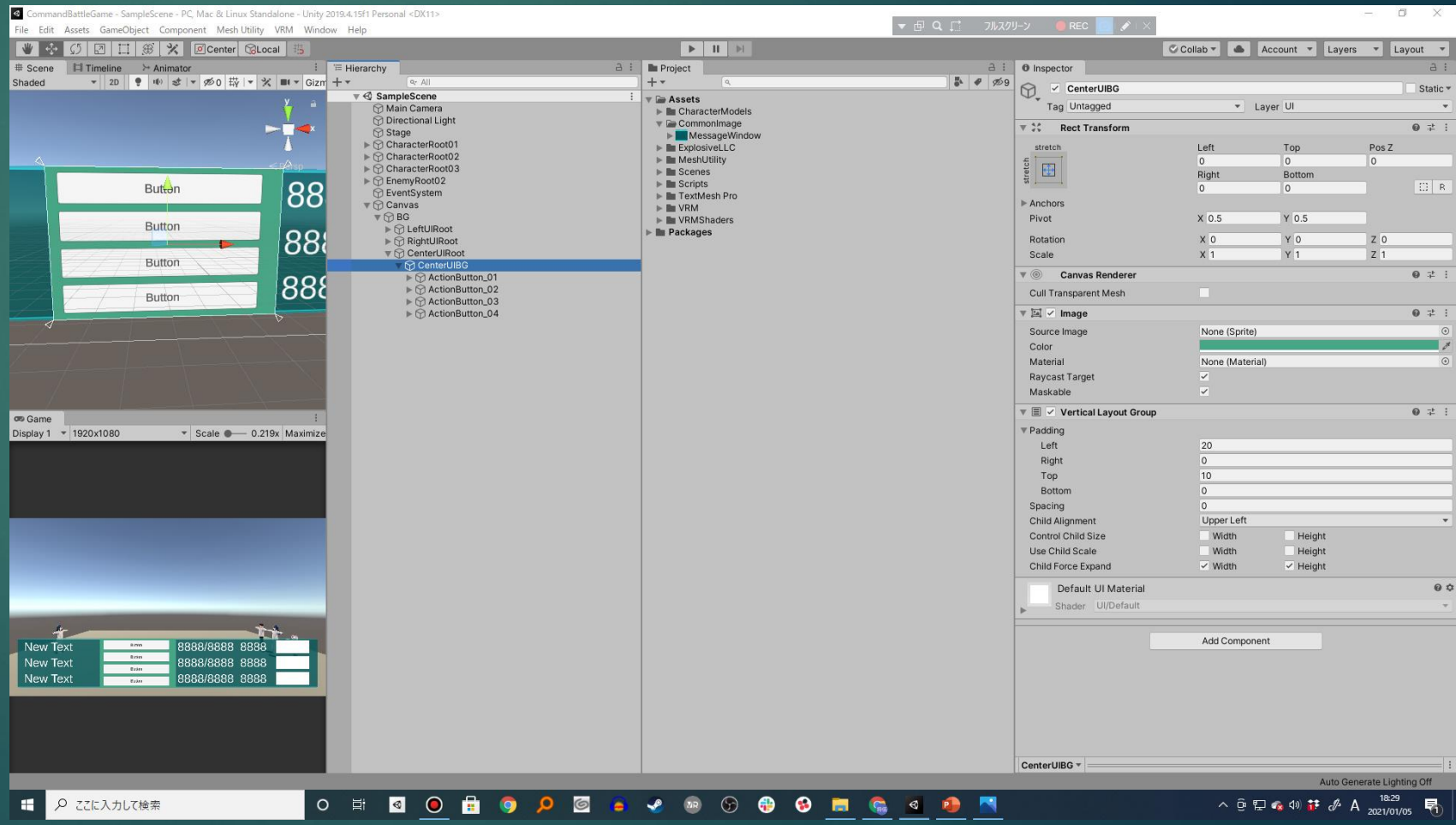
②WaitGaugeField直下の3つのImageにSourceImageとして
GaugeImageを設定します。

③ImageTypeをFilledに設定します。
※FilledにするとFillAmountの値によって、塗りつぶすことができます。1で全て塗りつぶし、0で表示しないという実装です。
※ちなみにFill Methodはどのように塗りつぶすかを設定できますので、時間がありましたら色々試してみてください。

④最後に、WaitGaugeFieldにWaitGaugeViewerをAddし、
設定してCharacterGaugeImagesの設定をしてください。

Unity
コマンドバトルゲーム

- ではキャラクターのSpeedを設定していきます。
- 試しにAttackerのSpeedを適当に編集して、120とかにしてみてください。
- GameParamUIPresenter.csの修正を行います。

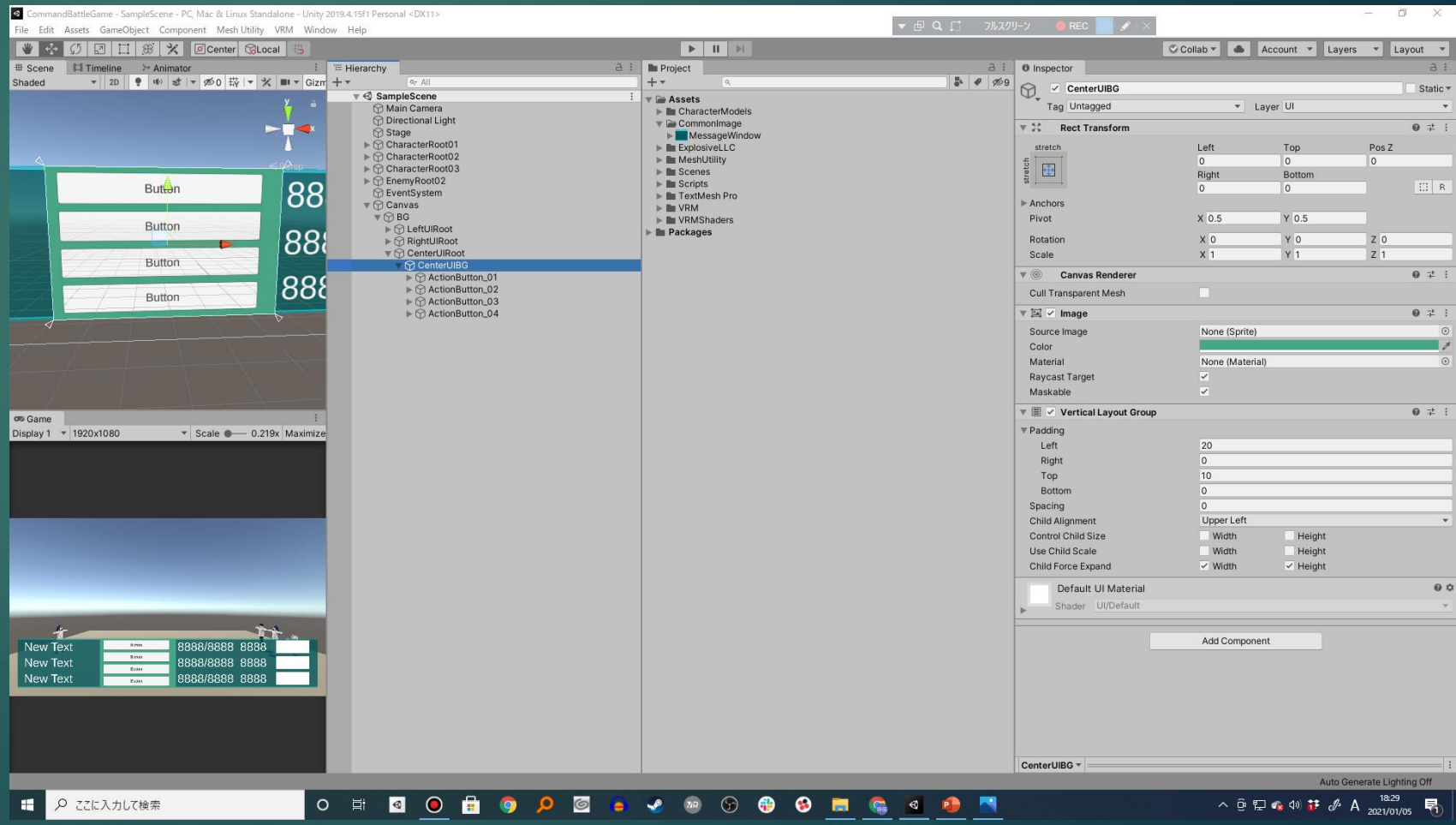


```
using System.Collections.Generic;
using UnityEngine;

Unity スクリプト|1 個の参照
public class GameParamUIPresenter : MonoBehaviour
{
    public CharacterHpViewer CharacterHpViewer = null;

    public WaitGaugeViewer WaitGaugeViewer = null;

    1 個の参照
    public void SetCharactersParamViewer(CharacterParam[] CharacterParams)
    {
        for (int i = 0; i < 3; i++)
        {
            if (CharacterParams[i] != null)
            {
                CharacterHpViewer.CharacterMaxHps[i] = CharacterParams[i].HitPoint;
                CharacterHpViewer.CharacterHps[i] = CharacterParams[i].HitPoint;

                WaitGaugeViewer.CharacterSpeeds[i] = CharacterParams[i].Speed;
            }
        }

        WaitGaugeViewer.Init();
    }
}
```

Unity
コマンドバトルゲーム

- GameParamUIPresenterにWaitGaugeRootにつけたWaitGaugeViewerを設定して、プレイしてみましょう。
- ゲージが徐々に塗りつぶされていくのが分かると思います。

- 次はCenterUIの制御をしていきます。
- ボタンを押して、キャラクター毎に定義されたアクションを粉うという処理になります。
- Scripts/UIフォルダ直下にCenterUIViewer.csを作成しましょう。
- また、それに合わせて各スクリプトも変更していきましょう。



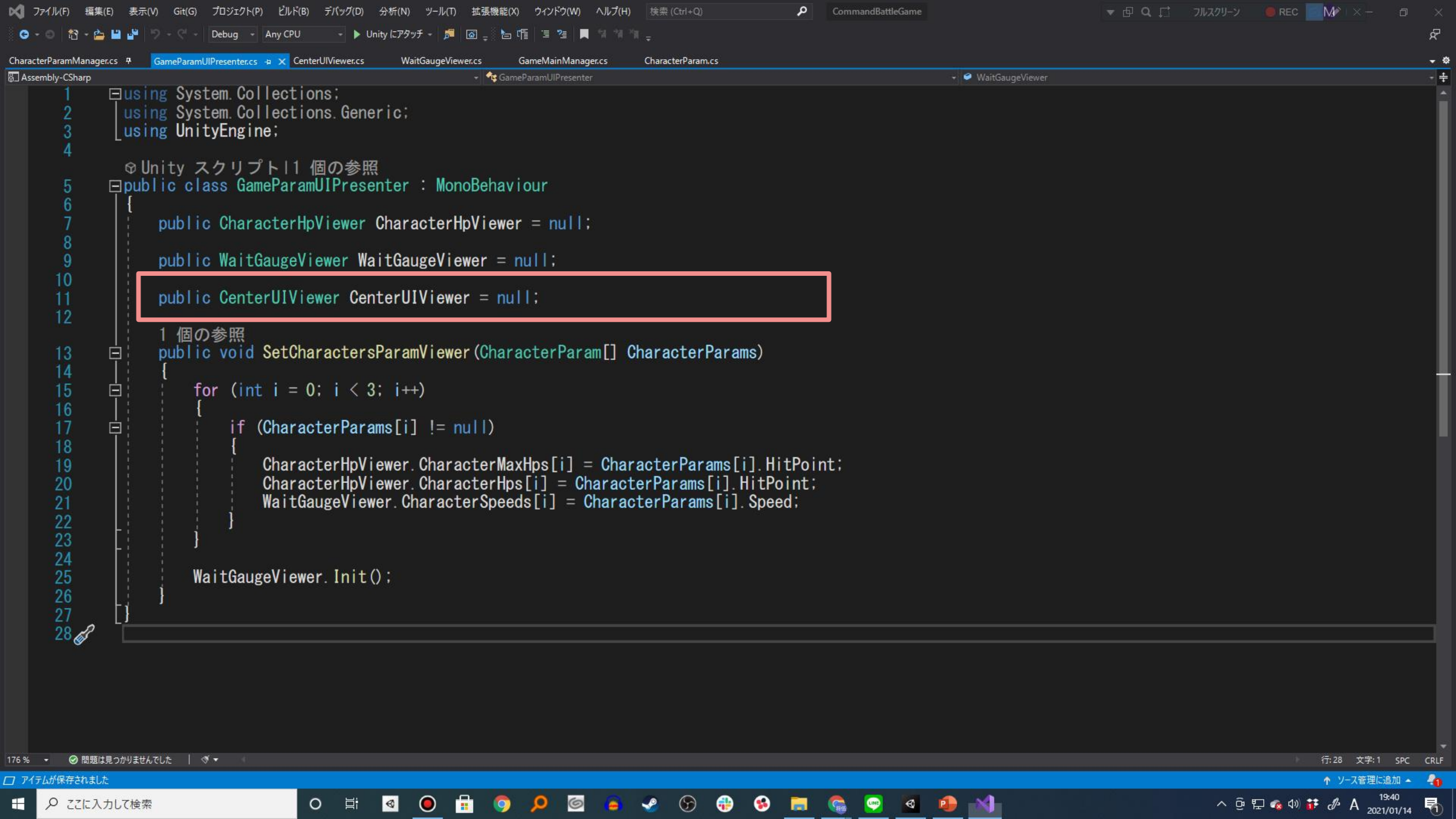
```
using System;
using System.Collections.Generic;
using UnityEngine;

8 個の参照
public class CharacterParam
{
    3 個の参照
    public enum GameCharacterType
    {
        Invalide,
        Attacker,
        SpellCaster,
        Healer,
    }

    public int HitPoint;
    public int MagicPoint;
    public float Speed;
    public GameCharacterType CharacterType;

    public Action FirstButtonAction;
    public Action SecondButtonAction;
    public Action ThirdButtonAction;
    public Action FourthButtonAction;
}
```

※Actionとは
Unityの機能として、System.Action型で変数を定義すると、そのAction型にはメソッドを代入することができます。
これは今回のように、同じボタンをキャラクター毎に使いまわす際に、メソッドの内容を変更しやすくすることができます。

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

Unity スクリプト | 1 個の参照
public class CenterUIViewer: MonoBehaviour {

    public Button[] CharacterActionButtons = new Button[4];
    public bool CenterUIVisible = false;

    1 個の参照
    public bool CheckCenterUIVisible(float gaugeRate)
    {
        return gaugeRate >= 1f;
    }

    1 個の参照
    public void SetCenterUIVisible(bool active)
    {
        this.gameObject.SetActive(active);
    }

    1 個の参照
    public void SetCharacterActionButtons(CharacterParam characterParam)
    {
        for (int i = 0;i < CharacterActionButtons.Length;i++)
        {
            CharacterActionButtons[i].onClick.RemoveAllListeners();
        }
        CharacterActionButtons[0].onClick.AddListener(() => characterParam.FirstButtonAction());
        CharacterActionButtons[1].onClick.AddListener(() => characterParam.SecondButtonAction());
        CharacterActionButtons[2].onClick.AddListener(() => characterParam.ThirdButtonAction());
        CharacterActionButtons[3].onClick.AddListener(() => characterParam.FourthButtonAction());
    }
}
```

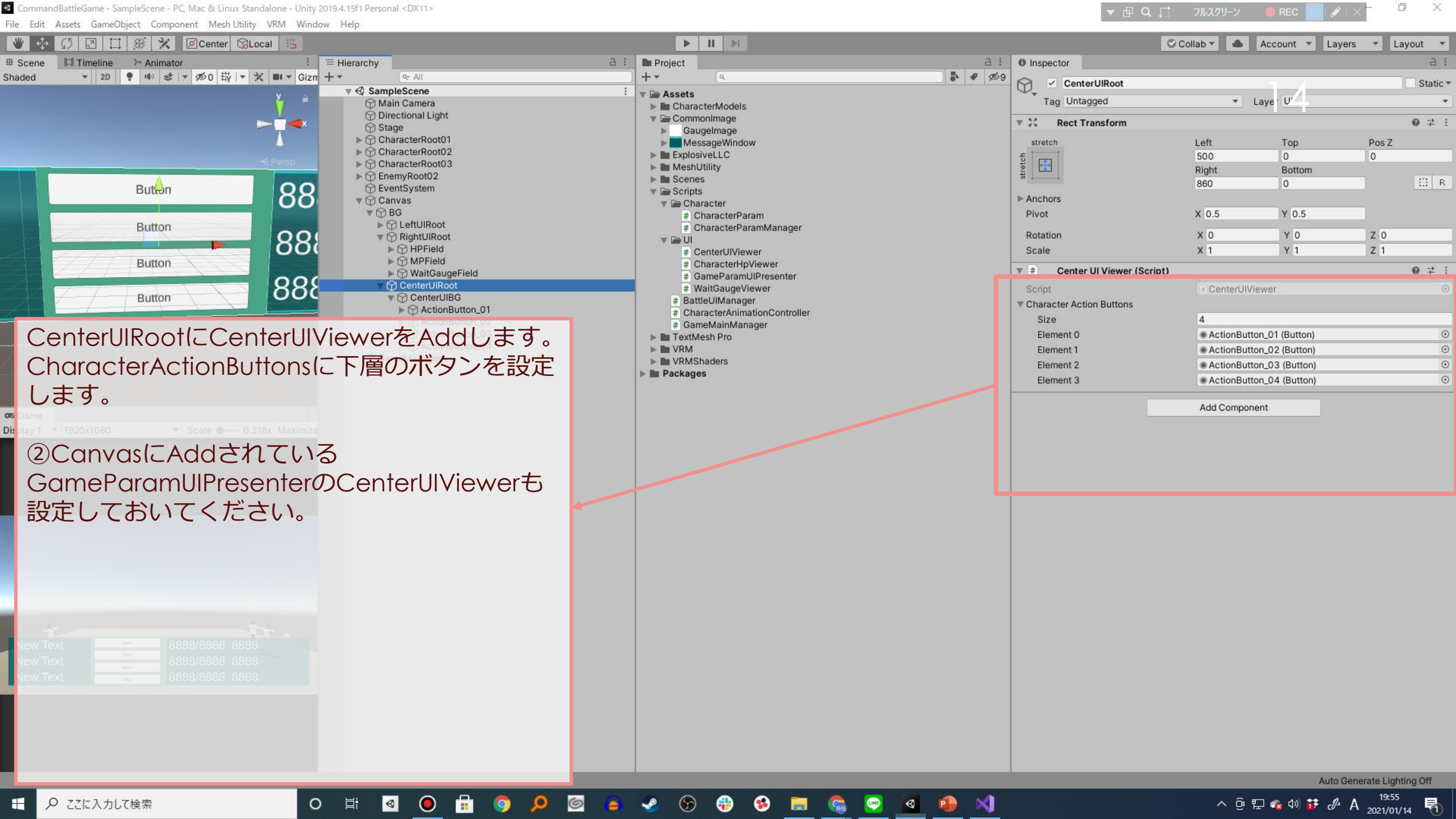

※ラムダ式
このスクリプトで初めて見る記述方法があると思います。
Hogehoge(()=>{Hogehoge()});
のように、メソッドの中に、コールバックとしてメソッドを定義する時に使用します。
今回は、ボタンを押した時に実行されるメソッドをCharacterParamで定義されたものを使うことにしました。

※コールバックとは
Unityには様々なイベントがあり、それに対しての反応のことです。例えばボタンを押す、離す、押し続ける等、ユーザーの操作によって引き起こされる挙動のことです。


ファイル(F) 編集(E) 表示(V) Git(G) プロジェクト(P) ビルド(B) デバッグ(D) 分析(N) ツール(T) 拡張機能(X) ウィンドウ(W) ヘルプ(H)
検索 (Ctrl+Q)
CommandBattleGame
Debug
Any CPU
Unity にアタッチ
CharacterParamManager.cs
GameParamUIPresenter.cs
CenterUIViewer.cs
WaitGaugeViewer.cs
GameMainManager.cs
CharacterParam.cs
Assembly-CSharp
GameParamUIPresenter
WaitGaugeViewer
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
Unity スクリプト | 1 個の参照
public class GameParamUIPresenter : MonoBehaviour
{
public CharacterHpViewer CharacterHpViewer = null;
public WaitGaugeViewer WaitGaugeViewer = null;
public CenterUIViewer CenterUIViewer = null;
1 個の参照
public void SetCharactersParamViewer(CharacterParam[] CharacterParams)
{
for (int i = 0; i < 3; i++)
{
if (CharacterParams[i] != null)
{
CharacterHpViewer.CharacterMaxHps[i] = CharacterParams[i].HitPoint;
CharacterHpViewer.CharacterHps[i] = CharacterParams[i].HitPoint;
WaitGaugeViewer.CharacterSpeeds[i] = CharacterParams[i].Speed;
}
}
WaitGaugeViewer.Init();
}
}
176 %
問題は見つかりませんでした
行: 28 文字: 1 SPC CRLF
アイテムが保存されました
ソース管理に追加
ここに入力して検索
19:40
2021/01/14

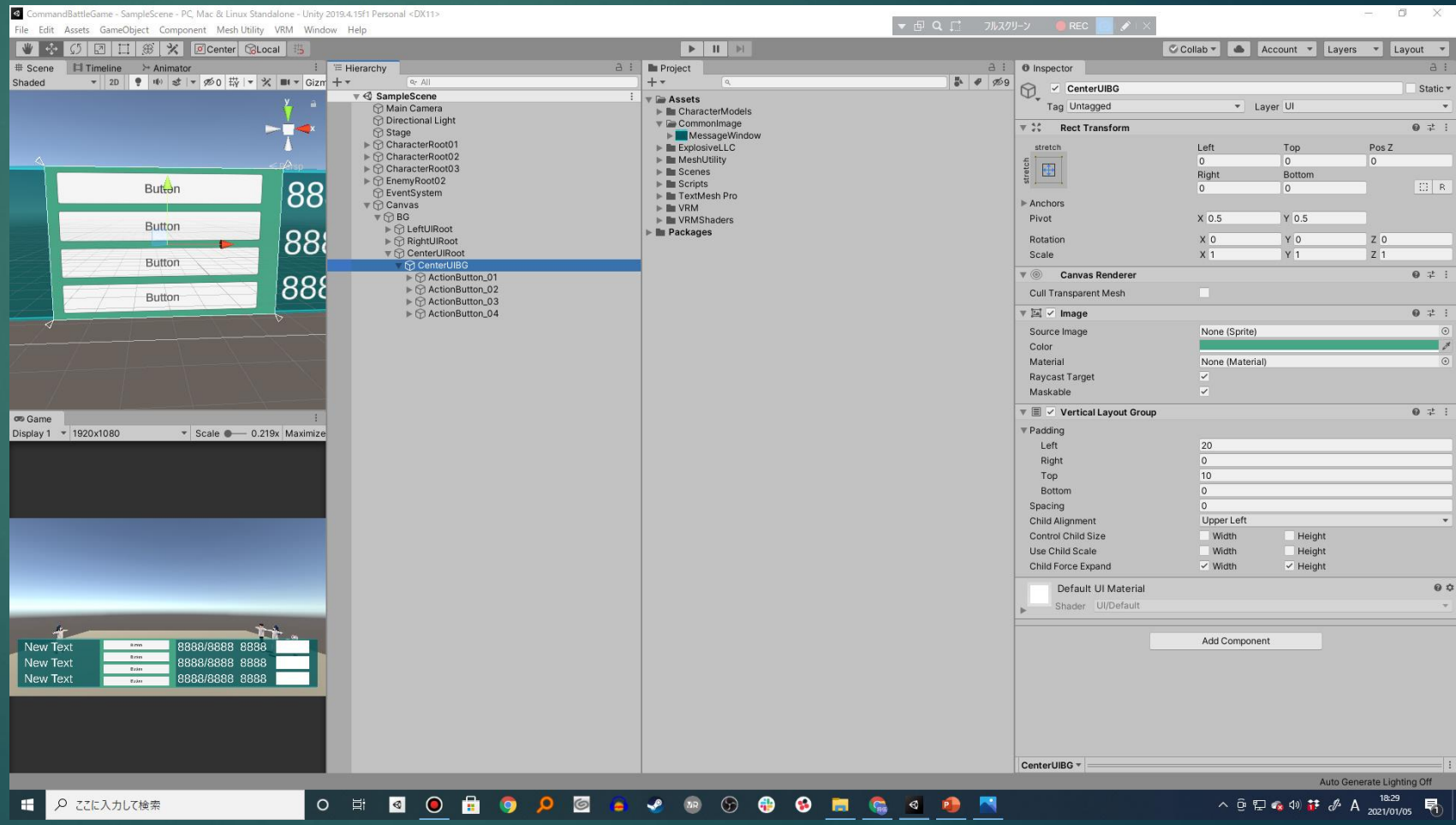
```
// 本来はサーバーから送られてくる事が多いのですが、今回は各キャラクター毎に設定します
public int CharacterHP = 0;
public int CharacterMP = 0;
public float CharacterSpeed = 0;
public CharacterParam.GameCharacterType CharacterType = CharacterParam.GameCharacterType.Invalide;

1 個の参照
private void Init()
{
    CharacterParam.HitPoint = CharacterHP;
    CharacterParam.MagicPoint = CharacterMP;
    CharacterParam.Speed = CharacterSpeed;
    CharacterParam.CharacterType = CharacterType;

    CharacterParam.FirstButtonAction = FirstButtonAction;
    CharacterParam.SecondButtonAction = SecondButtonAction;
    CharacterParam.ThirdButtonAction = ThirdButtonAction;
    CharacterParam.FourthButtonAction = FourthButtonAction;
}

Unity メッセージ | 0 個の参照
private void Awake()
{
    Init();
}

1 個の参照
private void FirstButtonAction()
{
    Debug.Log("aa");
}
1 個の参照
private void SecondButtonAction()
{
    Debug.Log("bb");
}
```

```
        CharacterParam.SecondButtonAction = SecondButtonAction;
        CharacterParam.ThirdButtonAction = ThirdButtonAction;
        CharacterParam.FourthButtonAction = FourthButtonAction;
    }

    Unity メッセージ | 0 個の参照
    private void Awake()
    {
        Init();
    }

    1 個の参照
    private void FirstButtonAction()
    {
        Debug.Log("aa");
    }
    1 個の参照
    private void SecondButtonAction()
    {
        Debug.Log("bb");
    }
    1 個の参照
    private void ThirdButtonAction()
    {
        Debug.Log("cc");
    }
    1 個の参照
    private void FourthButtonAction()
    {
        Debug.Log("dd");
    }
}
```


CommandBattleGame - SampleScene - PC, Mac & Linux Standalone - Unity 2019.4.15f1 Personal <DX11>
File Edit Assets GameObject Component Mesh Utility VRM Window Help
Hierarchy
SampleScene
Main Camera
Directional Light
Stage
CharacterRoot01
CharacterRoot02
CharacterRoot03
EnemyRoot02
EventSystem
Canvas
BG
LeftUIRoot
RightUIRoot
HPField
MPField
WaitGaugeField
CenterUIRoot
CenterUIBG
ActionButton_01
Project
Assets
CharacterModels
CommonImage
GaugeImage
MessageWindow
ExplosiveLLC
MeshUtility
Scenes
Scripts
Character
CharacterParam
CharacterParamManager
UI
CenterUIViewer
CharacterHpViewer
GameParamUIPresenter
WaitGaugeViewer
BattleUIManager
CharacterAnimationController
GameMainManager
TextMesh Pro
VRM
VRMShaders
Packages
Inspector
CenterUIRoot
Tag Untagged
Rect Transform
stretch
Left 500
Top 0
Pos Z 0
Right 860
Bottom 0
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Center UI Viewer (Script)
Script CenterUIViewer
Character Action Buttons
Size 4
Element 0 ActionButton_01 (Button)
Element 1 ActionButton_02 (Button)
Element 2 ActionButton_03 (Button)
Element 3 ActionButton_04 (Button)
Add Component
Button
Button
Button
Button
CenterUIRootにCenterUIViewerをAddします。
CharacterActionButtonsに下層のボタンを設定します。
②CanvasにAddされている
GameParamUIPresenterのCenterUIViewerも
設定しておいてください。
Auto Generate Lighting Off

Unity
コマンドバトルゲーム

- これで最低限の実装は終了しました。
- ただ、CenterUIViewerのSetCharacterActionButtonsをまだ呼んでいないので、呼ぶところを設定します。
- 例えばAttackerの行動を読んでみましょう。
- CharacterParamManagerでまずは一番上のボタンを押すと攻撃できるようにしましょう。
- その後、GameMainManagerで実際に挙動を実装して確認します。



```
    0 個の参照
    public CharacterParam GetCharacterParam
    {
        get { return CharacterParam; }
    }

    // 本来はサーバーから送られてくる事が多いのですが、今回は各キャラクター毎に設定します
    public int CharacterHP = 0;
    public int CharacterMP = 0;
    public float CharacterSpeed = 0;
    public CharacterParam.GameCharacterType CharacterType = CharacterParam.GameCharacterType.Invalide;

    public CharacterAnimationController CharacterAnimationController = null;

    1 個の参照
    private void Init()
    {
        CharacterParam.HitPoint = CharacterHP;
        CharacterParam.MagicPoint = CharacterMP;
        CharacterParam.Speed = CharacterSpeed;
        CharacterParam.CharacterType = CharacterType;

        CharacterParam.FirstButtonAction = FirstButtonAction;
        CharacterParam.SecondButtonAction = SecondButtonAction;
        CharacterParam.ThirdButtonAction = ThirdButtonAction;
        CharacterParam.FourthButtonAction = FourthButtonAction;


    }

    Unity メッセージ | 0 個の参照
    private void Awake()
    {
        Init();
    }
```

①Init()の中でCharacterAniamationController
をスクリプトで設定してください。
ヒント：自分に設定されているはずです

```
    1 個の参照
    private void Init()
    {
        CharacterParam.HitPoint = CharacterHP;
        CharacterParam.MagicPoint = CharacterMP;
        CharacterParam.Speed = CharacterSpeed;
        CharacterParam.CharacterType = CharacterType;

        CharacterParam.FirstButtonAction = FirstButtonAction;
        CharacterParam.SecondButtonAction = SecondButtonAction;
        CharacterParam.ThirdButtonAction = ThirdButtonAction;
        CharacterParam.FourthButtonAction = FourthButtonAction;


    }

    Unity メッセージ | 0 個の参照
    private void Awake()
    {
        Init();
    }

    1 個の参照
    private void FirstButtonAction()
    {

    }

    1 個の参照
    private void SecondButtonAction()
    {
        Debug.Log("bb");
    }
    1 個の参照
    private void ThirdButtonAction()
    {
```

①FirstButtonAction()の中で今回設定した
CharacterAnimationControllerの
StartAttackAnimation(2)を読んでください。

ヒント①：今のままでは呼べないので、
StartAttackAnimation(2)にpublic修飾子をつけましょう

ヒント②：呼び出し方はIEnumerator型に即した呼び出し方をしてください。

```
19      public CharacterParamManager[] CharacterParamManagers = new CharacterParamManager[3];
20
21      private CharacterParam[] characterParams = new CharacterParam[3];
22
23
        Unity メッセージ|0 個の参照
24      private void Update()
25      {
26          switch (GameState)
27          {
28              case State.Init:
29                  for (int i = 0; i < 3; i++)
30                  {
31                      if (CharacterParamManagers[i] != null)
32                      {
33                          characterParams[i] = CharacterParamManagers[i].CharacterParam;
34                      }
35                  }
36                  gameParamUIPresenter.SetCharactersParamViewer(characterParams);
37                  GameState = State.Start;
38                  break;
39              case State.Start:
40
41
42
43
44                  break;
45              case State.Command:
46
47
48
49                  break;
50
51              case State.Action:
52
53                  break;
```

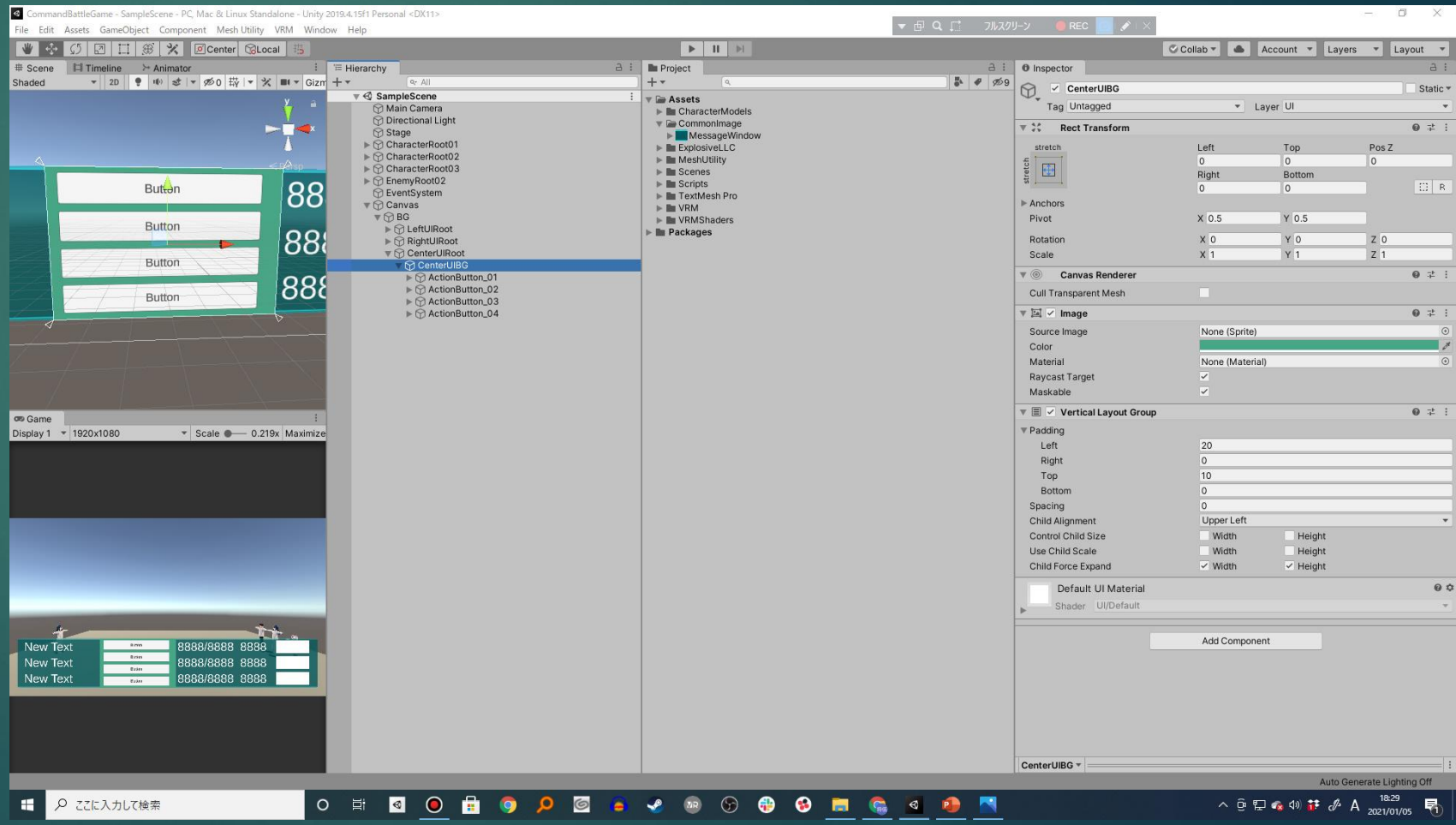
①State:Startに来たら、そのままGameStateをCommandに移動して下さい。

②State:Commandになったら、gameParamUIPresenterのCenterUIViewerからSetCharacterActionButtonsを呼びしょう。今回の引数はAttackerのCharacterParamを指定します

Unity
コマンドバトルゲーム

- では実際にゲームをプレイしてみましょう。

- ボタンを押すとAttackerが攻撃のモーションを行う事を確認します。
- ちょっと順番が前後しますが、後はゲージが溜まったら、CenterUIを表示するようにします。
- まず、WaitGaugeViewerの中で現在のキャラクターに設定されているゲージの埋まり具合をゲットしましょう。
- その後、GameMainManagerの中で設定します。



```
public class WaitGaugeViewer : MonoBehaviour
{
    public Image[] CharacterGaugeImages = new Image[3];
    public float[] CharacterSpeeds = new float[3];

    1 個の参照
    public void Init()
    {
        for (int i = 0; i < 3; i++)
        {
            CharacterGaugeImages[i].fillAmount = 0;
        }
    }

    1 個の参照
    public void WaitGaugeUpDate()
    {
        for (int i = 0; i < 3; i++)
        {
            CharacterGaugeImages[i].fillAmount += CharacterSpeeds[i]/10000f;
        }
    }

    // Updateで計算をした後にSpeedの反映が入るようにLateUpdateを使用する
    Unity メッセージ | 0 個の参照
    private void LateUpdate()
    {
        WaitGaugeUpDate();
    }
}
```

①publicでfloat型のGetWaitGaugeRateを作成します。また、Int型で引数を設定します。(現在は0にAttackerが接地されているので、0をいれるとAttackerのゲージのfillAmountが返るようにします)

②returnで引数に設定された数字によってCharacterGaugeImages[引数].fillamountを返すようにします。

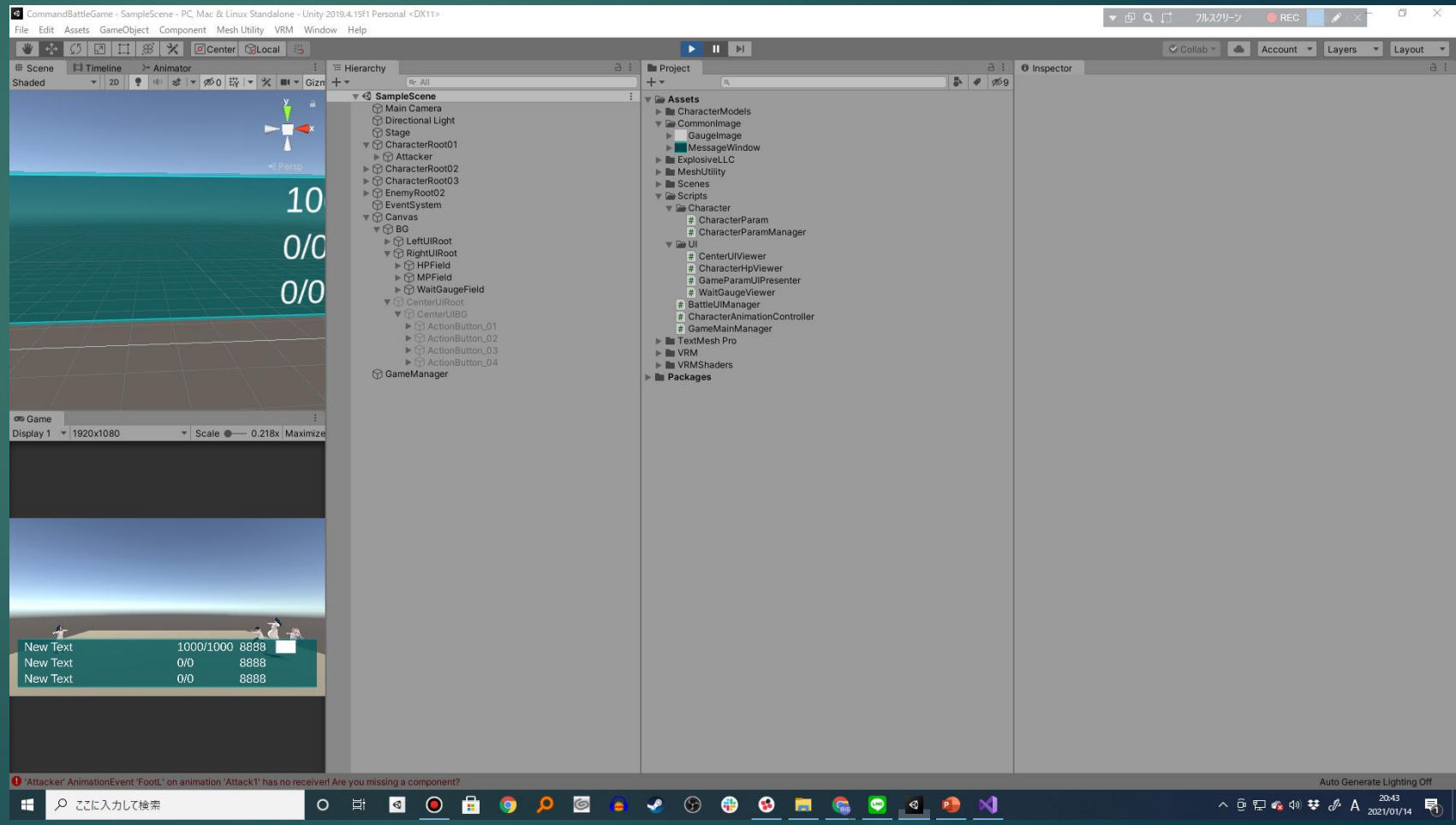
```
24      private void Update()
25      {
26          switch (GameState)
27          {
28              case State.Init:
29                  for (int i = 0; i < 3; i++)
30                  {
31                      if (CharacterParamManagers[i] != null)
32                      {
33                          characterParams[i] = CharacterParamManagers[i].CharacterParam;
34                      }
35                  }
36                  gameParamUIPresenter.SetCharactersParamViewer(characterParams);
37                  GameState = State.Start;
38                  break;
39              case State.Start:
40
41
42
43                  break;
44              case State.Command:
45
46
47
48                  break;
49
50              case State.Action:
51
52                  break;
53
54              case State.Result:
55
56                  break;
57
58          }
59      }
```

①State:Startの時に、Commandに行く条件として、gameParamUIPresenterのCenterUIViewerのCheckCenterUIVisibleを判定として使います。
②引数はgameParamUIPresenterのWaitGaugeViewerのGetWaitGaugeRateを使います。
③ GetWaitGaugeRateの引数は0を設定します。

④その条件の中に入った時に、gameParamUIPresenterのCenterUIViewerのSetCenterUIVisibleの引数をtrueにして呼びましょう。

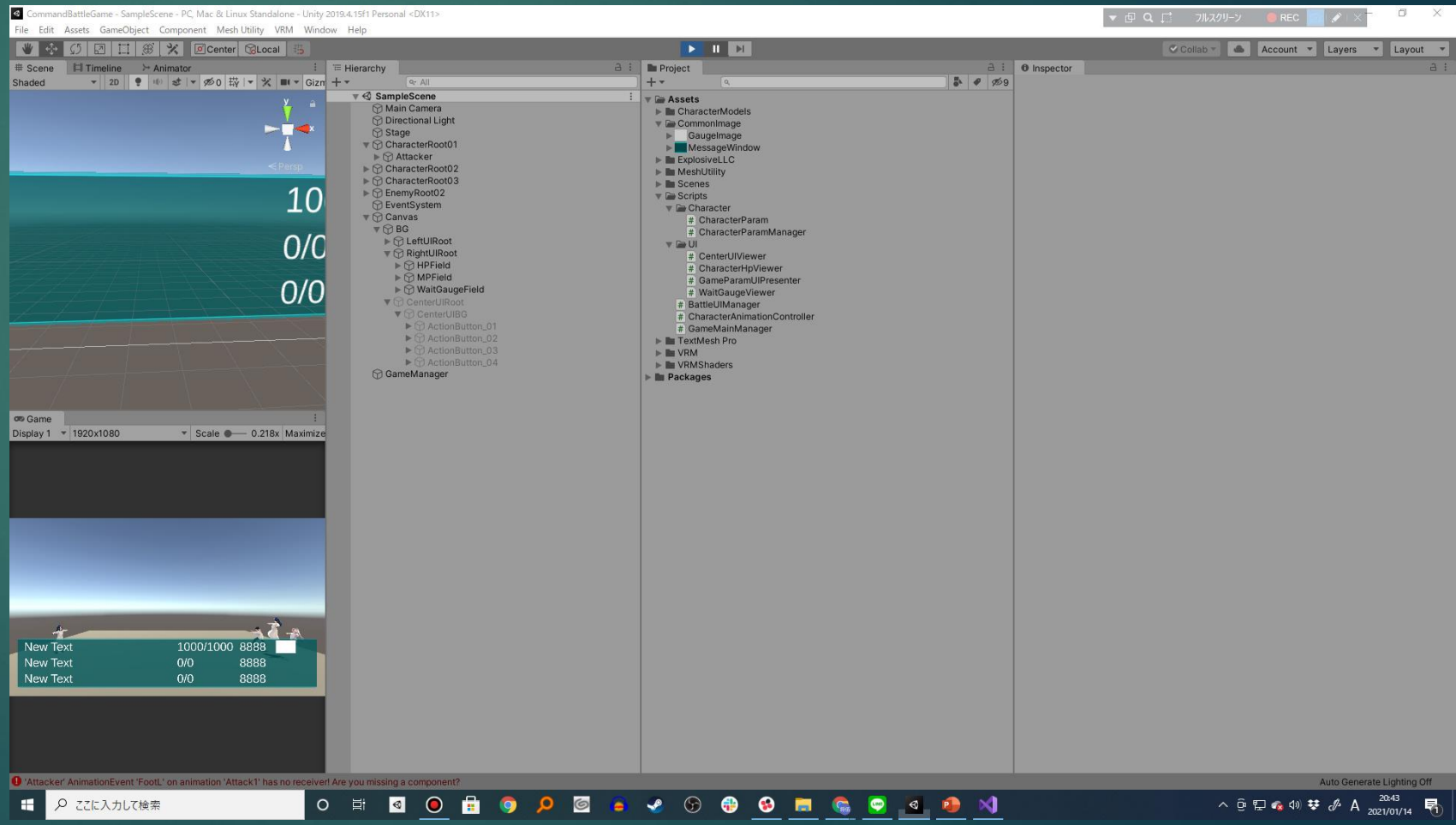
Unity
コマンドバトルゲーム

- では実際にゲームをプレイしてみましょう。
- ゲージの進捗が溜まるまではボタンのUIは表示されず、溜まり切ったら表示されるようになっていると思います。
- もちろん、表示された一番上のボタンを押すと攻撃モーションを発動しますね。



Unity
コマンドバトルゲーム

- 今からですが、ちょっと各人考えていただき、三人のアニメーションを設定できるようにしましょう。
- ①まずは各キャラクターにCharacterParamManagerをAddして、パラメーターを設定します。
- ②GameMainManagerのCharacterParamManagersに設定します。
- ③まずここまでで、ゲージが各キャラクターごとに動くかを確認してください。



Unity
コマンドバトルゲーム

- ここからは難題です。
- 宿題となります。
- 来週答え合わせをします。
- ④今、GameMainMangerのState.Commandへの移行は0(Attacker)のゲージレートを取得していますが、これを各キャラクターのうち、一番早く埋まり切ったキャラクターに変更してください。
- ⑤また、一番早く埋まり切ったキャラクターのアニメーションをボタンの一番上で再生するようにしましょう。
- ⑥ボタンを押したらCenterUIRootが消えるようにしてみましょう。