

前提知識：配列

C#で言う「配列」

配列：同じ型を同時にたくさん使う方法の一つ

「配列」の書き方

型 [] 配列名 = new 型 [要素数];

C#で言う「配列」

例:

```
int [] a = new int [3];
```

```
a[0] = 123;
```

```
a[1] = 234;
```

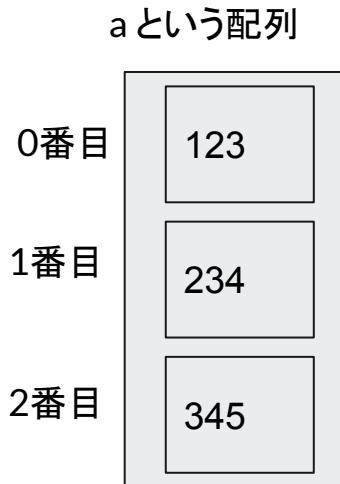
```
a[2] = 345;
```

とした時

a.Length ← 配列の要素数



int [] a = { 123 , 234 , 345 }



C#で言う「配列」

例：

```
int [] a = new int [3];
```

とした時

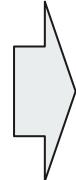
```
a[0] = 123;
```

```
a[1] = 234;
```

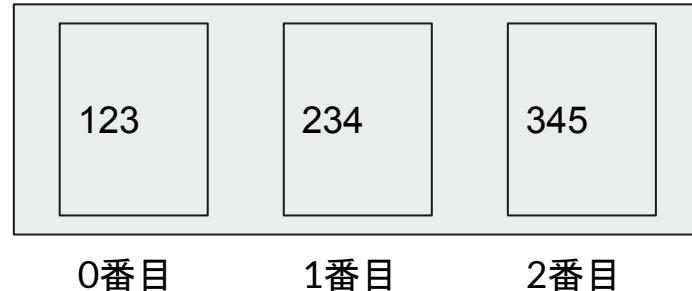
```
a[2] = 345;
```

```
a[3] = 456; //エラー
```

```
a = 567; //エラー
```



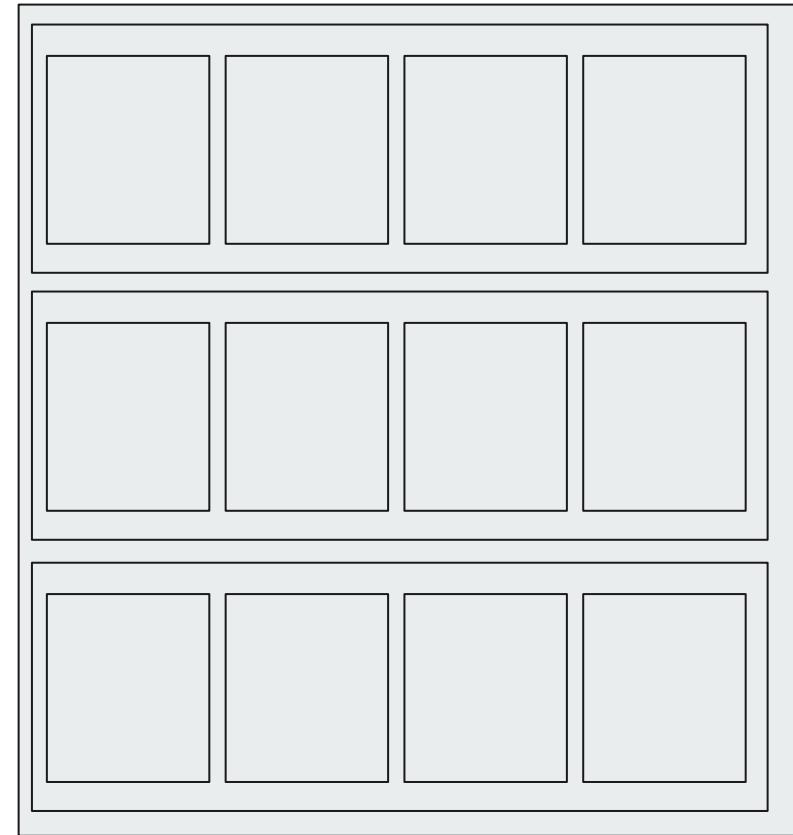
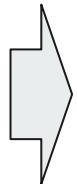
a という配列





多次元配列

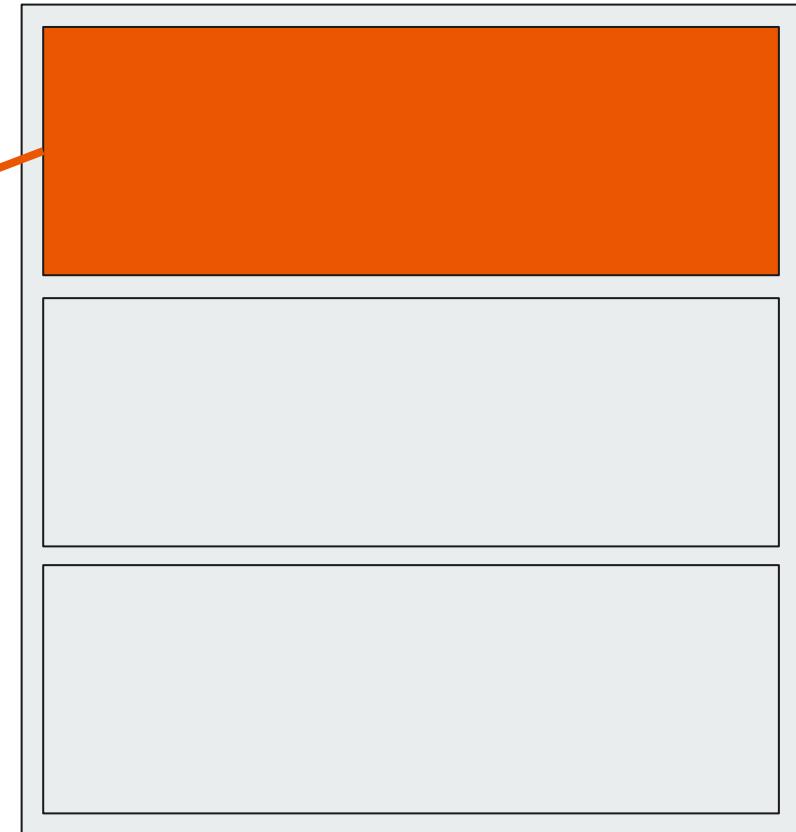
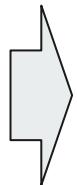
```
int [ , ] a = new int [ 3,4 ];
```



多次元配列

```
int [ , ] a = new int [ 3, 4 ];
```

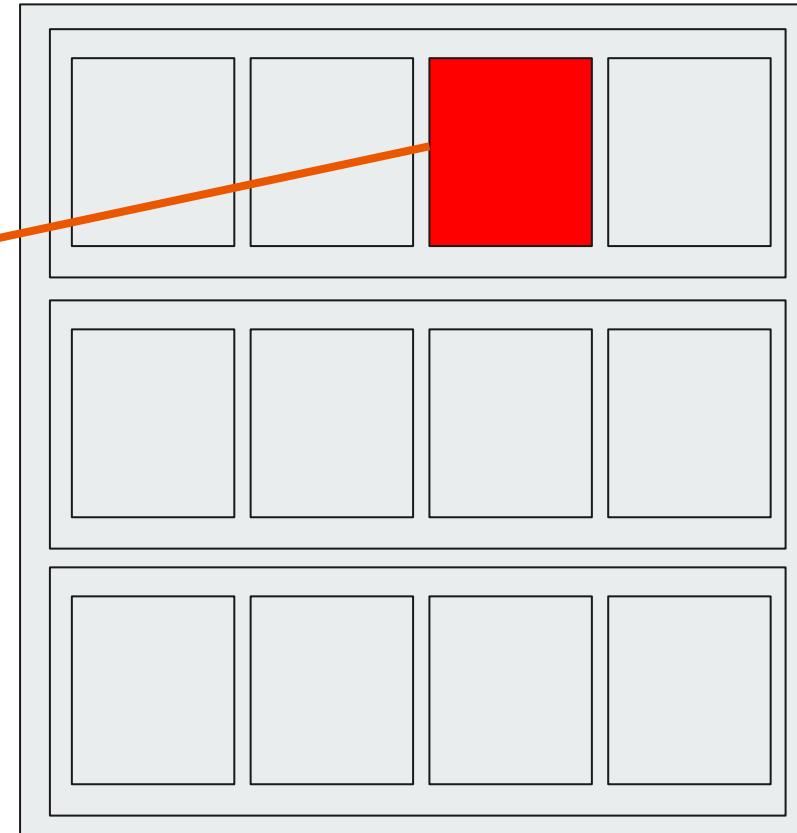
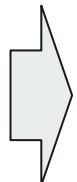
a[0]



多次元配列

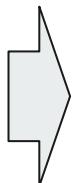
```
int [ , ] a = new int [ 3, 4 ];
```

a[0, 2]

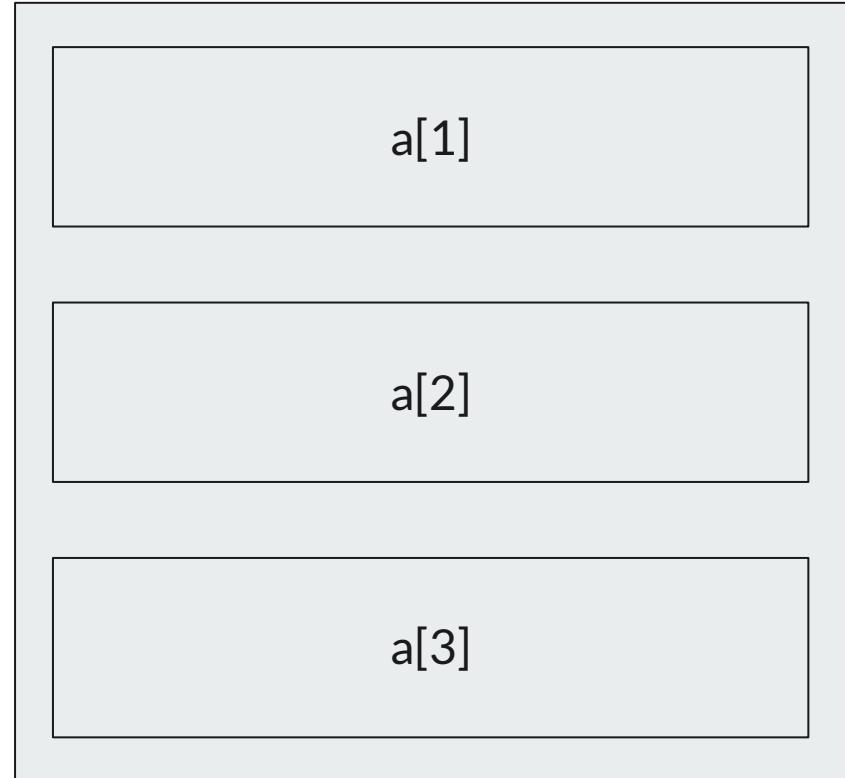


配列の配列

```
int [][] a = new int [3][];
```



変数a



配列の配列

```
int [][] a = new int [3][];
```

```
a[1] = new int[1];
```

```
a[2] = new int[3];
```

```
a[3] = new int[2];
```



変数a

変数a[1]



変数a[2]



変数a[3]

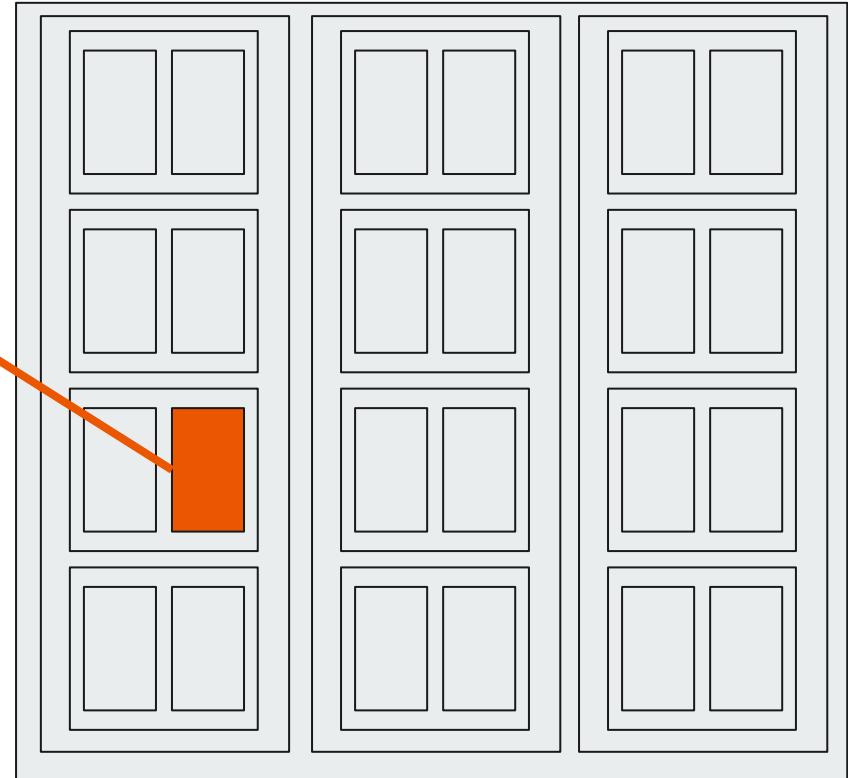
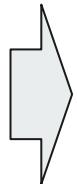




多次元配列

```
int [,,] a = new int [ 3,4,2 ];
```

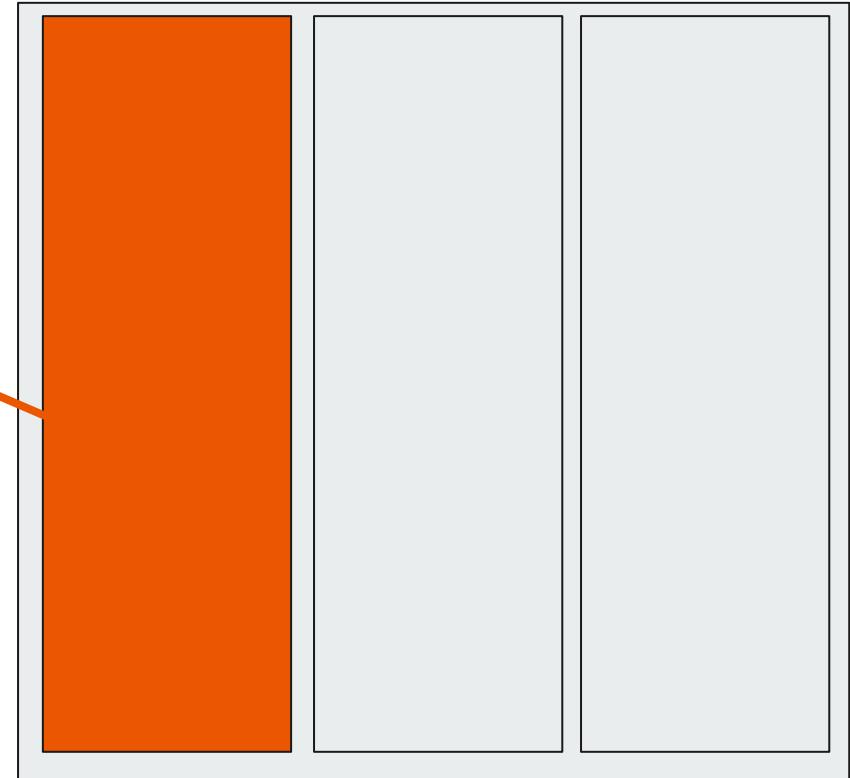
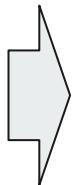
a[0,2,1]



多次元配列

```
int [ , , ] a = new int [ 3 , 4 , 2 ];
```

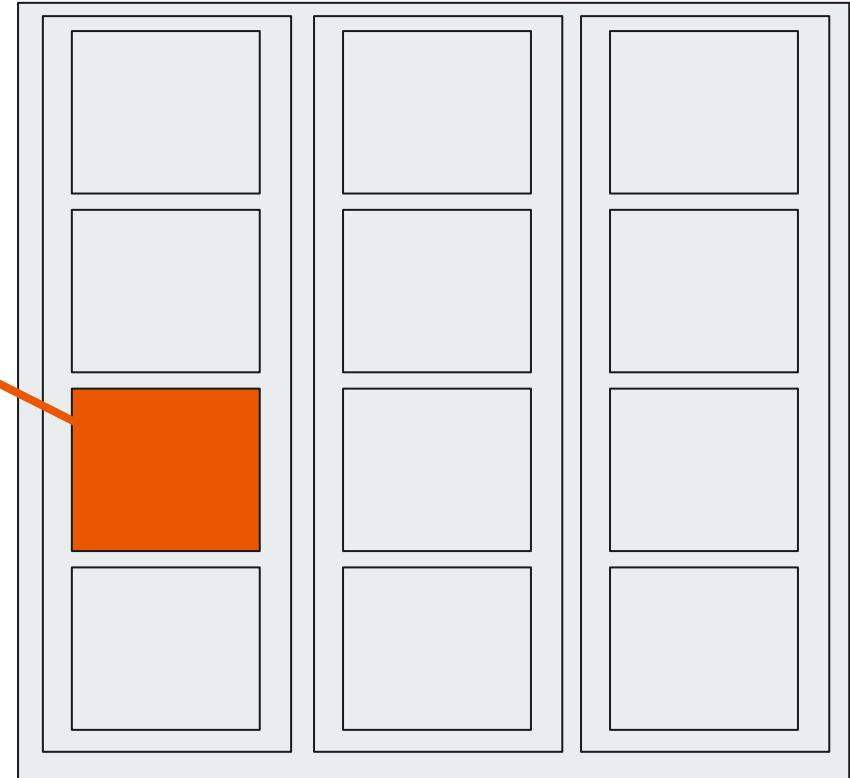
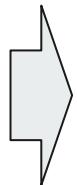
a[0, 2, 1]



多次元配列

```
int [ , , ] a = new int [ 3 , 4 , 2 ];
```

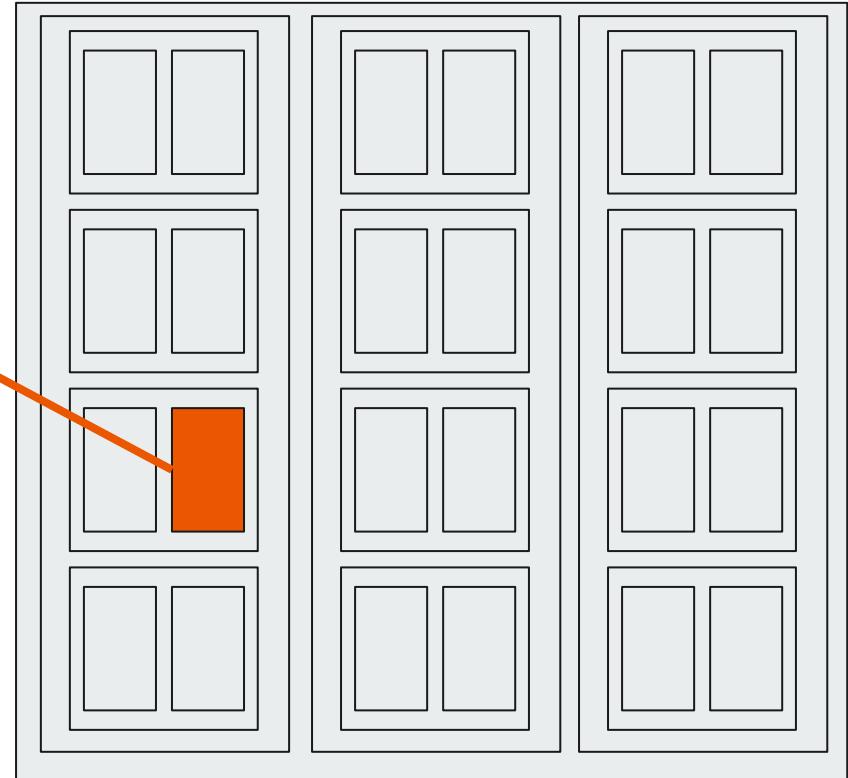
a[0, 2, 1]



多次元配列

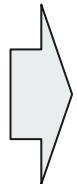
```
int [,,] a = new int [ 3,4,2 ];
```

a[0,2,1]



多次元配列

```
int [ , , ] a = new int [ 3 , 4 , 2 ];
```



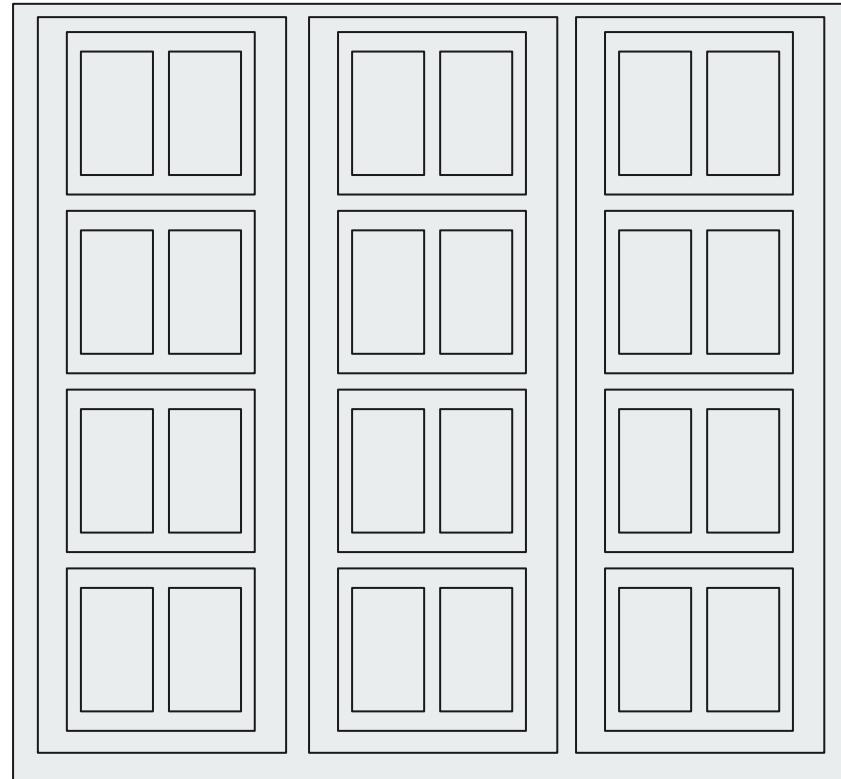
```
a.GetLength(0) //3    a.GetLength(1) //4
```



↑
指定した場所の要素数

a.Length //3×4×2=24

↑全体の要素数





配列型

配列型は、同じ型の複数の値を1つの固まりとして扱うための型です。配列に入れたデータ(要素)には「0」から順番に番号(インデックスと呼びます)が振られ、インデックスを使って値にアクセスし、データを読み取ったり変更したりできます。

要素の数を増やしたい場合は、新しい配列を丸ごと作り直して上書きする必要があります。

```
int[] values = new int[] {1, 2, 3};    「1, 2, 3」という3つのint型データが入った配列を作成する
Debug.Log(values.Length);    配列の要素が何個あるかをカウントする。3個入っているので、「3」が出力される
Debug.Log(values[0]);    0番目の要素を取得する。「1」が出力される
values[0] = 100;    0番目の要素を「100」に書き換える
Debug.Log(values[0]);    「100」が出力される
```

List型

List型は、配列型と似ていますが、要素の数を後から増やすことができます。

```
List<int> values = new List<int> {1, 2, 3};
```

1, 2, 3という3つのint型データが入ったListを作成する

```
Debug.Log(values.Count);
```

Listの要素が何個あるかをカウントする。3個入っているので、「3」が出力される

```
Debug.Log(values[0]);
```

0番目の要素を取得する。「1」が出力される

```
values[0] = 100;
```

0番目の要素を「100」に書き換える

```
Debug.Log(values[0]);
```

「100」が出力される

```
values.Add(999);
```

要素「999」を末尾に追加する

```
Debug.Log(values[values.Count - 1]);
```

「999」が出力される

Dictionary型

Dictionary型は辞書型とも呼ばれます。List型と似ていますが、自動で振られるインデックスの代わりに「キー」と「値」をセットにしてデータを保持します。

```
Dictionary<string, int> values = new Dictionary<string, int> {  
    {"いち", 1},  
    {"に", 2},  
    {"さん", 3},  
};
```

いち = 1, に = 2, さん = 3という3つのデータが入ったDictionaryを作成する

```
Debug.Log(values.Count);
```

Dictionaryの要素が何個あるかをカウントする。「3」が出力される

```
values["いち"] = 100;
```

"いち"の要素を「100」に書き換える

```
Debug.Log(values["いち"]);
```

「100」が出力される

```
values.Add("よん", 999);
```

"よん" = 999 の要素を追加する

```
Debug.Log(values["よん"]);
```

「999」が出力される

Enum

```
//game_mode 0 の時は簡単
//game_mode 1 の時は普通
//game_mode 2 の時は難しい
int game_mode = 1; //初期モード
void Update(){
    //モード選択
    int game_mode = 0;
    if (game_mode == 0)
    {
        Debug.Log("簡単");
    }
}
```

Enum

```
enum Game_Mode
{
    easy,
    normal,
    Difficult
};
//初期モード
Game_Mode game_mode = Game_Mode.normal;
void Update()
{
    //イージーモード選択
    game_mode = Game_Mode.easy;
    if (game_mode == Game_Mode.easy)
    {
        Debug.Log("簡単");
    }
}
```

定数

変数はいつでも値を変更できるのに対して、定数は値を変更できません。

定数を使うことで「この値は変更しない」ことを明確にできますので、プログラムの見通しが良くなります。

```
const int Value = 10;    constを付けることで定数になる
```

ベクトル型

ベクトル型(3Dの場合はVector3、2Dの場合はVector2)はUnityではとても良く使う型の1つです。

ベクトルとは、「向き」と、その向きへの「大きさ」を表すものです。Vector3では、3D空間の3軸(X・Y・Z)の値を格納することができます。Vector2では、2D空間の2軸(X・Y)の値を格納することができます。

Unityでは、このベクトルを使って以下のような処理を行います。

- オブジェクトに対し、任意の方向に任意の力を加える
- オブジェクトを配置する座標を指定する

ベクトル型

```
private void Start() {  
    var position = new Vector3(0, 1, 2);  
    transform.position = position;    ワールド座標(0, 1, 2)にオブジェクトを配置する  
    Debug.Log(vector.normalized);    方向だけを表す(大きさが1の)ベクトルを取得する  
    Debug.Log(vector.magnitude);    大きさだけを取得する
```

ベクトルは演算も可能

```
Debug.Log(new Vector3(3, 1, 0) + new Vector3(5, 3, 1));
```

Vector3同士での加減算が可能。結果は(8, 4, 1)

```
Debug.Log(new Vector3(3, 1, 0) * 2 / 5);    intやfloatとの加減算が可能。
```

結果は(1.2f, 0.4f, 0)

ベクトルは、よく使う値がかんたんに使えるよう宣言されている

```
Debug.Log(Vector3.zero);    new Vector(0, 0, 0)と同値
```

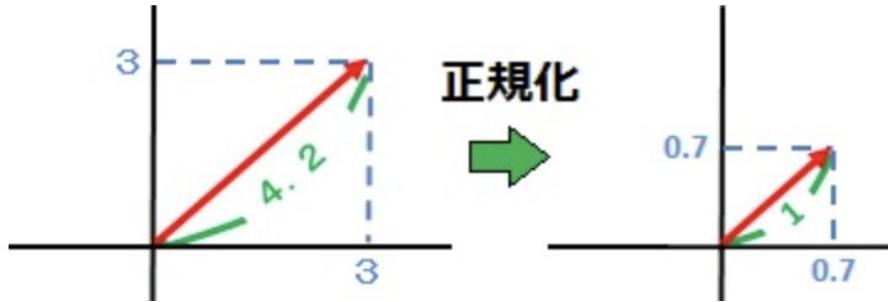
```
Debug.Log(Vector3.up);    new Vector(0, 1, 0)と同値
```

```
Debug.Log(Vector3.forward);    new Vector(0, 0, 1)と同値
```

```
}
```

ベクトルの正規化

ベクトルの正規化とは、ベクトルの方向は維持しつつ大きさを「1」にする事を指します。
この大きさが「1」のベクトルのことを単位ベクトルと呼びます。



```
Vector3 v = new Vector3(1, 2, 3);
Vector3 vn = v.normalized;
// vnは"vと同じ向き&長さが1"のベクトルになっている
```

```
Vector3 v = new Vector3(1, 2, 3);
v.Normalize();
// vが"向きはそのまま&長さが1"のベクトルになる
```

可読性は低くなる

1
Tips

変数にはvar(型推論)も使える

前述の例ではintやfloatなど明確に変数の型を宣言していますが、varで宣言することもできます。

varで宣言すると、初期値に応じて自動的に型が割り当てられます。

```
var a = 1;    int型になる
var b = 1.3f;  float型になる
var c = "test"; string型になる
var d = new Vector3(0, 0); Vector型になる
```

以下のように、初期値が空の場合はvarは使用できない

```
var e;
var f = null;
```

数値は明確にしたほうがいい

Coffee
Break

マジックナンバーは避けるべし

プログラム中、唐突に出てきて使用される数値をマジックナンバーといいます。

プログラムを書いた人はその数値の意味(その数値は何を表していて、どのような役目を果たすのか)を知っていますが、他の人が見てもすぐには意味がわからない場合があります。たとえば、キャラクターの移動スピードを制御する処理を例に挙げてみましょう。

```
rigidbody.velocity = rigidbody.velocity.normalized * 5;
```

この場合、「5」がマジックナンバー

この例の場合、Unityに慣れていれば「5は移動スピードだな」とわかるかもしれません、解説に時間がかかることがあります。

このようなものが増えると、プログラムがわかりづらくなるため、以下のように定数で宣言したりコメントを付けておくようにしましょう。

```
const int Speed = 5;    キャラクターの移動スピード
```

略

```
rigidbody.velocity = rigidbody.velocity.normalized * Speed;
```

これで他の人がプログラムを見ても「5はキャラクターの移動スピードだな！」と確実に理解できるようになります。

余談ですが、プログラムを書いてから数ヵ月経つと、書いた人自身が「他の人」と化してしまうことがしばしばあります(少なくとも筆者は、何ヵ月も前に書いたコードはほとんど覚えていません)。もし人に見せる予定が無くても、わかりやすく書くことを意識しておきましょう。

bool・int

◎ bool型

bool型は、true(真)またはfalse(偽)の2種類の値を格納できる、いちばんシンプルな型です。

```
bool value = true;
```

◎ int型

int型は、-2147483648～2147483647の範囲の整数を格納できます。この範囲を超える整数を扱う場合はlong型を使います。

```
int value = 128;
```

float型

float型は、小数を格納できます。C#では、数値の末尾に「f」を付けることでfloat型であることを表します。桁数が多いと誤差が出る（値がほんの少しだけ意図せず変わってしまう）のが特徴で、誤差なく扱える有効桁数は6～9桁程度です。

Unityは座標などの各所にfloat型を使っています。そのため、しばしば誤差が発生して、Inspectorウインドウに表示されている値が変わることがあります。

```
float value = 0.25f;    float型は、数値末尾にfを付ける  
float value2 = 5f;      小数点以下が無くてもOK
```

まとめ

List型 : 「配列」の要素数を後から増やせる版

Dictionary型 : JSONのようにKeyと値を1かたまりのデータとして複数扱える

Enum型 : 値の種類を名前であらかじめ宣言して1かたまりのデータとする

Const : 固定の値を持たせる。値を変更しないようにする

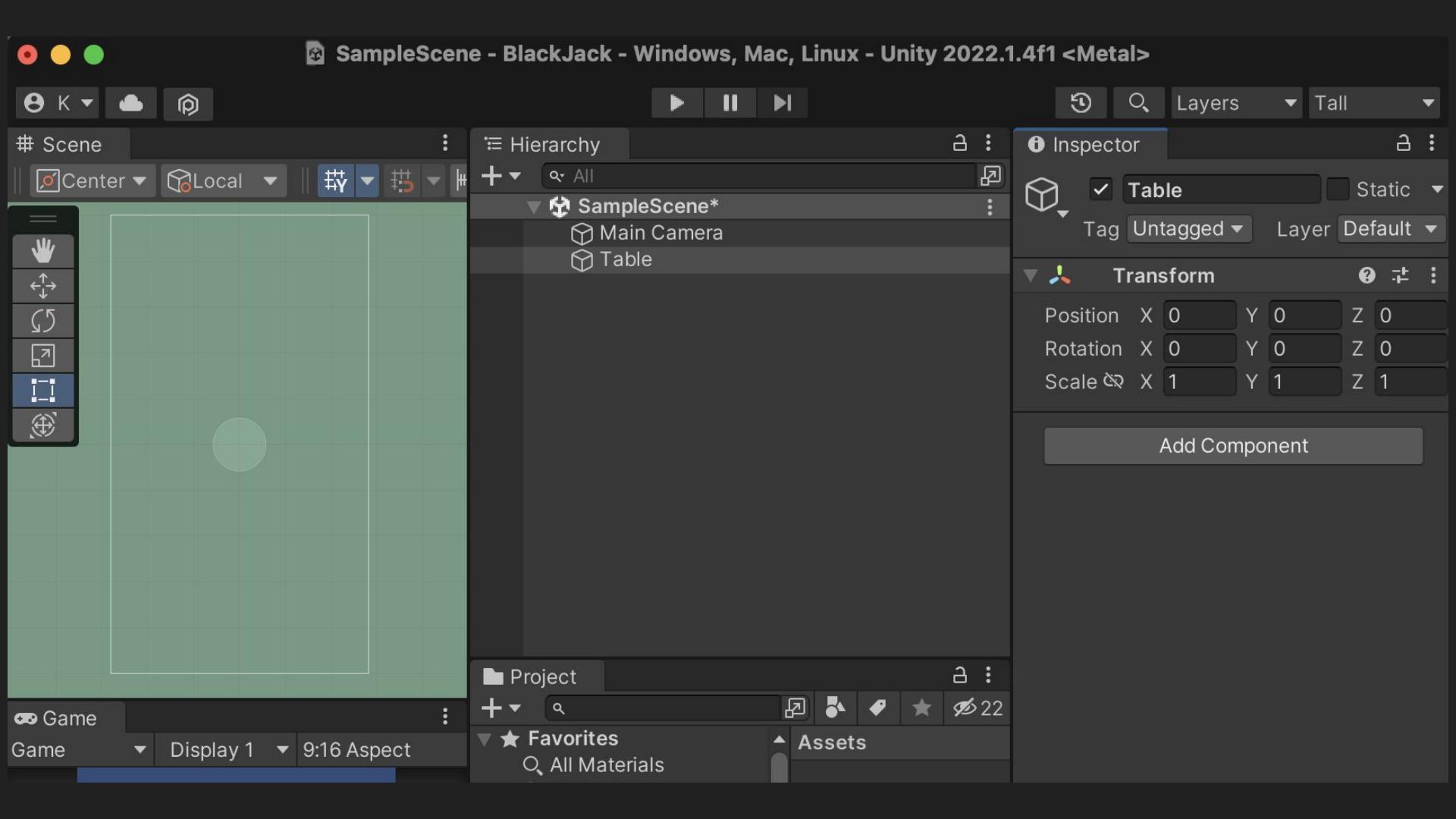
ベクトル型 : 構造体(次章で解説)。3DならVector3型になる

「向き」「大きさ」の情報を持つベクトル

値としては3D空間で(X,Y,Z)の値を格納できる

ブラックジャック

SampleScene - BlackJack - Windows, Mac, Linux - Unity 2022.1.4f1 <Metal>



- Cut
- Copy
- Paste
- Paste As Child

- Rename

- Duplicate

- Delete

- Select Children

- Set as Default Parent

- Create Empty

- Create Empty Parent

- 2D Object

- Square

- 3D Object

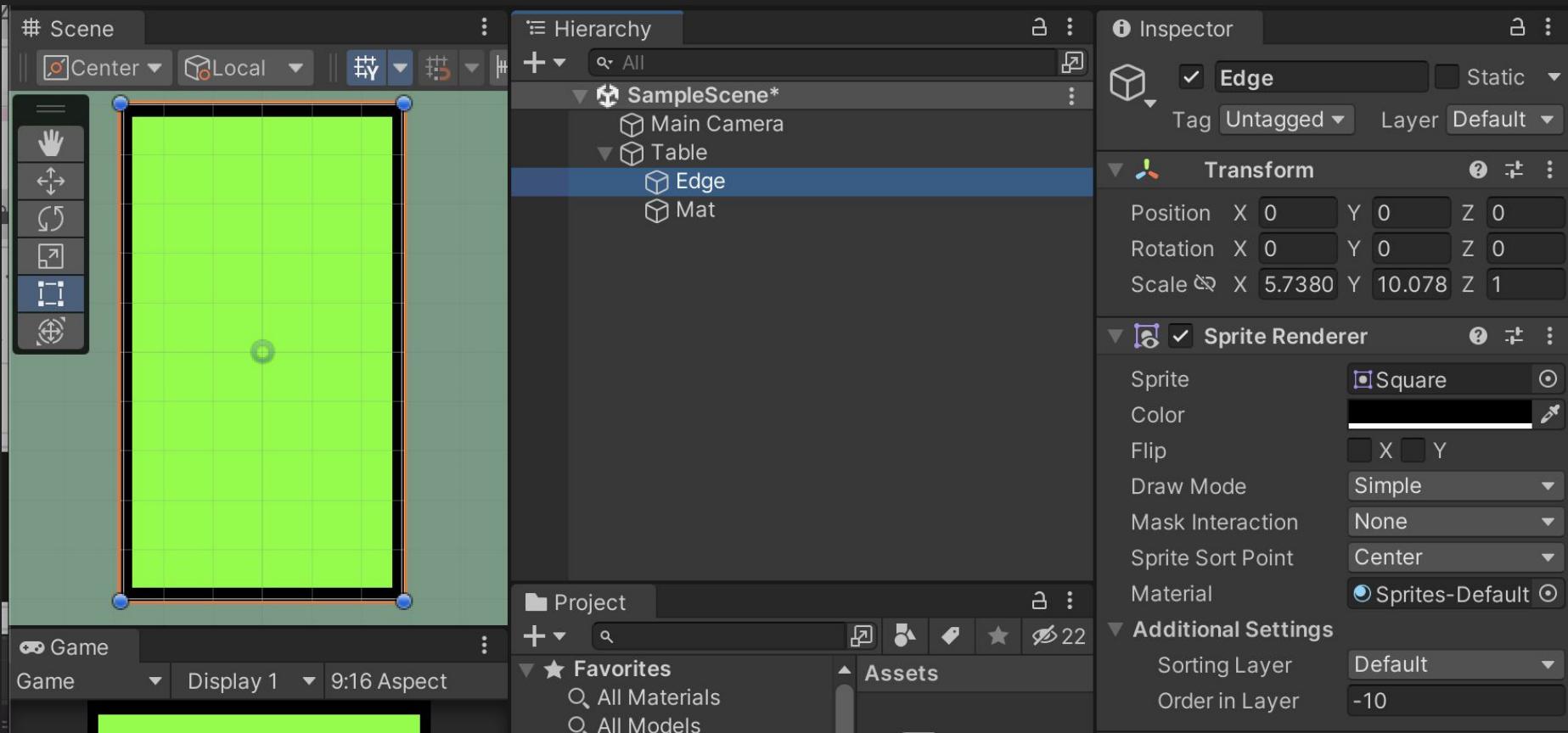
- Circle

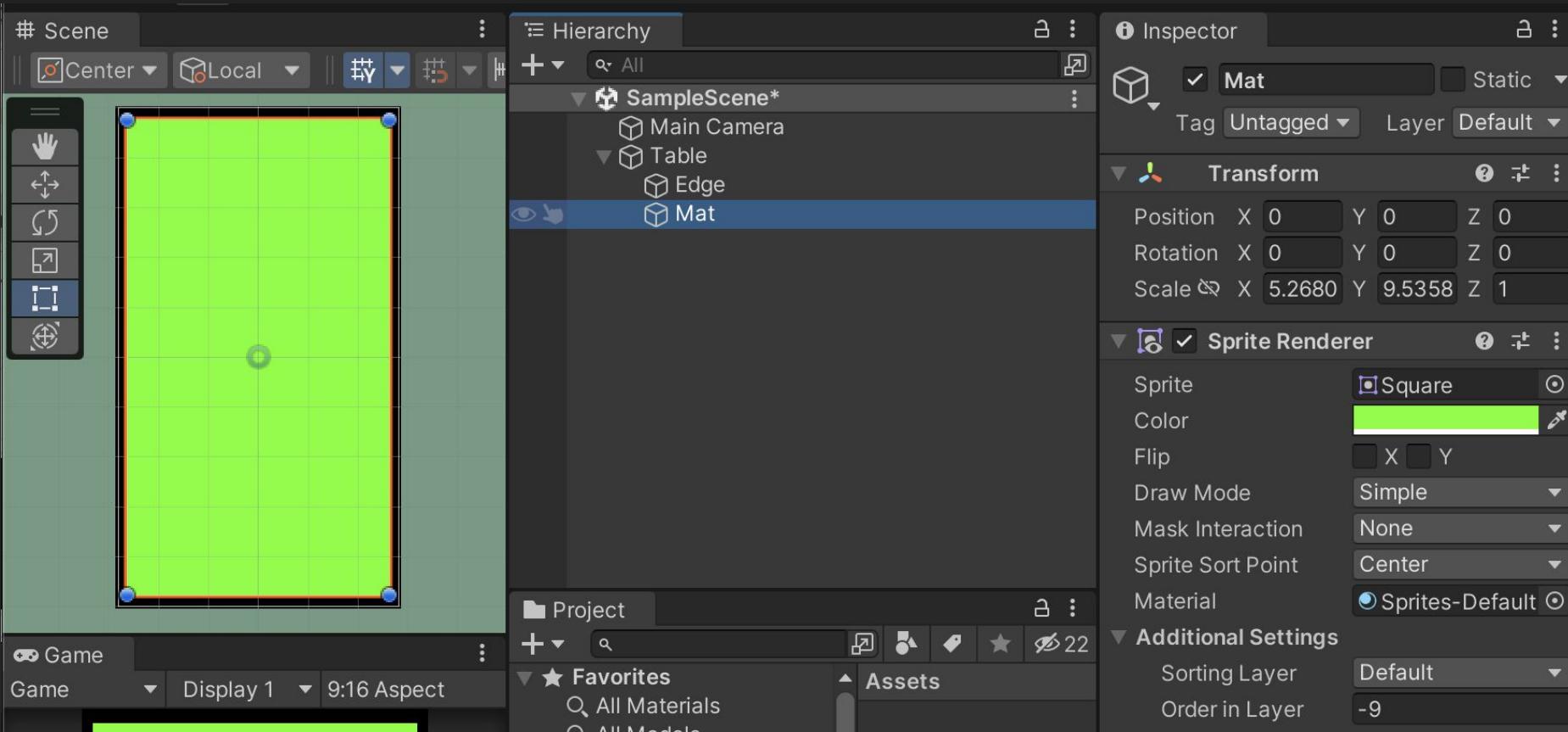
- Sprites

- >

- Physics

- >





ブラックジャックのゲーム画面とは？

ブラックジャックのゲーム画面とは？

カードが配られた初期状態

自分のターン

敵のターン

リザルト表示

ブラックジャックのゲーム画面とは？

- ・状態
 - カードが配られた初期状態
 - 自分のターン
 - 敵のターン
 - リザルト表示
- ・カードの表示

・カードの表示とは？

- ・カードの表示とは?
 - カード情報を配列として持つ
 - 配列を見て表示すべきものを表示する

- ・どういう配列を持つ？

- ・どういう配列を持つ?
 - 52個の要素を持たせて、後から13で割る

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

public class GameManager : MonoBehaviour
{
    //山札を作る
    int[] _cards;

    //カードの枚数
    int _totalNum = 52;

    //初期状態
    void InitialCards() {
        _cards = new int[_totalNum];//52個の要素が入る配列
        for (int i = 0; i < _totalNum; i++) {
            _cards[i] = i;//実際に数字を入れる
        }
    }

    //52枚のカードが入った
    //あとは13で割った時の「余り」と「商」から
    //「カードの数字」と「カードの種類」を判別する
}
```

0~51までの数がそのままカードに対応する

0~51までの数(カード)を配ったり、
配列 자체を山札と考え、シャッフルする

カードを一枚ずつ配ってみる

配るとは？

配るとは？

- 配列から要素を抜き取ること
- 抜き取った要素を手札に加える

PlayerBase.cs

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

public class PlayerBase : MonoBehaviour
{
    //手札のリスト
    List<int> _hand = new List<int>();

    //手札にカードを加える
    void RecieveCard(int card) {
        _hand.Add(card);
    }
}
```

PlayerSelf.cs

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

public class PlayerSelf : PlayerBase //PlayerBaseの設計図を継承
{
    ...
}
```

GameManager.cs

```
PlayerSelf _playerSelf; //自分のカード情報を保持しておく場所
void DealCards() {
    //山札の1つ目の要素と2つ目の要素を自分の手札に加えた
    _playerSelf.RecieveCard(_cards[0]);
    _playerSelf.RecieveCard(_cards[1]);
}
```

ここまで内容を丸写しにしている人は、

```
_playerSelf.RecieveCard(_cards[0]);  
_playerSelf.RecieveCard(_cards[1]);
```

上記内容が「エラー」になるはずです
修正して下さい

修正できた人は教えて下さい(見せてください)
加点します

次はどうする？

山札をシャッフルしたり、
山札は配れば減るはずなので、
山札の一番上のカードは何かを把握できるようにする

山札をシャッフルしてみる

山札のシャッフルとは？

山札のシャッフルとは？

-山札とは0~52までの数字という要素が入った配列

山札のシャッフルとは？

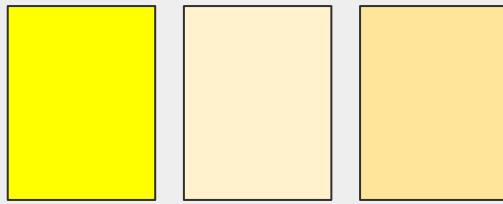
- 山札とは0~52までの数字という要素が入った配列
- この配列の要素の順番を入れ替える

GameManager.cs

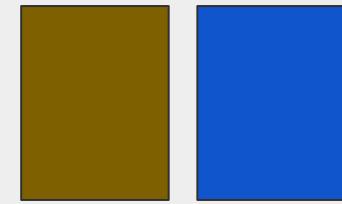
```
int[] _cards; //山札
int _totalNum = 52; //カードの枚数

//初期状態
void InitialCards() {
    _cards = new int[_totalNum]; //52個の要素が入る配列
    for (int i = 0; i < _totalNum; i++) {
        _cards[i] = i; //実際に数字を入れる
    }
    //配列の要素の中身をランダムに入れ替える
    for (int i = 0; i < _totalNum; i++)
    {
        int randomIndex = Random.Range(0, _totalNum);
        int temp = _cards[i]; //i番目のカードを一時保存
        _cards[i] = _cards[randomIndex]; //ランダムな場所の配列の要素をi番目に代入
        _cards[randomIndex] = temp; //ランダムな場所の配列の要素にi番目のカードを代入
    }
}
```

i 番目

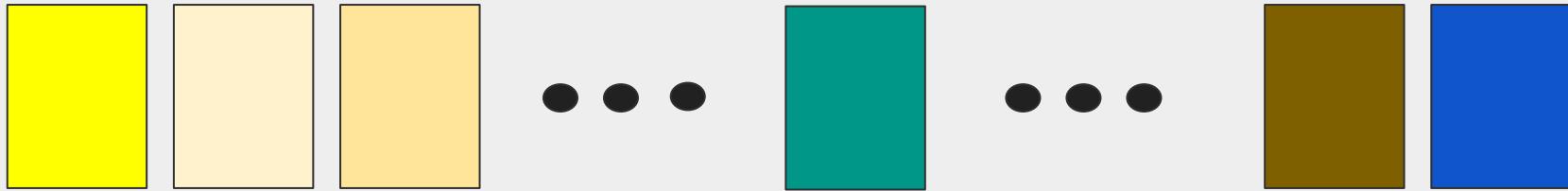


randomIndex 番目



i 番目

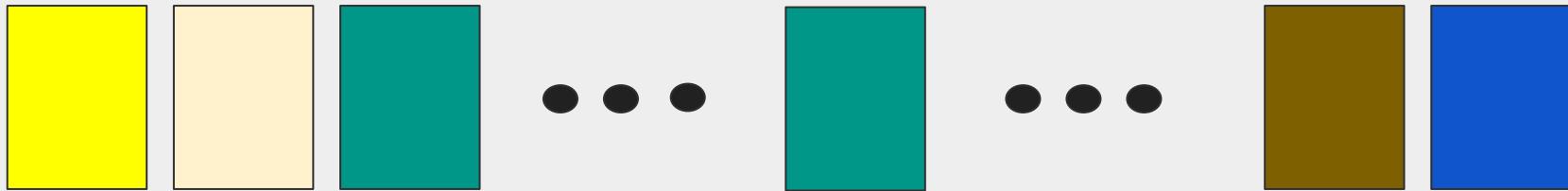
randomIndex 番目



temp

i 番目

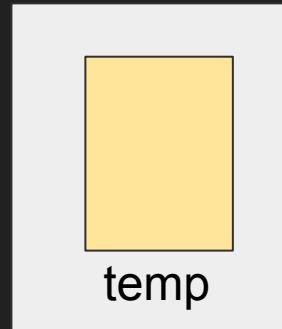
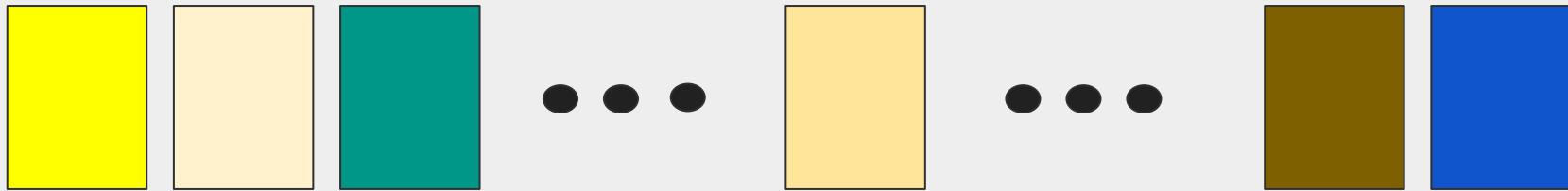
randomIndex 番目



temp

i 番目

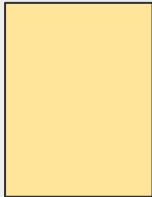
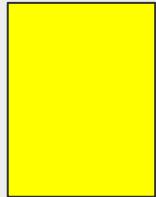
randomIndex 番目



Before

i 番目

randomIndex 番目



• • •



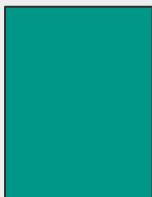
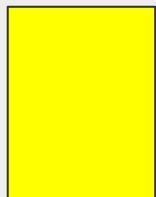
• • •



After

i 番目

randomIndex 番目



• • •



• • •



GameManager.cs

```
int[] _cards; //山札
int _totalNum = 52; //カードの枚数

//初期状態
void InitialCards() {
    _cards = new int[_totalNum]; //52個の要素が入る配列
    for (int i = 0; i < _totalNum; i++) {
        _cards[i] = i; //実際に数字を入れる
    }
    //配列の要素の中身をランダムに入れ替える
    for (int i = 0; i < _totalNum; i++)
    {
        int randomIndex = Random.Range(0, _totalNum);
        int temp = _cards[i]; //i番目のカードを一時保存
        _cards[i] = _cards[randomIndex]; //ランダムな場所の配列の要素をi番目に代入
        _cards[randomIndex] = temp; //ランダムな場所の配列の要素にi番目のカードを代入
    }
}
```

山札のシャッフルは終了
山札は配れば減るはずなので、
山札の一番上のカードは何かを把握できるようにする

GameManager.cs

```
int _cardTop = 0; //カードの一番上要素を入れておく変数

PlayerSelf _playerSelf; //自分のカード情報を保持しておく場所
void DealCards() {
    //山札の1つ目の要素と2つ目の要素を自分の手札に加えた
    _playerSelf.RecieveCard(_cards[0]);
    _cardTop++;

    _playerSelf.RecieveCard(_cards[1]);
    _cardTop++;
    //カードの一番上の要素は2 山札の配列の_cards[2]が一番上
}
```

カードを配ってる初期状態?
自分の番?
敵の番?

などを把握しておくために状態(ステート)を管理できるようにしてみる

GameManager.cs

```
public enum GameState
{
    None,           // 何もない
    Initialize,    // 初期状態
    PlayerTurn,    // 自分のターン
    EnemyTurn,     // 敵のターン
    Result         // 結果表示
}
public GameState State { get; private set; } = GameState.None;
```

GameManager.cs

```
private void Start()
{
    State = GameState.Initialize; //ステートを初期状態にする
    InitialCards(); //山札をシャッフル
    DealCards(); //カードを配る
    State = GameState.PlayerTurn; //自分のターン
}
```

同じように敵にもカードを配つてみる

PlayerEnemy.cs

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

public class PlayerEnemy : PlayerBase
{
    // Start is called before the first frame update
    void Start()
    {
        ...
    }

    // Update is called once per frame
    void Update()
    {
        ...
    }
}
```

GameManager.cs

```
int _cardTop = 0; //カードの一番上要素を入れておく変数

PlayerSelf _playerSelf; //自分のカード情報を保持しておく場所
PlayerEnemy _playerEnemy; //敵のカード情報を保持しておく場所
void DealCards() {
    _playerSelf.RecieveCard(_cards[0]);
    _playerSelf.RecieveCard(_cards[1]);
    _playerEnemy.RecieveCard(_cards[2]);
    _playerEnemy.RecieveCard(_cards[3]);
    _cardTop += 4;
}
```

GameManager.cs

```
int _cardTop = 0; //カードの一番上要素を入れておく変数

[SerializeField]PlayerSelf _playerSelf; //自分のカード情報を保持しておく場所
[SerializeField]PlayerEnemy _playerEnemy; //敵のカード情報を保持しておく場所
void DealCards() {
    _playerSelf.RecieveCard(_cards[0]);
    _playerSelf.RecieveCard(_cards[1]);
    _playerEnemy.RecieveCard(_cards[2]);
    _playerEnemy.RecieveCard(_cards[3]);
    _cardTop +=4;
}
```

GameManager.cs

```
int _cardTop = 0; //カードの一番上要素を入れておく変数

[SerializeField]PlayerSelf _playerSelf; //自分のカード情報を保持しておく場所
[SerializeField]PlayerEnemy _playerEnemy; //敵のカード情報を保持しておく場所
void DealCards() {
    _playerSelf.RecieveCard(_cards[0]);
    _playerSelf.RecieveCard(_cards[1]);
    _playerEnemy.RecieveCard(_cards[2]);
    _playerEnemy.RecieveCard(_cards[3]);
    _cardTop += 4;

    Debug.Log($"自分の手札1 {_cards[0] % 13 + 1}");
    Debug.Log($"自分の手札2 {_cards[1] % 13 + 1}");
    Debug.Log($"敵の手札1 {_cards[2] % 13 + 1}");
    Debug.Log($"敵の手札2 {_cards[3] % 13 + 1}");
}
```

Cut
Copy
Paste
Paste As Child

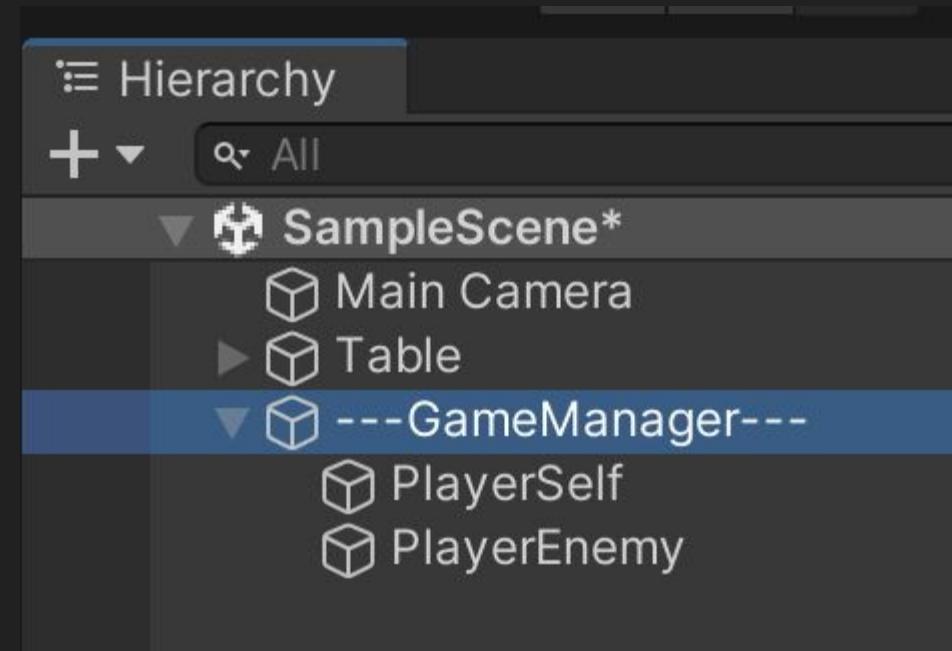
Rename
Duplicate
Delete

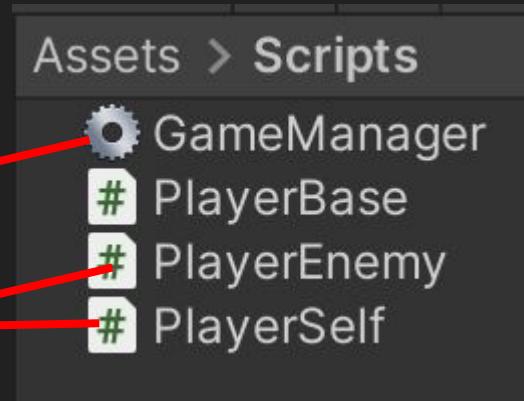
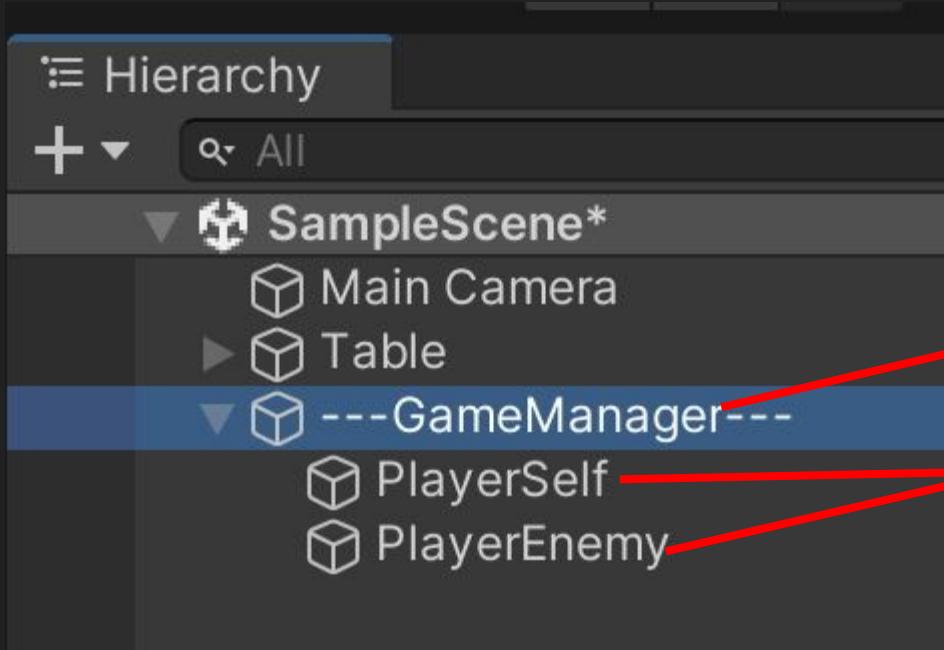
Select Children

Set as Default Parent

Create Empty

2D Object >
3D Object >
Effects >
Light >
Audio >
Video >
UI >
UI Toolkit >
Camera
Visual Scripting Scene Variables





Inspector



---GameManager---



Static

Tag Untagged Layer Default



Transform



Position X 0 Y 0 Z 0

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1



Game Manager (Script)



Script

GameManager

Player Self

PlayerSelf (Player Sel)

Player Enemy

PlayerEnemy (Player

Add Component



手札の計算

手札の計算とは？

手札の計算とは？

-手札 = 変数「_hand」の合計値を求めたい

PlayerBase.cs

```
using UnityEngine;

public class PlayerBase : MonoBehaviour
{
    //手札のリスト
    List<int> _hand = new List<int>();

    //手札にカードを加える
    public void RecieveCard(int card) {
        _hand.Add(card);
    }

    public int CalcPoint() {
        int totalPoint = 0;

        foreach (int card in _hand) {
            int point = card % 13 + 1;
            totalPoint += point;
        }
        return totalPoint;
    }
}
```

ブラックジャックにおける手札とは？

ブラックジャックにおけるカードの意味とは？

ブラックジャックにおけるカードの意味

- 絵札(J・Q・K)は「10」とみなす
- エース(A)は「1」もしくは「11」としてみなす

ブラックジャックにおけるカードの意味

- 絵札(J・Q・K)は「10」とみなす
- エース(A)は「1」もしくは「11」としてみなす

PlayerBase.cs

```
public int CalcPoint() {
    int totalPoint = 0;

    //配列番号0~12 → 実際の絵柄は1~13
    foreach (int card in _hand) {
        int point = card % 13 + 1;
        point = Mathf.Min(point, 10); //10~13を10に丸める
        totalPoint += point;
    }
    return totalPoint;
}
```

ブラックジャックにおけるカードの意味

- 絵札(J・Q・K)は「10」とみなす
- エース(A)は「1」もしくは「11」としてみなす

『エース(A)は「1」もしくは「11」としてみなす』
これをどうやって実装する？

『エース(A)は「1」もしくは「11」としてみなす』

1にしたい時はどういう時？

11にしたいときはどういう時？

『エース(A)は「1」もしくは「11」としてみなす』

11にしたいときはどういう時？

→ 今そのまま計算すると「A」は「1」とみなして計算することしかできない

現在の手札の合計が「11」の時 かつ 手札に「A」がある時
「A」を「11」として計算すれば、手札の合計は「21」になる

『エース(A)は「1」もしくは「11」としてみなす』

11にしたいときはどういう時？

→ 今そのまま計算すると「A」は「1」とみなして計算することしかできない

現在の手札の合計が「11」の時 かつ 手札に「A」がある時
「A」を「11」として計算すれば、手札の合計は「21」になる

現在の手札の合計が「11」以下の時 かつ 手札に「A」がある時
「A」を「11」として計算すれば、手札の合計は「21」以下で収まる

PlayerBase.cs

```
public int CalcPoint() {
    int totalPoint = 0;
    bool hasAce = false;
    //配列番号0~12 → 実際の絵柄は1~13
    foreach (int card in _hand) {
        int point = card % 13 + 1;
        point = Mathf.Min(point, 10); //10~13を10に丸める
        totalPoint += point;
        if (point == 1) {
            hasAce = true;
        }
    }
    //エースを持ってて、かつトータルのポイントが11以下ならエースを(1はすでに足してあるので)10として計算
    if (hasAce && totalPoint <= 11)
    {
        totalPoint += 10;
    }
    //22以上になると バースト ポイントを0として計算
    if (totalPoint > 21)
    {
        totalPoint = 0;
    }
    return totalPoint;
}
```

「バーストしたら(22以上になつたら)合計値0」
として考えるプログラムも追加した

自分と敵の計算結果を出力する

自分と敵の計算結果を出力する
-どこで何する？(どこのスクリプトファイルでどういう処理をする？)

GameManager.cs

```
//結果出力
void ExecResult() {
    int pointSelf = _playerSelf.CalcPoint();
    int pointEnemy = _playerEnemy.CalcPoint();
    Debug.Log($"自身の点数:{pointSelf} , 敵の点数:{pointEnemy}");
}
```

GameManager.cs

```
private void Start()
{
    State = GameState.Initialize; //ステートを初期状態にする
    InitialCards(); //山札をシャッフル
    DealCards(); //カードを配る
    State = GameState.PlayerTurn; //自分のターン
    ExecResult(); //結果を出力
}

//結果出力
void ExecResult() {
    int pointSelf = _playerSelf.CalcPoint();
    int pointEnemy = _playerEnemy.CalcPoint();
    Debug.Log($"自身の点数:{pointSelf} , 敵の点数:{pointEnemy} ");
}
```



Console

Console



Clear ▾

Collapse

Error Pause

Editor ▾



! 5

! 0

! 0



[11:27:16] 自分の手札1 3

UnityEngine.Debug:Log (object)



[11:27:16] 自分の手札2 1

UnityEngine.Debug:Log (object)



[11:27:16] 敵の手札1 9

UnityEngine.Debug:Log (object)



[11:27:16] 敵の手札2 12

UnityEngine.Debug:Log (object)



[11:27:16] 自身の点数:14 , 敵の点数:19

UnityEngine.Debug:Log (object)

ヒット・スタンド

「ヒット」と「スタンド」とは？

「ヒット」と「スタンド」とは？

- 「ヒット」もう一枚、山札からカードを取ってくる
- 「スタンド」手札を維持

「ヒット」と「スタンド」とは？

- 「ヒット」もう一枚、山札からカードを取ってくる
- 「スタンド」手札を維持

-「ヒット」もう一枚、山札からカードを取ってくる
-「ヒット」ボタンを押せば、山札からカードを取ってくる
-山札からカードを取ってくるとは？

- 「ヒット」もう一枚、山札からカードを取ってくる
- 「ヒット」ボタンを押せば、山札からカードを取ってくる
 - 山札からカードを取ってくるとは、
山札の一番上は変数として、
配列の何番目なのか、整数で所持していたので
「ヒット」処理を行ったら、変数の値を1つ大きくする

GameManager.cs

ちなみに現在の「GameManager.cs」の「Startメソッド」の中身は

```
private void Start()
{
    State = GameState.Initialize;//ステートを初期状態にする
    InitialCards();//山札をシャッフル
    DealCards();//カードを配る
    State = GameState.PlayerTurn;//自分のターン
    ExecResult();//結果を出力
}
```

GameManager.cs

```
//ヒットボタンを押した時の処理
public void OnHitClicked()
{
    Debug.Log("Hit!");
    if (State == GameState.PlayerTurn)
    {
        //山札の一番上からカードを引く
        _playerSelf.RecieveCard(_cards[_cardTop]);
        //カードの一番上の要素番号を更新
        _cardTop++;
    }
}
```

Select Children

Set as Default Parent

Create Empty

2D Object >

3D Object >

Effects >

Light >

Audio >

Video >

UI >

UI Toolkit >

Volume >

Rendering >

Camera

Visual Scripting Scene Variables

Image

Text - TextMeshPro

Raw Image

Panel

Toggle

Slider

Scrollbar

Scroll View

Button - TextMeshPro

Dropdown - TextMeshPro

Input Field - TextMeshPro

Canvas

Event System

Legacy >



TMP Importer

TMP Importer

⋮

TMP Essentials

This appears to be the first time you access TextMesh Pro, as such we need to add resources to your project that are essential for using TextMesh Pro. These new resources will be placed at the root of your project in the "TextMesh Pro" folder.

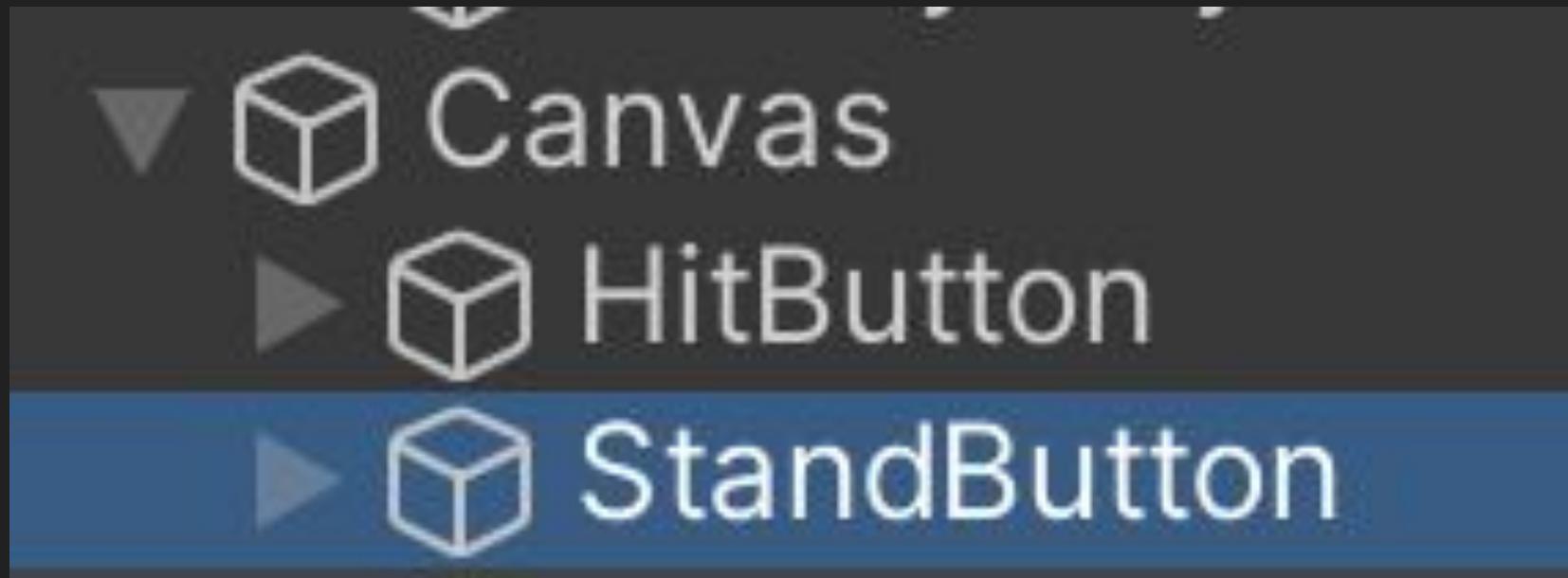
[Import TMP Essentials](#)

TMP Examples & Extras

The Examples & Extras package contains addition resources and examples that will make discovering and learning about TextMesh Pro's powerful features easier. These additional resources will be placed in the same folder as the TMP essential resources.

[Import TMP Examples & Extras](#)

Canvas内にヒットボタン・スタンドボタン設置



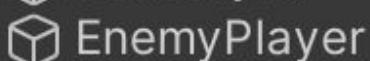
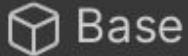
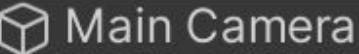
Hierarchy



All



Game



Text (TMP)

Static

Tag Untagged

Layer UI



Rect Transform

⋮

stretch

Left

Top

Pos Z

0

0

0

Right

Bottom

0

0

R

▶ Anchors

Pivot

X 0.5

Y 0.5

Rotation

X 0

Y 0

Z 0

Scale



X 1

Y 1

Z 1



Canvas Renderer

⋮



TextMeshPro - Text (UI)

⋮

Text Input

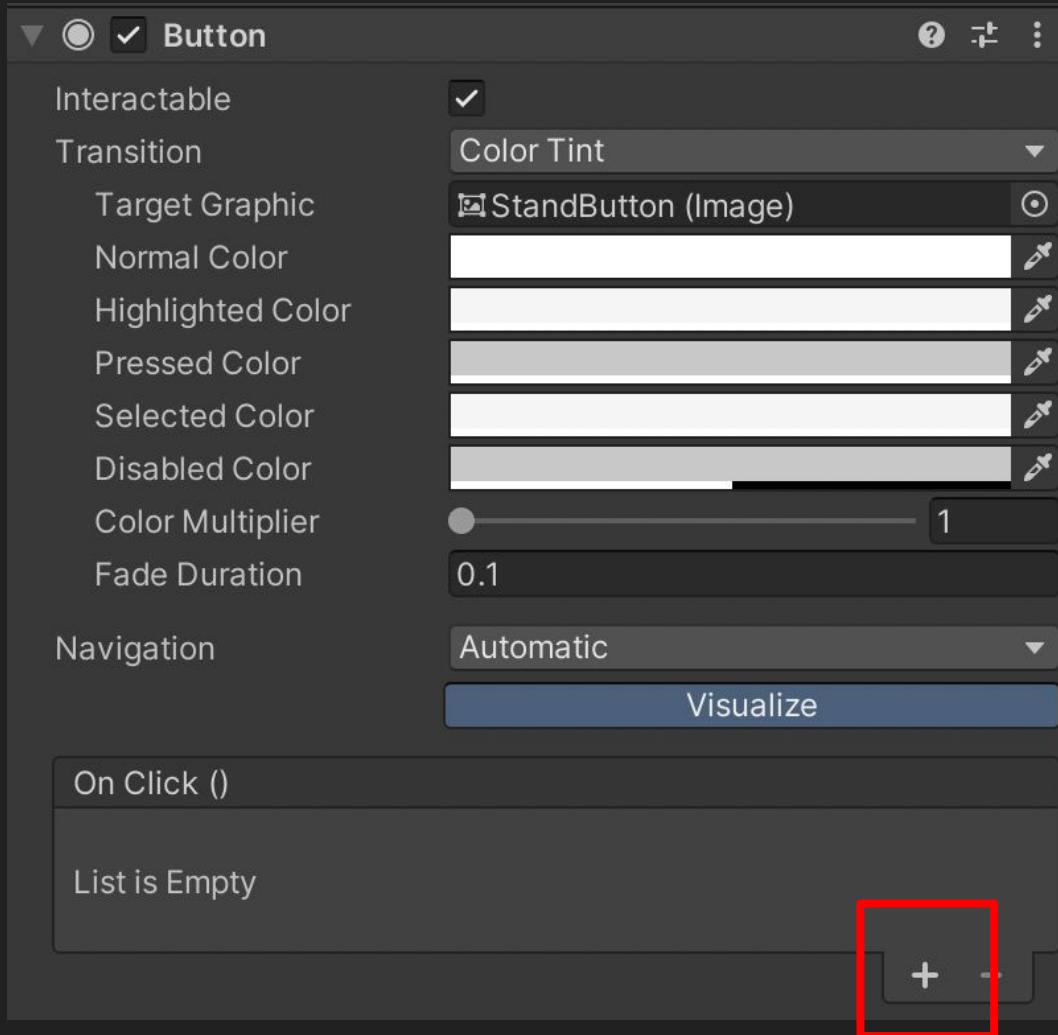
Enable RTL Editor

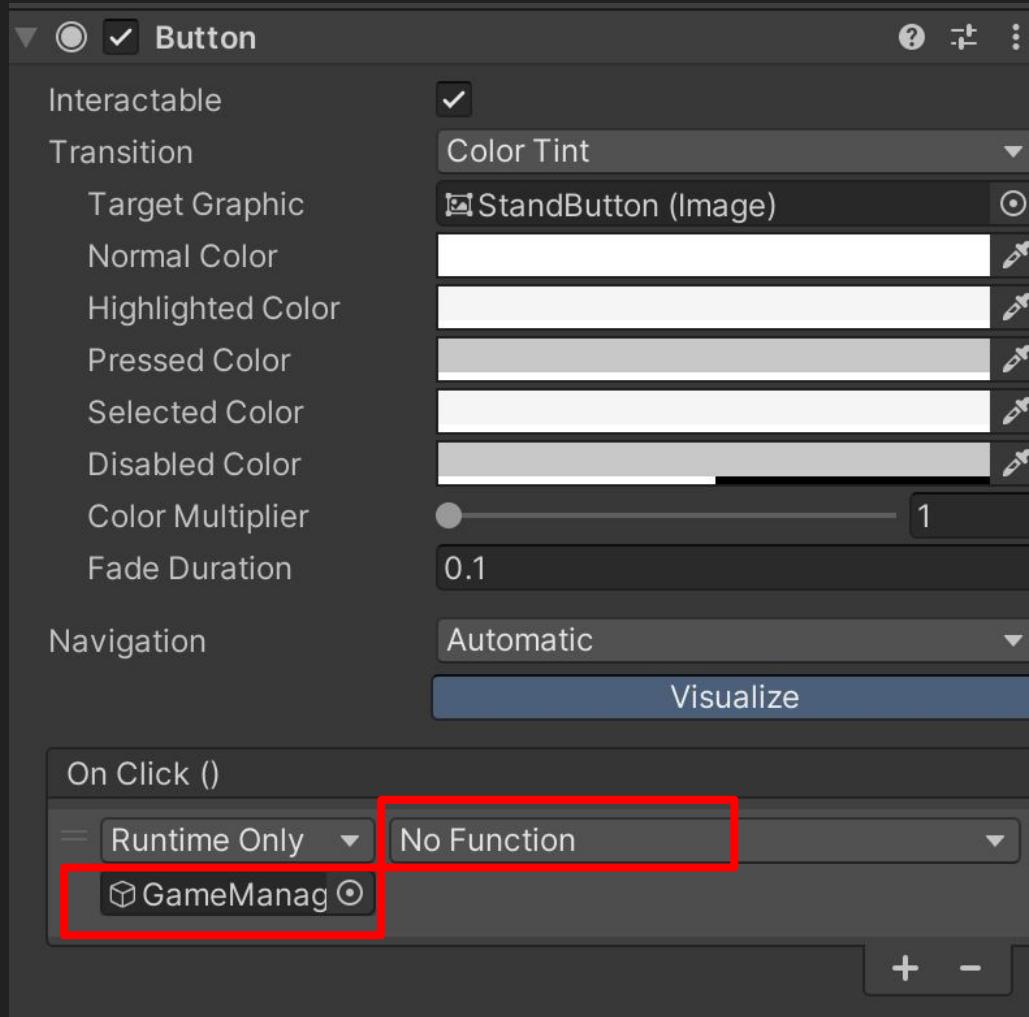
Hit

Text Style

Normal

▼





bool enabled

string name

bool runInEditMode

string tag

bool useGUILayout

BroadcastMessage (string)

CancelInvoke (string)

CancelInvoke ()

OnHitClicked ()

OnStandClicked ()

SendMessage (string)

SendMessageUpwards (string)

StopAllCoroutines ()

StopCoroutine (string)

0.1

Automatic

Visualize

No Function

✓ No Function

GameObject >

Transform >

GameManager >

▼  Canvas

- ▶  HitButton
- ▶  StandButton

▼  Canvas

Render Mode

Pixel Perfect

Render Camera

 A Screen Space Canvas with no specified camera acts like an Overlay Canvas.

Order in Layer

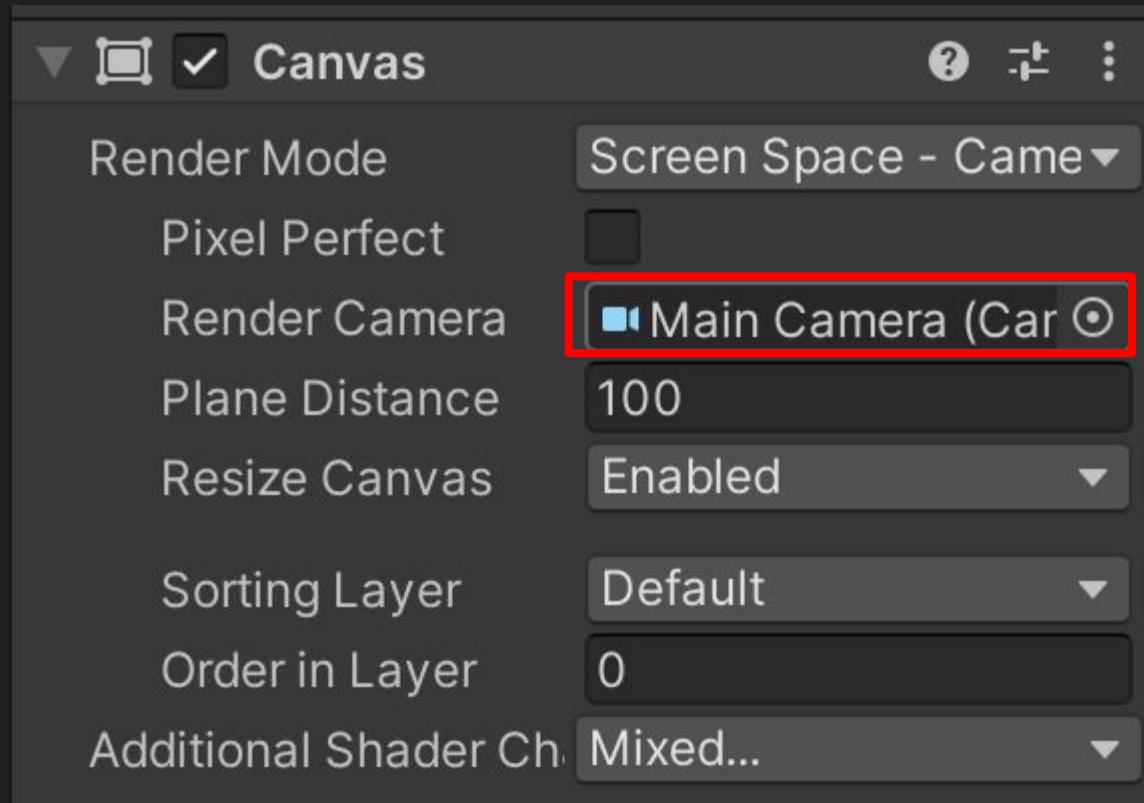
Additional Shader Ch. ▾

Screen Space - Overlay

Screen Space - Camera

World Space

▼  Canvas
►  HitButton
►  StandButton





Console

Console



Clear ▾

Collapse

Error Pause

Editor ▾



! 6

! 0

! 0



[11:59:00] 自分の手札1 3
UnityEngine.Debug:Log (object)



[11:59:00] 自分の手札2 6
UnityEngine.Debug:Log (object)



[11:59:00] 敵の手札1 1
UnityEngine.Debug:Log (object)



[11:59:00] 敵の手札2 2
UnityEngine.Debug:Log (object)



[11:59:00] 自身の点数:9 , 敵の点数:13
UnityEngine.Debug:Log (object)



[11:59:10] Hit!
UnityEngine.Debug:Log (object)

全てStart関数の中で処理を行っているので、
Update関数にて処理を行うプログラムに変更する
(Start関数・Update関数を変更)

GameManager.cs

```
//実際のゲームロジック実装
private void Update()
{
    switch (State) {
        case GameState.None:
        case GameState.Initialize:
        case GameState.PlayerTurn:
        case GameState.EnemyTurn:
        case GameState.Result:
            break;
    }
}
```

GameManager.cs

```
private void Start()
{
    State = GameState.Initialize;//ステートを初期状態にする
    //InitialCards();//山札をシャッフル
    //DealCards();//カードを配る
    //State = GameState.PlayerTurn;//自分のターン
    //ExecResult();//結果を出力
}
```

GameManager.cs

```
private void Update()
{
    switch (State) {
        case GameState.None:
        case GameState.Initialize:
            InitialCards(); //山札をシャッフル
            DealCards(); //カードを配る
            State = GameState.PlayerTurn;
            break;
        case GameState.PlayerTurn:
        case GameState.EnemyTurn:
        case GameState.Result:
            ExecResult(); //結果を出力
            break;
    }
}
```

GameManager.cs

```
//クリックしたかどうかの判定  
bool _hitClicked = false;  
  
//ヒットボタンを押した時の処理  
public void OnHitClicked()  
{  
    Debug.Log("Hit!");  
    _hitClicked = true;  
}
```

GameManager.cs

```
case GameState.PlayerTurn:  
    if (_hitClicked)  
    {  
        _hitClicked = false;  
        //山札の一番上からカードを引く  
        _playerSelf.RecieveCard(_cards[_cardTop]);  
        //カードの一番上の要素番号を更新  
        _cardTop++;  
        State = GameState.EnemyTurn;  
    }  
    break;
```

GameManager.cs

「スタンド」の処理はどうする？

GameManager.cs

```
//クリックしたかどうかの判定  
bool _hitClicked = false;  
bool _standClicked = false;  
  
//ヒットボタンを押した時の処理  
public void OnHitClicked()  
{  
    Debug.Log("Hit!");  
    _hitClicked = true;  
}  
  
//スタンド(ステイ)ボタンを押した時の処理  
public void OnStandClicked()  
{  
    Debug.Log("Stand!");  
    _standClicked = true;  
}
```

GameManager.cs

```
case GameState.PlayerTurn:  
    if (_hitClicked)  
    {  
        _hitClicked = false;  
        //山札の一番上からカードを引く  
        _playerSelf.RecieveCard(_cards[_cardTop]);  
        //カードの一番上の要素番号を更新  
        _cardTop++;  
        State = GameState.EnemyTurn;  
    }  
    if (_standClicked)  
    {  
        _standClicked = false;  
        State = GameState.EnemyTurn;  
    }  
break;
```

「スタンド」のボタン設定も行って、実行してみる

敵のターン

「敵のターン」の処理はどうする？

「敵のターン」の処理はどうする？

-敵のターンの「ヒット」「スタンド」はどう処理する？

「敵のターン」の処理はどうする？

-敵のターンの「ヒット」「スタンド」はどう処理する？

-ここでは簡単にある一定の値を越えなければ「ヒット」

越えれば「スタンド」行為を行ってもらう

ある一定の値を越えないなら「ヒット」

ある一定の値を越えないなら「ヒット」

- 「敵のターン」のプログラム

- 敵の行動のスクリプトに書き込む

- 手札の合計値を計算し、その後「16」点を超えないなら山札を引く

GameManager.cs

```
case GameState.EnemyTurn:  
    //ヒットするかどうか判断  
    //「計算結果が0より上」かつ「16より下」ならtrue (合計値が1~15の時、実行)  
    //山札の一番上からカードを引く  
    //忘れずにカードの一番上を表す変数を足す  
    //スタンドするかどうか判断  
    //計算結果が0より上かつ16より下なら(1~15なら)      false ,  
    //0 (22以上はバーストで0として処理) もしくは16以上ならtrue  
    //結果を出力
```

PlayerEnemy.cs

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

public class PlayerEnemy : PlayerBase
{
    //ヒット処理
    public bool HitAI() {
        int point = CalcPoint();
        return 0 < point && point < 16; //1~15かどうか
    }

    //スタンダード処理
    public bool StandAI()
    {
        return !HitAI(); //0~15ならfalse, 0(バースト)もしくは16以上ならtrue
    }
}
```

GameManager.cs

```
case GameState.EnemyTurn:  
    //ヒットするかどうか判断  
    //「計算結果が0より上」かつ「16より下」ならtrue (合計値が1~15の時、実行)  
    //山札の一番上からカードを引く  
    //忘れずにカードの一番上を表す変数を足す  
    //スタンドするかどうか判断  
    //計算結果が0より上かつ16より下なら(1~15なら)      false ,  
    //0 (22以上はバーストで0として処理) もしくは16以上ならtrue  
    //結果を出力
```

GameManager.cs

```
case GameState.EnemyTurn:  
    //ヒットするかどうか判断  
    //「計算結果が0より上」かつ「16より下」ならtrue (合計値が1~15の時、実行)  
    if (_playerEnemy.HitAI()) {  
        //山札の一番上からカードを引く  
        _playerEnemy.RecieveCard(_cards[_cardTop]);  
        //忘れずにカードの一番上を表す変数を足す  
        _cardTop++;  
    }  
    //スタンドするかどうか判断  
    //計算結果が0より上かつ16より下なら(1~15なら)      false ,  
    //0 (22以上はバーストで0として処理) もしくは16以上ならtrue  
    else if (_playerEnemy.StandAI()) {  
        ExecResult(); //結果を出力  
    }  
    break;
```

リトライ

リトライとは？もう一度ゲームをやるとは？

GameManager.cs

```
//クリックしたかどうかの判定
bool _hitClicked = false;
bool _standClicked = false;
bool _retryClicked = false;

//リトライボタンを押した時の処理
public void OnRetryClicked()
{
    Debug.Log("Retry!");
    _retryClicked = true;
}
```

GameManager.cs

```
case GameState.EnemyTurn:  
    //ヒットするかどうか判断  
    //「計算結果が0より上」かつ「16より下」ならtrue (合計値が1~15の時、実行)  
    if (_playerEnemy.HitAI()) {  
        //山札の一番上からカードを引く  
        _playerEnemy.RecieveCard(_cards[_cardTop]);  
        //忘れずにカードの一番上を表す変数を足す  
        _cardTop++;  
    }  
    //スタンドするかどうか判断  
    //計算結果が0より上かつ16より下なら(1~15なら)      false ,  
    //0 (22以上はバーストで0として処理) もしくは16以上ならtrue  
    else if (_playerEnemy.StandAI()) {  
        ExecResult(); //結果を出力  
        State = GameState.Result;  
    }  
    break;
```

GameManager.cs

```
case GameState.Result:  
    if (_retryClicked)  
    {  
        _retryClicked = false; //リトライボタンの状態を初期状態に戻す  
        State = GameState.Initialize; //ステートを初期化状態にする  
    }  
    break;
```

GameManager.cs

```
case GameState.Initialize:  
    InitialCards(); //山札をシャッフル  
    //自分の手札を初期状態に  
    //敵の手札を初期状態に  
    DealCards(); //カードを配る  
    State = GameState.PlayerTurn;  
    break;
```

PlayerBase.cs

```
//手札を初期化
public void Initialize()
{
    _hand.Clear();
}
```

GameManager.cs

```
case GameState.Initialize:  
    InitialCards(); //山札をシャッフル  
    //自分の手札を初期状態に  
    //敵の手札を初期状態に  
    DealCards(); //カードを配る  
    State = GameState.PlayerTurn;  
    break;
```

GameManager.cs

```
case GameState.Initialize:  
    InitialCards(); //山札をシャッフル  
    _playerSelf.Initialize(); //自分の手札を初期状態に  
    _playerEnemy.Initialize(); //敵の手札を初期状態に  
    DealCards(); //カードを配る  
    State = GameState.PlayerTurn;  
    break;
```

UI(ボタン)追加

▼  RetryButton
  Text (TMP)

▼  TextMeshPro - Text (UI)   

Text Input Enable RTL Editor

Retry

▼ ○ ✓ **Button** ? ⌂ :

Interactable

Transition

Target Graphic RetryButton (Image) ⌂

Normal Color

Highlighted Color

Pressed Color

Selected Color

Disabled Color

Color Multiplier

Fade Duration 0.1

Navigation Automatic ⌂

Visualize

On Click ()

= Runtime Only ▾ GameManager.OnRetryClicked ▾

GameManag ⌂

+ -

実行してみる

```
! [13:03:03] Retry!
UnityEngine.Debug:Log (object)
! [13:03:03] 自分の手札1 12
UnityEngine.Debug:Log (object)
! [13:03:03] 自分の手札2 7
UnityEngine.Debug:Log (object)
! [13:03:03] 敵の手札1 5
UnityEngine.Debug:Log (object)
! [13:03:03] 敵の手札2 5
UnityEngine.Debug:Log (object)
```

トランプの描画

Free Playing Cards Pack

Game Asset Studio

Version 1.0 - October 23, 2019

[asset store](#)

[View in the Asset Store](#) • [Publisher Website](#) • [Publisher Support](#)

Game Asset Studio introduces a set of 3D models and 2D illustrations of playing cards that help you create your playing card games!

With decks of cards from Aces to Kings plus Jokers, this is a must-have asset that

[More...](#)

Images & Videos



[View images & videos on Asset Store](#)

Package Size

Size: 113.3 MB (Number of files: 579)

Supported Unity Versions

2019.2.3 or higher

Purchased Date

July 11, 2022

Release Details

1.0 (Current) - released on October 23,

[More...](#)

Original - released on October 23, 2019

Assigned Labels

(None)

Import

Re-Download

Assets > Playing Cards > Image > PlayingCards

 BackColor_Black

 BackColor_Blue

 BackColor_Red

 Club01

 Club02

 Club03

 Club04

 Club05

 Club06

 Club07

 Club08

 Club09

 Club10

 Club11

 Club12

 Club13

 Diamond01

 Diamond02

 Diamond03

Assets > Playing Cards > Image > PlayingCards

BackColor_Black

BackColor_Blue

BackColor_Red

Club01

Club02

Club03

Club04

Club05

Club06

Club07

Club08

Club09

Club10

Club11

Club12

Club13

Diamond01

Diamond02

Diamond03

i Inspector



Back Color_Black (Texture 2D) In

Texture Type

Sprite (2D and UI)

Texture Shape

2D

Sprite Mode

Single

Packing Tag

Pixels Per Unit

100

Mesh Type

Tight

Extrude Edges

0

Pivot

Center

Generate Physics :

Spr



Unapplied import settings

Unapplied import settings for '57' files

Apply

Revert

Cancel

Inspector Open

57 Texture 2Ds Import Settings

Texture Type: Sprite (2D and UI) ▼

Texture Shape: 2D ▼

Sprite Mode: Single ▼

Packing Tag:

Pixels Per Unit: 100

Mesh Type: Tight ▼

Extrude Edges: 1

Pivot: Center ▼

Generate Physics S:

Sprite Editor

Advanced

sRGB (Color Texture):

Alpha Source: Input Texture Alpha ▼

Alpha Is Transparent:

Read/Write:

Generate Mipmaps:

Wrap Mode: Clamp ▼

Filter Mode: Bilinear ▼

Aniso Level: 1

Default PC iOS

Override For Windows, Mac, Linux

Max Size: 512 ▼

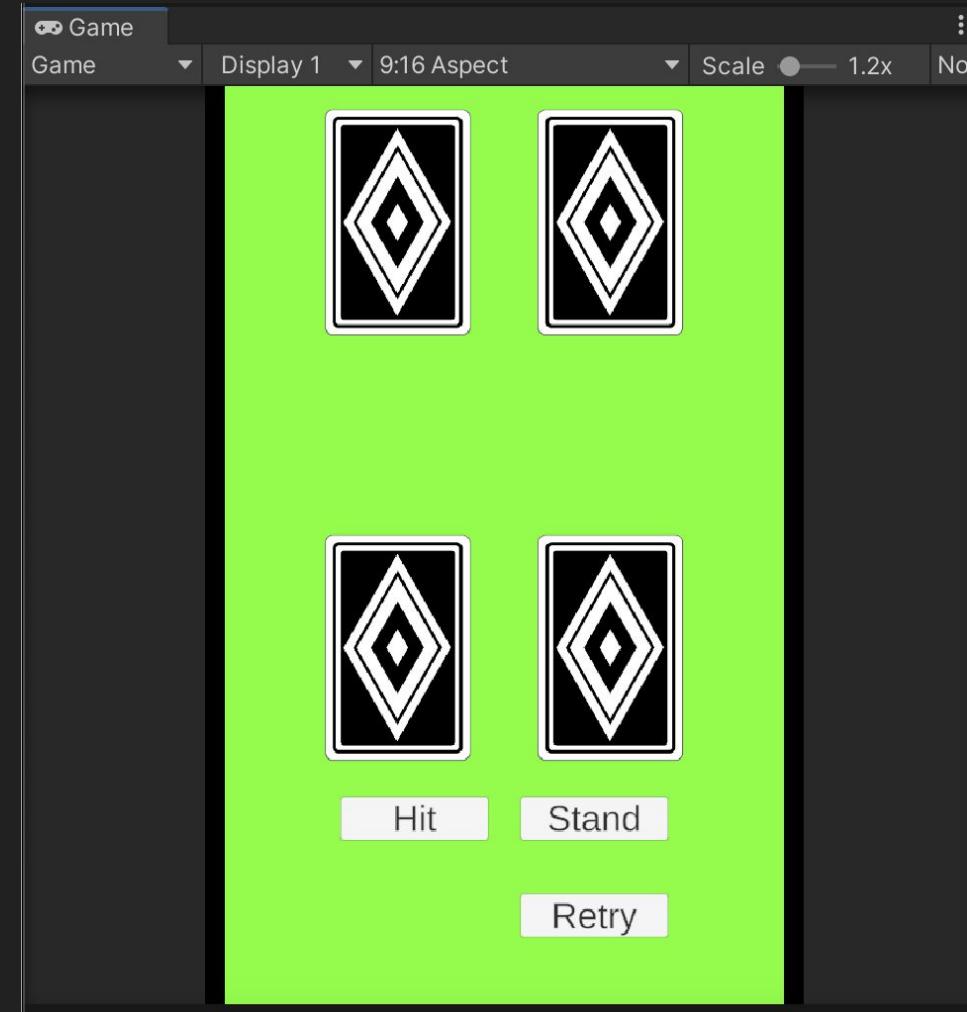
Resize Algorithm: Mitchell ▼

Format: RGBA Compressed DXT5|BC3 ▼

Revert Apply



BackColor_Black



Inspector

BackColor_Blk Static

Tag Untagged Layer Default

Transform

Position	X -1	Y 3	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 0.11	Y 0.11	Z 1

Sprite Renderer

Sprite	<input checked="" type="radio"/> BackColor_Blk
Color	[Color Box] <input type="button" value="Brush"/>
Flip	<input type="checkbox"/> X <input type="checkbox"/> Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	<input checked="" type="radio"/> Sprites-Default

Additional Settings

Sorting Layer	Default
Order in Layer	0

どこからでもアクセスできるように
シングルトンなクラスを用意

SingletonMonobehaviour.cs

```
using UnityEngine;
using System;

public abstract class SingletonMonoBehaviour<T> : MonoBehaviour where T : MonoBehaviour
{
}
```

SingletonMonoBehaviour.cs

```
// 「同一のゲームオブジェクトは常に 1 つだけにする」
using UnityEngine;
using System;

public abstract class SingletonMonoBehaviour<T> : MonoBehaviour where T : MonoBehaviour
{
    private static T instance;
    public static T Instance
    {
        get
        {
            if (instance == null)//インスタンスがまだ作られていない
            {
                Type t = typeof(T);
                instance = (T)FindObjectOfType(t); //全オブジェクトを探索,名前が一致するクラスがあった場合は取得する
                if (instance == null)
                {
                    Debug.LogError(t + " をアタッチしているGameObjectはありません");
                }
            }
            return instance;
        }
    }
}
```

```
virtual protected void Awake()
{
    CheckInstance(); // 他のゲームオブジェクトにスクリプトがアタッチされているか調べる
    DontDestroyOnLoad(this.gameObject);
}

protected bool CheckInstance()
{
    if (instance == null) // インスタンスがまだ作られていない
    {
        instance = this as T;
        return true;
    }
    else if (Instance == this)
    {
        return true;
    }
    Destroy(this.gameObject); // 既に同一名のクラスが存在していた場合は破棄する。
    return false;
}
```

「GameManager.cs」が1つしか存在しないようにする

GameManager.cs

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

public class GameManager : SingletonMonoBehaviour<GameManager>
{
```

下準備終了

カードをプログラムから描画できるようにしていく

カードの全種類のありかをGameManager.csに覚えさせておく

GameManager.cs

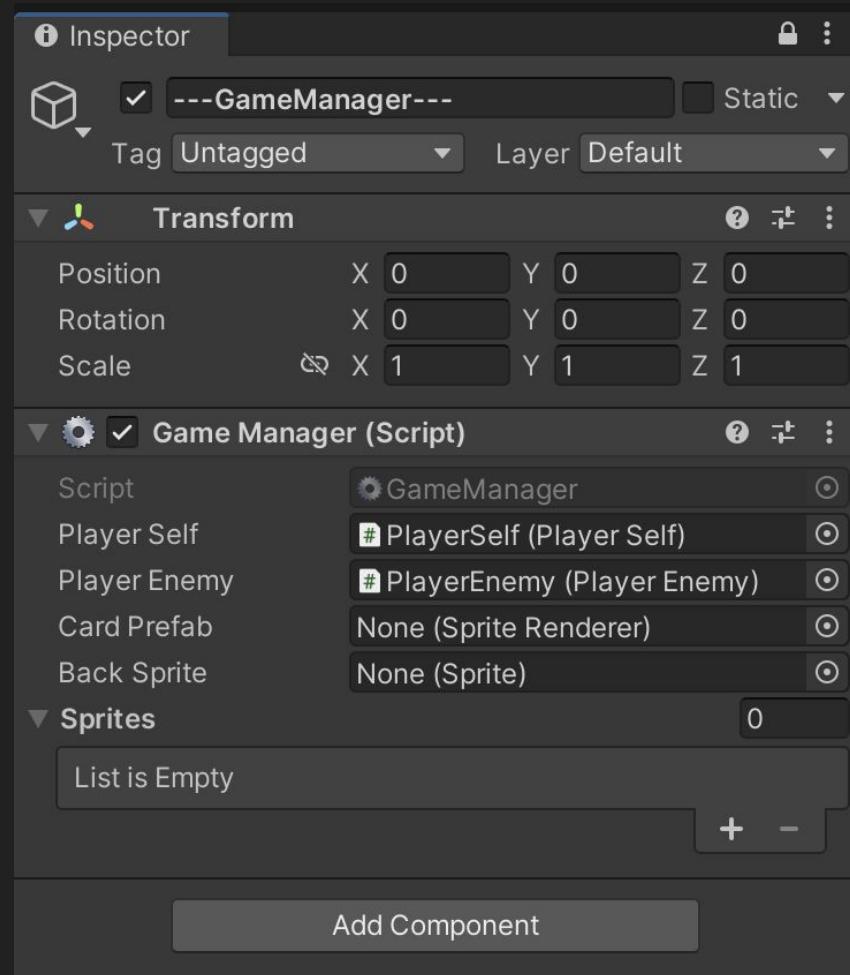
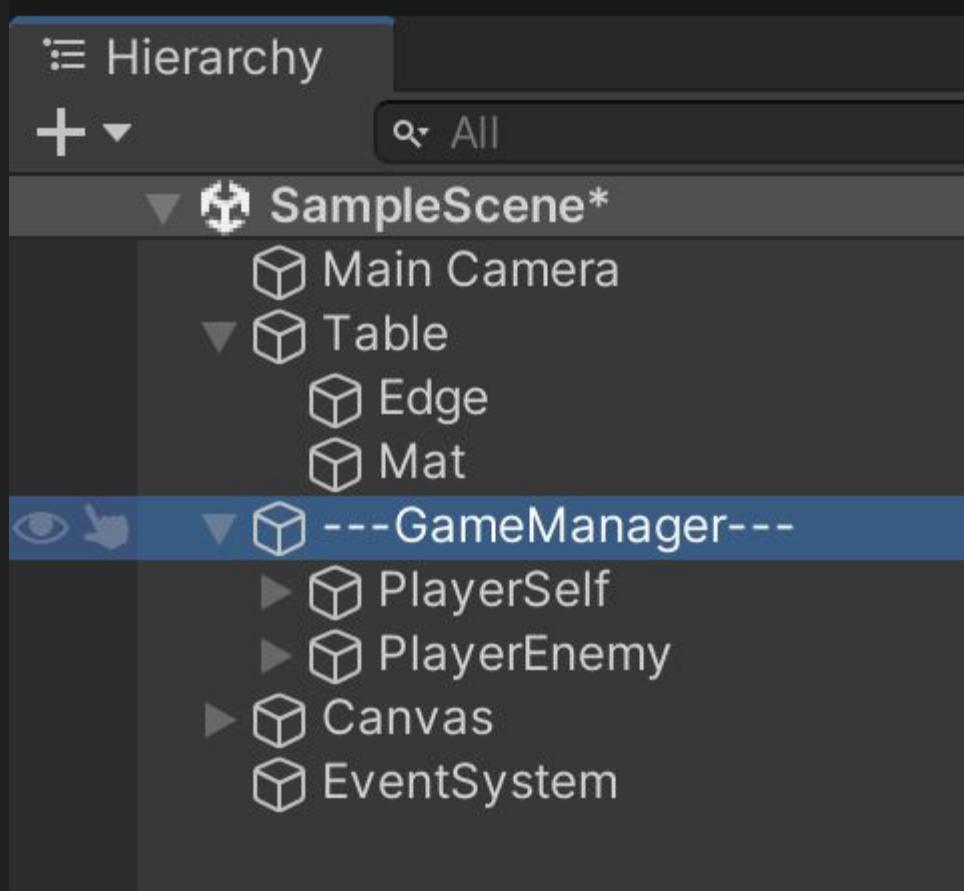
```
//カードを画面表示するためのPrefab
[SerializeField] SpriteRenderer _cardPrefab;
public SpriteRenderer CardPrefab => _cardPrefab; //外から参照できるようにする

//裏面の絵柄
[SerializeField] Sprite _backSprite;
public Sprite BackSprite => _backSprite;

/// <summary>
/// 0~12 : スペード
/// 13~25 : ハート
/// 26~38 : ダイア
/// 39~51 : クラブ
/// </summary>
//表面の絵柄
[SerializeField] Sprite[] _sprites;
public Sprite[] Sprites => _sprites;
```

カードの全種類のありかをGameManager.csに設定





Assets > Playing Cards > Image > PlayingCards

- ▶  Heart08
- ▶  Heart09
- ▶  Heart10
- ▶  Heart11
- ▶  Heart12
- ▶  Heart13
- ▶  Joker_Color
- ▶  Joker_Monochrome
- ▶  Spade01
- ▶  Spade02
- ▶  Spade03
- ▶  Spade04
- ▶  Spade05
- ▶  Spade06
- ▶  Spade07
- ▶  Spade08
- ▶  Spade09
- ▶  Spade10
- ▶  Spade11
- ▶  Spade12
- ▶  Spade13

Inspector

 ---GameManager--- Static
Tag Untagged Layer Default

Transform

Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Game Manager (Script)

Script GameManager
Player Self # PlayerSelf (Player Self)
Player Enemy # PlayerEnemy (Player Enemy)
Card Prefab None (Sprite Renderer)
Back Sprite None (Sprite)

Sprites

List is Empty

+ -

Add Component

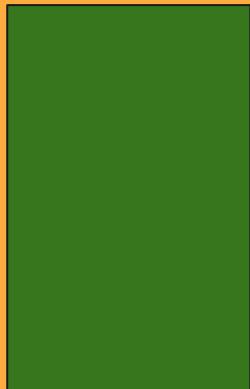
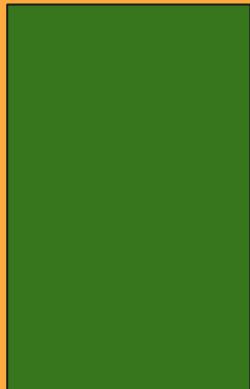
```
/// <summary>
/// 0~12 : スペード
/// 13~25 : ハート
/// 26~38 : ダイア
/// 39~51 : クラブ
/// </summary>
```

Assets > Playing Cards > Image > PlayingCards

- Diamond11
- Diamond12
- Diamond13
- Heart01
- Heart02
- Heart03
- Heart04
- Heart05
- Heart06
- Heart07
- Heart08
- Heart09
- Heart10
- Heart11
- Heart12
- Heart13

Sprites		52
=	Element	Spade01
=	Element	Spade02
=	Element	Spade03
=	Element	Spade04
=	Element	Spade05
=	Element	Spade06
=	Element	Spade07
=	Element	Spade08
=	Element	Spade09
=	Element	Spade10
=	Element	Spade11

カードを表示する
ためのプレファブを
用意し、後で
画像を差し替える



カードの全種類のありかをGameManager.csに覚えさせておく

Player・Enemyから対応するカードをGameManager.csから呼び出し表示

Player・Enemyから対応するカードをGameManager.csから呼び出し表示

Playerにて
描画するためのリストを用意
(今回は初期化処理についても書く)

PlayerBase.cs

```
//自クラス及び派生クラスにおいて参照可能 //実際に描画するものを入れておくリスト
protected List<SpriteRenderer> _drawCardsObjs = new List<SpriteRenderer>();

//手札を初期化
public void Initialize()
{
    _hand.Clear(); //手札リストを初期化
    //実際に描画するためのリストを初期化
    foreach (SpriteRenderer drawCardsObj in _drawCardsObjs) {
        Destroy(drawCardsObj.gameObject);
    }
    _drawCardsObjs.Clear(); //リスト内を綺麗にしておく
}
```

PlayerBase.cs

カードを受け取った時の処理として、カード描画まで行う。
そのため、カード受け取り時の処理を上書きできるようにする。

```
//カードを受け取り手札に加える
public virtual void RecieveCard(int card) {
    _hand.Add(card);
}
```

PlayerSelf

PlayerSelf.cs

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

public class PlayerSelf : PlayerBase //PlayerBaseの設計図を継承
{
    public override void RecieveCard(int card)
    {
        base.RecieveCard(card);
        Debug.Log($"自分 Recieve : {card % 13}");

        //GameManagerを参照
        GameManager game = GameManager.Instance;
        //参照先のPrefab生成
        var cardObj = Instantiate(game.CardPrefab, transform);
        //カードのゲームオブジェクトのspriteを該当する絵柄に変える
        cardObj.sprite = game.Sprites[card];
    }
}
```

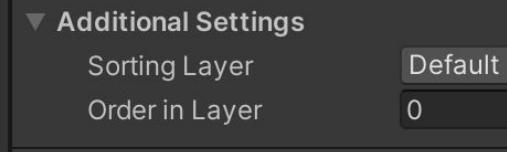
Playerのカードの描画位置のプログラム

PlayerSelf.cs

```
public override void RecieveCard(int card)
{
    base.RecieveCard(card);
    Debug.Log($"自分 Recieve : {card % 13}");

    //GameManagerを参照
    GameManager game = GameManager.Instance;
    //参照先のPrefab生成
    var cardObj = Instantiate(game.CardPrefab, transform);
    //カードのゲームオブジェクトのspriteを該当する絵柄に変える
    cardObj.sprite = game.Sprites[card];

    //描画の順番 追加されたら重ねる
    cardObj.sortingOrder = _drawCardsObjs.Count;
    //リストに生成したPrefab追加
    _drawCardsObjs.Add(cardObj);
}
```



Playerのカードの描画位置のプログラム
-どうやって描画する？

Assets > Prefabs



Card

Inspector

Card (Prefab Asset)

Open

Root in Prefab Asset (Open for full editing support)

Card

Tag Untagged Layer Default

Transform

Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 0.1 Y 0.1 Z 1

Sprite Renderer

Sprite BackColor_Blk

Color

Flip X Y

Draw Mode Simple

Mask Interaction None

Sprite Sort Point Center

Material Sprites-Default

Additional Settings

Sorting Layer Default
Order in Layer 0

Sprites-Default (Material)

Shader Sprites/Default

Add Component

PlayerSelf.cs

```
//GameManagerを参照
GameManager game = GameManager.Instance;
//参照先のPrefab生成
var cardObj = Instantiate(game.CardPrefab, transform);
//カードのゲームオブジェクトのspriteを該当する絵柄に変える
cardObj.sprite = game.Sprites[card];

//描画の順番 追加されたら重ねる
cardObj.sortingOrder = _drawCardsObjs.Count;
//リストに生成したPrefab追加
_drawCardsObjs.Add(cardObj);

/*カードの枚数によってカードの描画の位置を決める*/
//y座標
float yPos = -1f;//Selfのカードのy座標は固定

//x座標
float leftEdge = -2.3f;//カードを置ける限界値(左)
float rightEdge = 2.3f;//カードを置ける限界値(右)

float interval = (rightEdge - leftEdge) / (_drawCardsObjs.Count * 2);
float xPos = leftEdge + interval;
foreach (SpriteRenderer drawCardsObj in _drawCardsObjs)
{
    drawCardsObj.transform.localPosition = new Vector3(xPos, yPos, 0);
    xPos += interval * 2;
}
```



Game Manager (Script)



Script

GameManager



Player Self

PlayerSelf (Player Self)



Player Enemy

PlayerEnemy (Player Enemy)



Card Prefab

Card (Sprite Renderer)



Back Sprite

BackColor_Black



Sprites

52

「leftEdge」

rightEdge



カードを置ける幅

PlayerSelf.cs

```
//GameManagerを参照
GameManager game = GameManager.Instance;
//参照先のPrefab生成
var cardObj = Instantiate(game.CardPrefab, transform);
//カードのゲームオブジェクトのspriteを該当する絵柄に変える
cardObj.sprite = game.Sprites[card];

//描画の順番 追加されたら重ねる
cardObj.sortingOrder = _drawCardsObjs.Count;
//リストに生成したPrefab追加
_drawCardsObjs.Add(cardObj);

/*カードの枚数によってカードの描画の位置を決める*/
//y座標
float yPos = -1f;//Selfのカードのy座標は固定

//x座標
float leftEdge = -2.3f;//カードを置ける限界値(左)
float rightEdge = 2.3f;//カードを置ける限界値(右)

float interval = (rightEdge - leftEdge) / (_drawCardsObjs.Count * 2);
float xPos = leftEdge + interval;
foreach (SpriteRenderer drawCardsObj in _drawCardsObjs)
{
    drawCardsObj.transform.localPosition = new Vector3(xPos, yPos, 0);
    xPos += interval * 2;
}
```

「leftEdge」

「rightEdge」



「_drawCardsObjs.Count」
=「手札のカードの枚数」

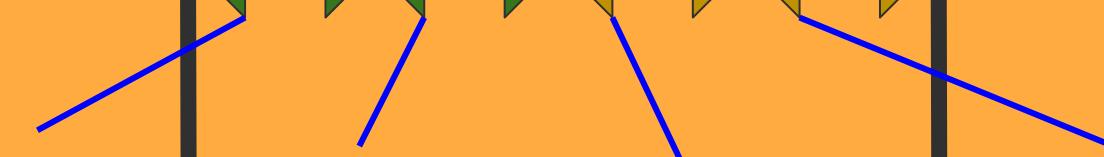


「interval」

「interval」

「interval」

「interval」



PlayerSelf.cs

```
//GameManagerを参照
GameManager game = GameManager.Instance;
//参照先のPrefab生成
var cardObj = Instantiate(game.CardPrefab, transform);
//カードのゲームオブジェクトのspriteを該当する絵柄に変える
cardObj.sprite = game.Sprites[card];

//描画の順番 追加されたら重ねる
cardObj.sortingOrder = _drawCardsObjs.Count;
//リストに生成したPrefab追加
_drawCardsObjs.Add(cardObj);

/*カードの枚数によってカードの描画の位置を決める*/
//y座標
float yPos = -1f;//Selfのカードのy座標は固定

//x座標
float leftEdge = -2.3f;//カードを置ける限界値(左)
float rightEdge = 2.3f;//カードを置ける限界値(右)

float interval = (rightEdge - leftEdge) / (_drawCardsObjs.Count * 2);
float xPos = leftEdge + interval;
foreach (SpriteRenderer drawCardsObj in _drawCardsObjs)
{
    drawCardsObj.transform.localPosition = new Vector3(xPos, yPos, 0);
    xPos += interval * 2;
}
```

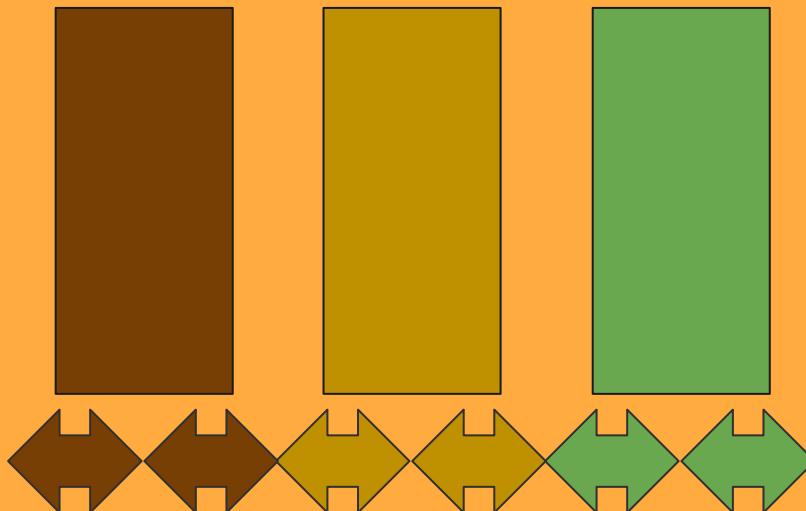
「leftEdge」

「rightEdge」



「_drawCardsObjs.Count」

=「手札のカードの枚数」



実行

数値は個人で設定

授業通りにプログラムしても
挙動が思った通りにならないはずです。直して下さい。

どういった挙動がおかしいか、どう直したかなどを教えて下さい。
(こちらの想定しているバグは
「いろんなカードが表示され続けるが、エラーはでない」)

点数を加算します。

授業終了

あとは

- ・バグを直す
 - ・結果を表示させる
- など、色々自分で作業してみて下さい

次からのスライド

- ・バグの治し方
- ・追加実装例

など

講義はしない

GameManager.cs

修正前

```
case GameState.PlayerTurn:  
    if (_hitClicked)  
    {  
        _hitClicked = false;  
        //山札の一番上からカードを引く  
        _playerSelf.RecieveCard(_cards[_cardTop]);  
        //カードの一番上の要素番号を更新  
        _cardTop++;  
        State = GameState.EnemyTurn;  
    }  
    if (_standClicked)  
    {  
        _standClicked = false;  
        State = GameState.EnemyTurn;  
    }  
    break;
```

GameManager.cs

修正版

```
case GameState.PlayerTurn:  
    if (_hitClicked)  
    {  
        _hitClicked = false;  
        //山札の一番上からカードを引く  
        _playerSelf.RecieveCard(_cards[_cardTop]);  
        //カードの一番上の要素番号を更新  
        _cardTop++;  
        if (0 == _playerSelf.CalcPoint())  
        {  
            State = GameState.EnemyTurn;  
        }  
    }  
    if (_standClicked)  
    {  
        _standClicked = false;  
        State = GameState.EnemyTurn;  
    }  
    break;
```

```
case GameState.Initialize:  
    InitialCards(); //山札をシャッフル  
    _playerSelf.Initialize(); //自分の手札を初期状態に  
    _playerEnemy.Initialize(); //敵の手札を初期状態に  
    DealCards(); //カードを配る  
    State = GameState.PlayerTurn;  
    break;
```

```
case GameState.Initialize:  
    InitialCards(); //山札をシャッフル  
    _playerSelf.Initialize(); //自分の手札を初期状態に  
    _playerEnemy.Initialize(); //敵の手札を初期状態に  
    DealCards(); //カードを配る  
    State = GameState.PlayerTurn;  
  
    _hitClicked = false;  
    _standClicked = false;  
    _retryClicked = false;  
    break;
```

Enemyも同じようにしてカードを描画

PlayerEnemy.cs

```
public override void RecieveCard(int card)
{
    base.RecieveCard(card);
    Debug.Log($" 敵 Recieve (インデックス番号) : {card % 13}");

    //GameManagerを参照
    GameManager game = GameManager.Instance;
    //参照先のPrefab(カードのゲームオブジェクト)を生成
    SpriteRenderer cardObj = Instantiate(game.CardPrefab, transform);

    if (_drawCardsObjs.Count == 0)          初めのカードは伏せておく
    {
        cardObj.sprite = game.BackSprite;
    }
    else
    {
        //カードのゲームオブジェクトのspriteを該当する絵柄に変える
        cardObj.sprite = game.Sprites[card];
    }
    cardObj.sortingOrder = _drawCardsObjs.Count; //描画の順番 追加されたら重ねる
    _drawCardsObjs.Add(cardObj); //リストに生成したPrefab追加
```

PlayerEnemy.cs

```
//カードの枚数によってカードの描画の位置を決める
float yPos = 3f; //自身のカードのy座標は固定

//x座標に関して
float leftEdge = -2.3f; //カードを置ける限界値(左)
float rightEdge = 2.3f; //カードを置ける限界値(右)
float interval = (rightEdge - leftEdge) / (_drawCardsObjs.Count * 2);
float xPos = leftEdge + interval;
foreach (SpriteRenderer drawCardsObj in _drawCardsObjs)
{
    drawCardsObj.transform.localPosition = new Vector3(xPos, yPos, 0);
    xPos += interval * 2;
}
```

Enemyのカードを裏返していたものを表にする

PlayerBase.cs

```
public class PlayerBase : MonoBehaviour
{
    //手札のリスト
    protected List<int> _hand = new List<int>();
```

PlayerEnemy.cs

```
public void CardOpen()
{
    _drawCardsObjs[0].sprite = GameManager.Instance.Sprites[_hand[0]];
}
```

GameManager.cs

```
case GameState.EnemyTurn:  
    _playerEnemy.CardOpen();  
    //ヒットするかどうか判断  
    //「計算結果が0より上」かつ「16より下」ならtrue（合計値が1~15の時、実行）  
    if (_playerEnemy.HitAI()) {  
        //山札の一番上からカードを引く  
        _playerEnemy.RecieveCard(_cards[_cardTop]);  
        //忘れずにカードの一番上を表す変数を足す  
        _cardTop++;  
    }  
    //スタンドするかどうか判断  
    //計算結果が0より上かつ16より下なら(1~15なら)      false ,  
    //0 (22以上はバーストで0として処理) もしくは16以上ならtrue  
    else if (_playerEnemy.StandAI()) {  
        ExecResult(); //結果を出力  
        State = GameState.Result;  
    }  
    break;
```

実行

結果表示

- ・ここまでの中身(バグも直してある状態)
- ・「結果表示」(勝ち・負け・引き分けなどをUIとして表示)
- ・その他

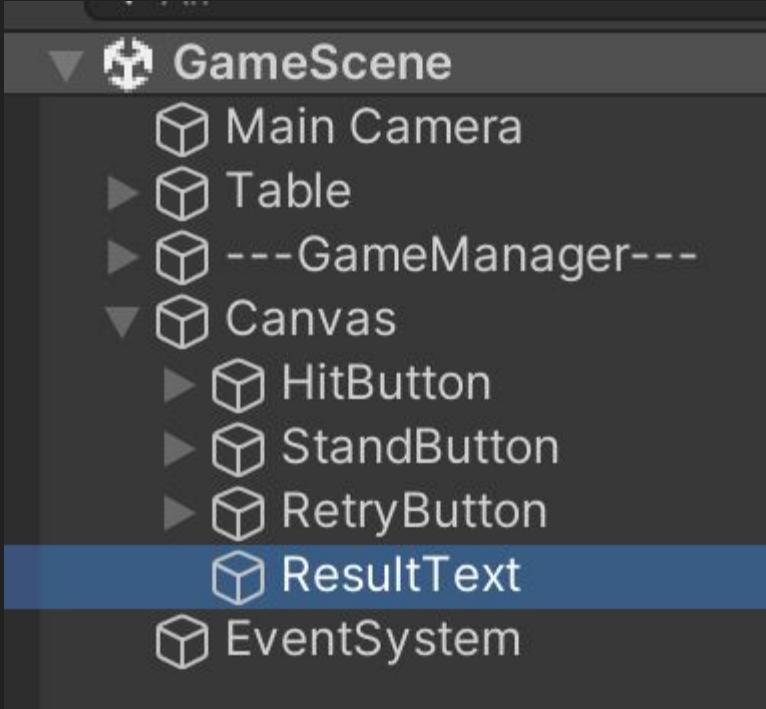
上記のうち2点以上を自分でうまく実装できれば、
単位取得できるようにします。

New Text

Hit

Stand

Retry



GameManager.cs

```
using UnityEngine;
using TMPro;
```

```
[SerializeField] private TextMeshProUGUI _resultText;
```

GameManager.cs

```
//初期状態に行う処理
void InitializeCards()
{
    _resultText.gameObject.SetActive(false); //結果画面非表示

    _cardIndex = 0; //初期化として 0を代入しておく
    _cards = new int[totalNum];
    //カードの枚数分 52枚 リストの要素として数字を入れる
    for (int i = 0; i < totalNum; i++)
    {
        _cards[i] = i;
    }

    //リストをシャッフル
    for (int i = 0; i < totalNum; i++) {
        //randomIndexは0～totalNumの間のランダムな数とする
        int randomIndex = Random.Range(0, totalNum);
        int temp = _cards[i]; //i番目のカードをtempに代入
        _cards[i] = _cards[randomIndex]; //randomIndex番目のカードをi番目に代入
        _cards[randomIndex] = temp;      //tempをrandomIndex番目に代入
    }
}
```

//結果出力

```
void ExecResult() {
    State = GameState.Result;

    int selfPoint = _selfPlayer.CalcPoint(); //自身の手札の計算結果
    int enemyPoint = _enemyPlayer.CalcPoint(); //敵の手札の計算結果
    if (selfPoint > enemyPoint)
    {
        Debug.Log($"勝負:勝ち");
    }
    else if (selfPoint == enemyPoint)
    {
        Debug.Log($"勝負:引き分け");
    }
    else if (selfPoint < enemyPoint)
    {
        Debug.Log($"勝負:負け");
    }
    //Debug.Log($"自身の手札(インデックス番号):{_cards[0] % 13}・{_cards[1] % 13}");
    //Debug.Log($"敵の点数(インデックス番号):{_cards[2] % 13}・{_cards[3] % 13}");
    Debug.Log($"自身の点数:{selfPoint} , 敵の点数:{enemyPoint}");
}
```

GameManager.cs

GameManager.cs

```
//結果出力
void ExecResult() {
    State = GameState.Result;

    int selfPoint = _selfPlayer.CalcPoint(); //自身の手札の計算結果
    int enemyPoint = _enemyPlayer.CalcPoint(); //敵の手札の計算結果

    _resultText.gameObject.SetActive(true); //結果画面非表示
    if (selfPoint > enemyPoint)
    {
        Debug.Log($"勝負:勝ち");
        _resultText.text = "Win";
    }
    else if (selfPoint == enemyPoint)
    {
        Debug.Log($"勝負:引き分け");
        _resultText.text = "Draw";
    }
    else if (selfPoint < enemyPoint)
    {
        Debug.Log($"勝負:負け");
        _resultText.text = "Lose";
    }
    //Debug.Log($"自身の手札(インデックス番号):{_cards[0] % 13}・{_cards[1] % 13}");
    //Debug.Log($"敵の手札(インデックス番号):{_cards[2] % 13}・{_cards[3] % 13}");
    Debug.Log($"自身の点数:{selfPoint} , 敵の点数:{enemyPoint} ");
```

Inspector **GameManager** Static

Tag Untagged

Layer Default

**Transform**

Position

X -0.628 Y -0.059 Z -0.091

Rotation

X 0 Y 0 Z 0

Scale

X 1 Y 1 Z 1

**Game Manager (Script)**

Script

GameManager

Self Player

SelfPlayer (Self Player)

Enemy Player

EnemyPlayer (Enemy Pla

Card Prefab

Card (Sprite Renderer)

Back Sprite

BackColor_B

Sprites

52

Result Text

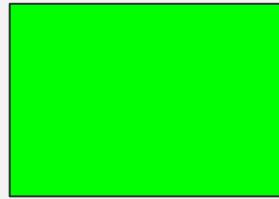
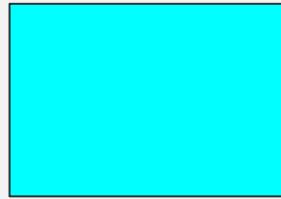
ResultText (Text Mesh Pr

Add Component

実行

終了

メモリ



メモリ

