講義メモ
・p.120 chap3_8_1（配列とforeachによる２重ループ）から

提出フォロー：ミニ演習：フィボナッチ数列 mini117 続き

・数列の要素数を10にして、要素[2]以降の値のセットをfor文による繰返しで行うようにしよう
・ヒント： for(int i = 2; i < fib.Length; i++) { fib[i] = fib[i - 2] + fib[i - 1]; }
・実行結果： 1,1,2,3,5,8,13,21,34,55
・合計は、そのままforeachで得ると良い

作成例

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

public class mini117 : MonoBehaviour {
    void Start() {
        int[] fib = new int[10]; //要素数10のフィボナッチ数列を生成
        fib[0] = fib[1] = 1; //配列fibの要素[0]と要素[1]に1を代入
        for (int i = 2; i < fib.Length; i++) { //要素[2]以降の全要素について繰返す
            fib[i] = fib[i - 2] + fib[i - 1]; //前２要素の和を代入
        }
        Debug.Log(string.Join(",", fib)); //配列fibの全要素を連結表示(カンマ区切り)
        int sum = 0; //合計用
        foreach(var work in fib) { //数列の全要素について作業変数を用いて繰返す
            sum += work; //合計に足し込む
        }
        Debug.Log("計＝" + sum); //合計を表示
    }
    void Update() {

    }
}
```

p.120 chap3_8_1（配列とforeachによる２重ループ）から

・foreachによる多重ループが記述できる
・この場合、作業変数は別の名前にすること
・p.120 chap3_8_1では同じ配列を用いているが、同じでなくても良い

p.120 chap3_8_1

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

public class chap3_8_1 : MonoBehaviour {
    void Start() {
        string[] team = { "A", "B", "C", "D" };
        foreach (string t1 in team) { //全要素の分、作業変数t1を用いて繰返す(x4)
            foreach (string t2 in team) { //全要素の分、作業変数t2を用いて繰返す(x4)
                Debug.Log(t1 + "vs" + t2); //2重ループの中なので作業変数t1とt2が
```

```
使える(x16)
            }
        }
    }
    void Update() {

    }
}
```

p.122 chap3_8_2：chap3_8_1をそのまま修正し同一チームの対戦を取り除く

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

public class chap3_8_1 : MonoBehaviour {
    void Start() {
        string[] team = { "A", "B", "C", "D" };
        foreach (string t1 in team) { //全要素の分、作業変数t1を用いて繰返す(x4)
            foreach (string t2 in team) { //全要素の分、作業変数t2を用いて繰返す
(x4)
                if (t1 != t2) { //同一チームでなければ
                    Debug.Log(t1 + "vs" + t2); //2重ループの中なので作業変数t1と
t2が使える(x12)
                }
            }
        }
    }
    void Update() {

    }
}
```

p.123 chap3_8_3：chap3_8_1をそのまま修正し同一組み合わせも取り除く

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

public class chap3_8_1 : MonoBehaviour {
    void Start() {
        string[] team = { "A", "B", "C", "D" };
        int start = 1; //開始添字
        foreach (string t1 in team) { //全要素の分、作業変数t1を用いて繰返す(x4)
            for (int cnt = start; cnt < 4; cnt++) { //開始添字以降の要素で、変数
cntを用いて繰返す
                Debug.Log(t1 + "vs" + team[cnt]); //2重ループの中なので作業変数
t1とt2が使える(x12)
            }
            start++; //開始添字を先に進める
        }
    }
    void Update() {

    }
}
```

ミニ演習：chap3_8_1～3 改造 mini123

・forの２重ループにして変数startを用いないように改良しよう

作成例

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

public class mini123 : MonoBehaviour {
    void Start() {
        string[] team = { "A", "B", "C", "D" };
        for (int start = 0; start < 4; start++) { //全要素の分、作業変数startを
用いて繰返す(x4)
            for (int cnt = start + 1; cnt < 4; cnt++) { //次の要素以降で、変数
cntを用いて繰返す
                Debug.Log(team[start] + "vs" + team[cnt]); //2重ループの中(x6)
            }
        }
    }
    void Update() {

    }
}
```

p.124 無限ループを止める

・コンポーネントの削除によって対処する場合は、Inspectorのスクリプト名の右端の「…」をクリックして「Remove Conponent」
※ Unityのバージョンアップで歯車アイコンから「…」の縦字に変更

p.125 配列の上限を超える場合

・配列の添字が要素数以上または負の数になった場合、実行時エラー「IndexOutOfRangeException（添字範囲外例外）」が発生する
・実行時エラーは文法エラーではないので、Visual Studioのエディタでは発生がわからない
  ※ 実行時エラーが起こることが明らかな場合はエラーや警告が表示されることもある
・実行時エラーの発生場所はUnity側に表示されるメッセージで確認できる
・書式：
  [発生時刻]発生したエラーを示す例外クラス名:メッセージ
  実行クラス名.メソッド名(…)(at Assets/ソースファイル名:行番号)
・例：
  [12:50:02]IndexOutOfRangeException: Index was outside the bounds of the array.
  chap3_6_1.Start () (at Assets/chap3_6_1.cs:9)

p.126 復習ドリル 問題1 chap3_10_1

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

public class chap3_10_1 : MonoBehaviour {
    void Start() {
        string[] grade = { "松", "竹", "梅" };
        foreach (var g in grade) { //配列gradeの全要素について作業変数gを用いて
繰返す
            Debug.Log(g);
```

```
        }
    }
    void Update() {

    }
}
```

p.126 復習ドリル 問題1 chap3_10_2

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

public class chap3_10_2 : MonoBehaviour {
    void Start() {
        string[] dirs = { "東", "西", "南", "北"};
        for (int cnt = dirs.Length - 1; cnt >= 0 ; cnt--) { //変数cntを用いて逆順で繰返す
            Debug.Log(dirs[cnt] + "方向");
        }
    }
    void Update() {

    }
}
```

補足：２次元配列

・C#の２次元配列はC/C++とは異なり、２つの添字を「配列名[添字①, 添字②]」の形式で指定する
　※ 後述するが、C/C++と同様に「配列名[添字①][添字②]」の形式で指定する別形式もある
・宣言の書式： 型[,] 配列名;
・生成の書式： 配列名 = new 型[要素数①, 要素数②];
・同時にして、 型[,] 配列名 = new 型[要素数①, 要素数②]; としても良い
例： 縦３列横４列のダンジョンのモンスター配置数 int[,] monsters = new int[3,4];
・２次元配列も添字は0からなので、要素は[0, 0]から[要素数①-1, 要素数②-1]まで。
・２次元配列の全要素を扱うには、forの2重ループを用いると良い

```
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
      Debug.Log(monsters[i][j]);
    }
  }
```

・２次元配列も初期化が可能で、{}を２重に用いる
・例： int[,] monsters = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};

ミニ演習 mini128

・上記の例を試そう
・２次元配列monstersを初期化し、全データを表示し、配置数の合計も表示しよう

作成例

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

```
public class mini128 : MonoBehaviour {
    void Start() {
        //２次元配列の初期化
        int[,] monsters = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };
        int sum = 0; //合計用
        for (int i = 0; i < 3; i++) { //変数iを用いて繰返す
            for (int j = 0; j < 4; j++) { //変数jを用いて繰返す
                Debug.Log("monsters[" + i + "," + j + "] = " + monsters[i, j]);
                sum += monsters[i, j]; //合計に足し込む
            }
        }
        Debug.Log("配置数合計＝" + sum);
    }
    void Update() {

    }
}
```

ミニ演習 mini128a

・２次元配列monstersの[0,0]から[0,3]までを0階、[1,0]から[1,3]までを1階、[2,0]から[2,3]までを2階とみなし、階ごとの配置数と合計を表示しよう
実行結果
0階配置数＝10
1階配置数＝26
2階配置数＝42
合計＝78

作成例

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

public class mini128a : MonoBehaviour {
    void Start() {
        //２次元配列の初期化
        int[,] monsters = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };
        int all = 0, sum = 0; //合計用(全、階ごと)
        for (int i = 0; i < 3; i++) { //変数iを用いて繰返す
            for (int j = 0; j < 4; j++) { //変数jを
用いて繰返す

          sum += monsters[i, j]; //階ごと合計に足し込む
            }
            Debug.Log(i + "階配置数＝" + sum);
            all += sum; //全合計に足し込む
            sum = 0; //階ごと合計をクリア
        }
        Debug.Log("合計＝" + all);
    }
    void Update() {

    }
}
```

補足：ジャグ配列

・C/C++と同様に「配列名[添字①][添字②]」の形式で指定する別形式の多次元配列
・処理効率は下がるが、要素数がそろっていない場合にも対応できる
例： a[0][0]←1, a[0][1]←2, a[1][0]←3, a[2][0]←4, a[2][1]←5, a[2][2]←6
・内部構造は「配列の配列」であり、上の例は内部的にはa[0]{1,2}、a[1]{3}、a[2]{4,5,6}とみなされる
・配列数は配列名.Lengthで、内側の配列の要素数は配列名[添字].Lengthで得られる
・宣言の書式： 型[][] 配列名;
・生成の書式： 配列名 = new 型[要素数①][]; 配列名[添字] = new 型[要素数②];…
・例： int[][] monsters = new int[3][]; monsters[0] = new int[2]; monsters[1] = new int[1]; monsters[2] = new int[3];
・初期化は通常の配列と同様
例： int[][] monsters = new int[3][]; monsters[0] = new int[]{1, 2}; monsters[1] = new int[]{3}; monsters[2] = new int[]{4, 5, 6};

ミニ演習 mini128b

・上記の例を試そう
・２次元のジャグ配列monstersを初期化し、全データを表示し、配置数の合計も表示しよう

作成例

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

public class mini128b : MonoBehaviour {
    void Start() {
        //２次元ジャグ配列の初期化
        int[][] monsters = new int[3][];
        monsters[0] = new int[]{1, 2}; monsters[1] = new int[]{3}; monsters[2] = new int[]{4, 5, 6};
        int sum = 0; //合計用
        for (int i = 0; i < monsters.Length; i++) { //変数iを用いて内側の全配列分繰返す
            for (int j = 0; j < monsters[i].Length; j++) { //変数jを用いて内側の全要素分繰返す
                Debug.Log("monsters[" + i + "," + j + "] = " + monsters[i][j]);
                sum += monsters[i][j]; //合計に足し込む
            }
        }
        Debug.Log("配置数合計＝" + sum);
    }
    void Update() {

    }
}
```

ミニ演習 mini128c

・２次元ジャグ配列monstersの[0]を0階、[1]を1階、[2]を2階とみなし、階ごとの配置数と合計を表示しよう
実行結果
0階配置数＝3
1階配置数＝3
2階配置数＝15
合計＝21

作成例

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

public class mini128c : MonoBehaviour {
    void Start() {
        //２次元ジャグ配列の初期化
        int[][] monsters = new int[3][];
        monsters[0] = new int[] { 1, 2 }; monsters[1] = new int[] { 3 };
monsters[2] = new int[] { 4, 5, 6 };
        int all = 0, sum = 0; //合計用(全、階ごと)
        for (int i = 0; i < monsters.Length; i++) { //変数iを用いて内側の全配列
分繰返す
            for (int j = 0; j < monsters[i].Length; j++) { //変数jを用いて内側の
全要素分繰返す
                sum += monsters[i][j]; //合計に足し込む
            }
            Debug.Log(i + "階配置数＝" + sum);
            all += sum; //全合計に足し込む
            sum = 0; //階ごと合計をクリア
        }
        Debug.Log("合計＝" + all);
    }
    void Update() {

    }
}
```

補足：パブリック配列

・Unity環境では、パブリック変数と同様に、配列をpublic指定で定義できる
・すると、要素数の分だけ、入力領域が用意される
  例： public int[] monsters = new int[3];
・定義と生成までを行うこと

ミニ演習 mini128d

・3要素のint型パブリック配列monstersを宣言・生成する
・Unity側で適当な値を入力すると、平均値を実数で表示しよう

提出：ミニ演習 mini128d

次回予告：p.130 ゲームオブジェクト、コンポーネント、クラス