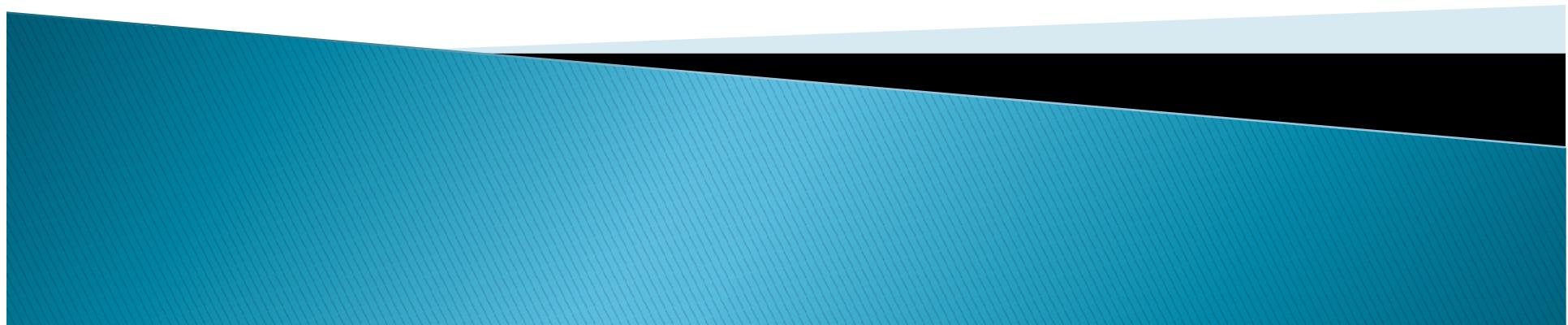


C#でゲームを作ろう2017

第2回 担当:ten

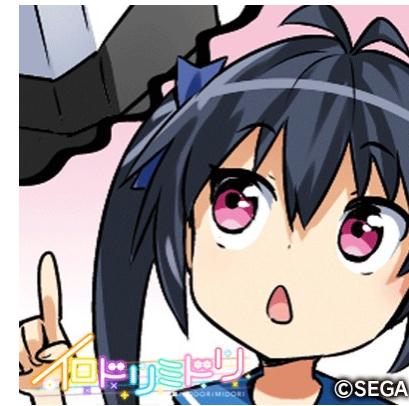


自己紹介

- ▶ ID:ten
- ▶ 京都大学工学部情報学科2回生
 - 計算機科学コース
- ▶ 第40代会長
- ▶ ゲーム制作して競プロしてる
- ▶ パズルと音ゲー



Twitter



Slackとか

初めての店で大盛りにするな！！



ten

@ten986



13

いいね



初めての店で大盛りにするな！！

- ▶ 死亡
- ▶ ぎりぎり食えたけどつらい



ten

@ten986



初めての店で飯増し肉増しを頼んではいけない

1
リツイート

14
いいね

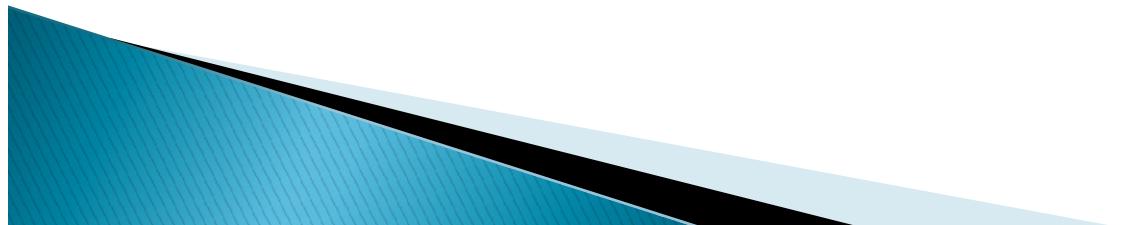


自己紹介をしましょう

- ▶ ID or 本名
- ▶ 所属（学部学科とか）
- ▶ 好きな or 印象的な 飯屋
- ▶ なにか一言

※すべて任意です

※なにか一言が一番難しい気がする



今日の流れ

1. 復習
2. 関数
3. 多次元配列
4. List
5. クラス
6. enum(列挙型)



ゲームを考えてきてね

- ▶ 夏休み後あたりに発表会をする予定
- ▶ 夏休み前に内容を終えるので、そこから自分のゲームを作ってもらう予定
- ▶ というわけでゲームの内容を考えてきてください
- ▶ いわゆる「3分ゲー」のようなもので大丈夫です



Monogameは次回やります

- ▶ 今日はプログラムのお話の続き
- ▶ 以降はこの話を用いて実際にゲームを作ろうというお話をします
- ▶ 応用とか、補足説明とか



Slack

- ▶ #csgame
- ▶ 質問があればここで or 口頭で
- ▶ この時間中の質問はいつでも受け付けています
- ▶ どんどん質問しましょう
- ▶ 忘れたらスライドとかどんどん見よう
- ▶ 調べよう



復習

- ▶ 第1回スライドを使うなど



今日やること

- ▶ ここを脱出します

```
using System.Threading.Tasks;
```

```
- namespace ConsoleApp10
  [
    - class Program
      [
        static void Main(string[] args)
        [
          ]
      ]
    ]
  ]
}
```



関数

- ▶ メソッドともいう
- ▶ いくつか値を受け取っていろいろ操作して1つの値を返す

- ▶ 数学的にはこんなの
- ▶ $f(x) = x^3$
 - x を受け取って3乗を返す～と言えそう



関数

- ▶ 関数の例
- ▶ `System.Console.ReadLine();`
 - 入力を読み取る
 - `string`型を返す
- ▶ `System.Console.WriteLine(なんか);`
 - ()内の変数や文字列を出力する
 - 返す値はない
- ▶ こういう関数を自分で作れる



関数

- ▶ 例えばこんなの
- ▶ Mainの外にCubeがあることに注意

```
static int Cube(int x)
{
    return x * x * x;
}
static void Main(string[] args)
{
    System.Console.WriteLine(Cube(2));
    System.Console.WriteLine(Cube(7));
}
```

関数

- ▶ キーワード 返り値型 関数名(引数1,引数2,...){
 処理
 return なんとか;
}
- ▶ こういう感じで作る



関数

- ▶ キーワード
 - 今はstaticにしてください
- ▶ 返り値型
 - 返す値の型　返さない場合はvoid
- ▶ 引数
 - 関数の処理に必要な値
- ▶ return なんとか;
 - なんとかを返す



関数

- ▶ int型を返す関数
- ▶ 引数としてint型の値を受け取る
 - その値をxに代入
- ▶ $x \times x \times x$ (つまりxの3乗)を返す

```
static int Cube(int x)
{
    return x * x * x;
}
```



関数

- ▶ 関数を使うときは
関数名(引数1,引数2...)と書けばよい
- ▶ Cube(2)→8

```
static void Main(string[] args)
{
    System.Console.WriteLine(Cube(2));
    System.Console.WriteLine(Cube(7));
}
```



関数

- ▶ 関数内で関数を呼び出すこともできる

```
bool IsPrime(int x) {  
    if (x <= 1) {return false;}  
    else  
    {  
        for(int i = 2; i < x; ++i) {  
            if (IsPrime(i)){  
                if (x % i == 0) {return false;}  
            }  
        }  
        return true;  
    }  
}
```

関数

- ▶ ちなみに今のは**非常に効率の悪いコード**
- ▶ **使うな。**
- ▶ 中で素数判定せずに全整数で割れよ
- ▶ 素数と分かったものについては配列とかで管理するとい

- ▶ プログラムの高速化について興味がある人は
「競技プログラミング練習会」
へ！！！！！！！！！！



関数

- ▶ 関数の名前が同じで引数が違うものを複数作ることができます
- ▶ オーバーロードといいます

```
static int Cube(int x)
{
    return x * x * x;
}
static double Cube(double x)
{
    return x * x * x;
}
```

関数

- ▶ 備考
- ▶ 実はMainも関数だったりする
- ▶ 関数名は長くても分かりやすく！！
 - Visual Studioは補完が優秀
 - 補完最高



多重ループ

- ▶ forやwhileの中にforやwhileを入れられる
- ▶ 九九を表示する例

```
for (int i = 1; i <= 9; ++i)
{
    for (int j = 1; j <= 9; ++j)
    {
        System.Console.WriteLine(i + " × " + j + " = " + (i * j));
    }
}
```



多次元配列

- ▶ 1次元配列（前回扱った配列）
 - 一直線に変数を並べる感じ



- ▶ 2次元配列
 - 四角く変数を並べる



多次元配列

- ▶ 配列の初期化の方法は2つあった
- ▶ 要素数を指定→1つずつ値を入れる

```
int[] enemy_hp=new int[3];  
enemy_hp[0]=20;  
enemy_hp[1]=25;  
enemy_hp[2]=30;
```

- ▶ 値を始めから入れる

```
int[] enemy_hp=new int[]{20,25,30};
```



多次元配列

- ▶ 要素数を指定→1つずつ値を入れる

```
int[,] array = new int[3,4];
for (int i = 0; i < 3; ++i)
{
    for (int j = 0; j < 4; ++j)
    {
        array[ i , j ] = i * 10 + j;
    }
}
System.Console.WriteLine(array[1, 0]);
System.Console.WriteLine(array[2, 3]);
```

多次元配列

- ▶ `int[,] array = new int[3,4];`
- ▶ このように書くと
- ▶ `array[0,0] array[0,1] array[0,2] array[0,3]`
- ▶ `array[1,0] array[1,1] array[1,2] array[1,3]`
- ▶ `array[2,0] array[2,1] array[2,2] array[2,3]`
- ▶ が使える



多次元配列

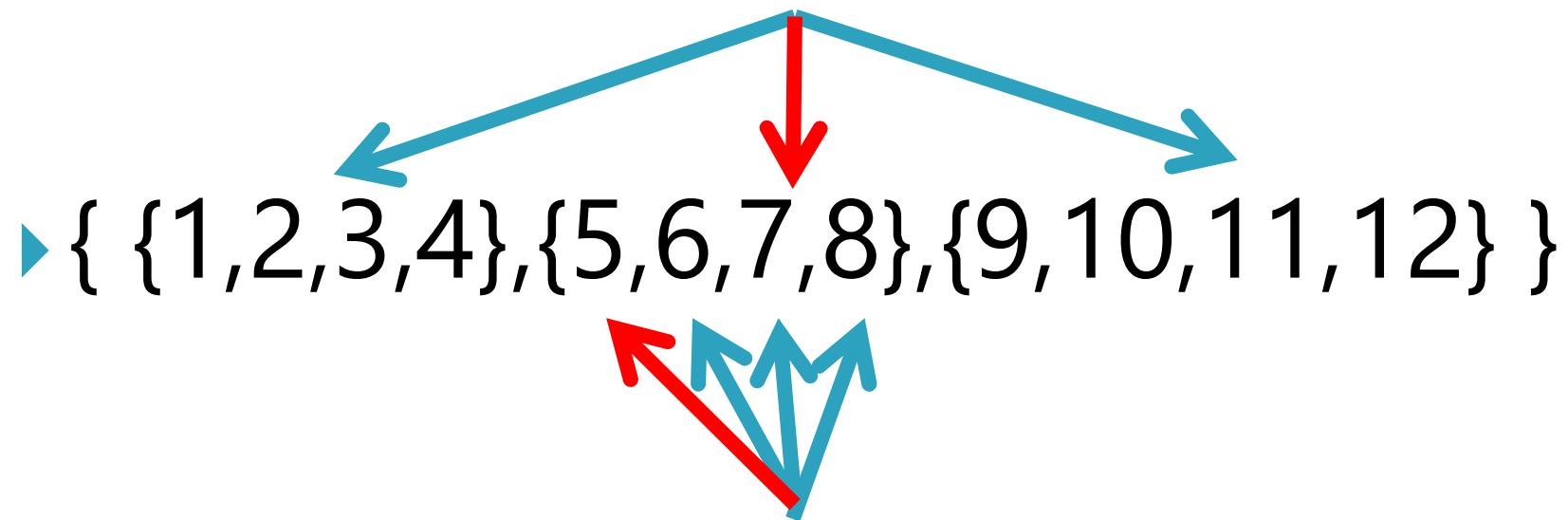
- ▶ 値を始めから入れる

```
int[,] array = new int[,] { {1,2,3,4},{5,6,7,8},{9,10,11,12} };  
System.Console.WriteLine(array[1, 0]); //5  
System.Console.WriteLine(array[2, 3]); //12
```



多次元配列

[a,b]のaで選ぶ



[a,b]のbで選ぶ

多次元配列

- ▶ 3次元以上もできる

```
int[,] array = new int[3,5,7];  
int[,] array2=new int[,]{ {{1,2},{3,4}},{{5,6},{7,8}} }
```

- ▶ 2次元以上の配列のイメージ

```
{1,2}  
{ {1,2}, {1,2} }  
{ { {1,2}, {1,2} }, { {1,2}, {1,2} } }
```



List

- ▶ 配列では要素数が後から変えられなかった
- ▶ 世の中には数が後で変わるもののがいっぱい
 - 敵の数とか弾の数とか
- ▶ そこでList
- ▶ Listは要素数可変
- ▶ List<型> 変数名=new List<型>();
- ▶ これで初期化



List

▶ こんなん

```
List<int> list = new List<int>();  
list.Add(3);  
list.Add(5);  
list.Add(7);  
list.Add(5);  
  
for (int i = 0; i < list.Count; ++i)  
{  
    System.Console.WriteLine(list[i]);  
}
```

List

- ▶ `List<int> list = new List<int>();`
 - int型のListである変数「list」を宣言
 - この時点listには何も入っていない
- ▶ `list.Add(3);`
 - listの後ろに「3」を挿入
- ▶ `list.Count`
 - listの要素数をあらわす



ちょっとforeachの説明

- ▶ 次の2つは同じ

```
for (int i = 0; i < list.Count; ++i)
{
    System.Console.WriteLine(list[i]);
}
```

```
foreach(int a in list)
{
    System.Console.WriteLine(a);
}
```



ちょっとforeachの説明

- ▶ foreach(型 変数名 in リストor配列名)
 - リスト内にある変数を順番に見ていくて、それぞれ処理
- ▶ リストがint型のリストなら、foreachで取り出す変数もintのはず



ListのRemove

▶ こう書いてみる

```
List<int> list = new List<int>();  
list.Add(3);  
list.Add(5);  
list.Add(7);  
list.Add(5);  
  
list.Remove(5);  
  
foreach(int a in list)  
{  
    System.Console.WriteLine(a);  
}
```

ListのRemove

- ▶ `list.Remove(5);`
- ▶ `list`を先頭から見ていき、
最初にあった「5」を`list`から取り除く



ListのRemoveAll

▶ こう書いてみる

```
List<int> list = new List<int>();  
list.Add(3);  
list.Add(5);  
list.Add(7);  
list.Add(5);  
list.Add(11);  
  
list.RemoveAll(e => e >= 5 && e <= 8);  
  
foreach(int a in list)  
{  
    System.Console.WriteLine(a);  
}
```

ListのRemoveAll

- ▶ `list.RemoveAll(e => e >= 5 && e <= 8);`
 - list内の条件を満たす要素すべてを取り除く
- ▶ ()内にあるのはラムダ式
 - まだ分からなくともいいです
 - eがlist内の要素
 $e=>(e$ を使った条件)
くらいの認識で大丈夫そう



クラス

- ▶ 型
 - List<int>
 - Random
- ▶ 型は自分で作ることができる



クラス

- ▶ こんな感じで記述
- ▶ Class クラス名{
 クラスの実装
}

- ▶ クラスの実装では以下を記述する
 - メンバ変数
 - メンバ関数



クラス

- ▶ こういう感じで書く

```
class Enemy
{
    private string name;
    public string GetName()
    {
        return name;
    }
    public void SetName(string s)
    {
        name = s;
    }
}
```

クラス

- ▶ 実装しても使わなきゃ意味がない
- ▶ Main関数内で使ってあげよう
- ▶ 「オブジェクト」を生成している

```
class Program
{
    static void Main(string[] args)
    {
        Enemy enemy = new Enemy();
        enemy.SetName("こいつ");
        System.Console.WriteLine
            ("敵名：" + enemy.GetName());
    }
}
```

クラス

- ▶ Enemyには最初からいろんな処理をしたい
 - 名前なら最初から持つとけよと
- ▶ コンストラクタというものがある

```
class Enemy
{
    (省略)

    public Enemy(string s)
    {
        SetName(s);
    }
}
```

クラス

- ▶ コンストラクタ詳細
 - クラスと同じ名前
 - 戻り値を持たない（けどvoidでもない）
 - 必ずpublic
 - オブジェクト生成時に自動で呼ばれる
 - オーバーロードできる
- ▶ オブジェクト生成時にコンストラクタが呼ばれる
 - なので引数も必要

```
Enemy enemy = new Enemy("こいつ");
```

クラス

- ▶ Random r=new Random();とかもクラス

```
Random r = new Random();
int a = r.Next(10); //0以上10未満の整数
int b = r.Next(-5, 5); // -5以上5未満の整数
double d = r.NextDouble(); //0以上1未満の実数
```



enum 列挙型

- ▶ 特定の値しか取らないものがある
 - 上下左右
 - 日月火水木金土
- ▶ ある変数の0,1,2,3,...を割り振る?
 - 覚えられない
 - バグる
 - つらい
- ▶ そこでenumが使われる



enum 列挙型

- ▶ enum 列挙型名{
 種類1,種類2,種類3,...
}
- ▶ こういう感じで書く
- ▶ クラスと同列に書く（ことが多いと思う）



enum 列挙型

- ▶ 前回のじゃんけんの判定をシュッと書きたい
- ▶ enumを使ってみるといいかも
- ▶ enum Hands
 - {
 - Rock,Scissors,Paper
 - }
- ▶ playerHandsとかをHands型に
- ▶ playerHands==Hands.Rock;とかいい感じに使う



なんか作ろう



参考文献

- ▶ ++C++;(未確認飛行C)
 - <http://ufcpp.net/>

