ゲームプログラミング演習・実習A
ガイダンス

向井 智彦

授業のねらい

- ゲームソフトウェアを題材として
  - Unityを利用した2次元ゲーム
- オブジェクト指向プログラミング言語の基礎を
  - Unity C#スクリプトの一部
- 楽しく＆苦労して修得する
  - 試行錯誤的にオリジナルの面白いゲームを開発

ゲームプログラミング演習・実習B

- ゲームソフトウェアを題材として
  - 3次元CG
- オブジェクト指向デザインの基礎を
  - Unity C#スクリプト → C++プログラム
- 楽しく＆苦労して修得する
  - ゲームエンジンの基礎＆面白いゲーム

修得を目指すスキル

骨太のスキル

- プログラミングのロジック
  - システム要素の分析と分解
  - 抽象化と具現化
  - 数学思考に立脚した開発
    - 命題論理など
- オブジェクト指向デザイン
  - カプセル化（の一部）
  - ポリモーフィズム(の一部)

枝葉のテクニック

- GitHub
- Unity & Visual Studio
- ゲームプログラミング
  - ゲームループ
  - ゲームオブジェクト
  - イベント駆動型プログラミング
  - 状態遷移
  - 衝突判定

プログラミング技術修得の流れ

1. 他人が作ったプログラムを読んでみる
   – 読んでもわからない部分を調べる（講義）

2. プログラムを自分なりに改造してみる
   – 各回の指定課題 or 自由課題

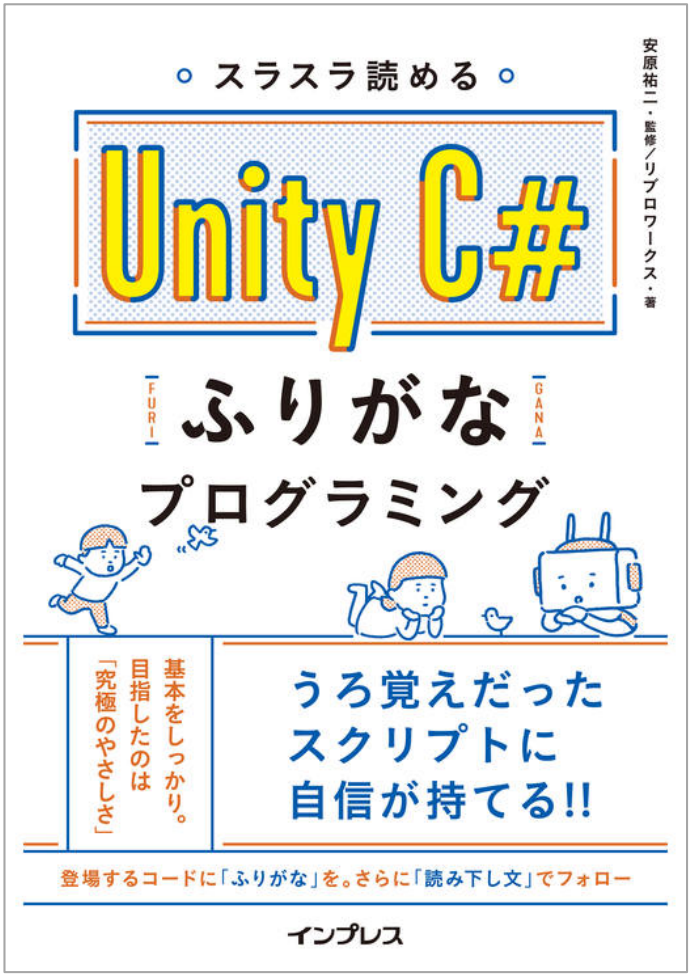
3. オリジナルプログラムを作ってみる
   – 最終制作課題

評価の方針

- 各回の課題：70%
  - 提出状況＋達成状況
- 最終課題：30%
  - レポート（重視）＋ 最終ゲーム

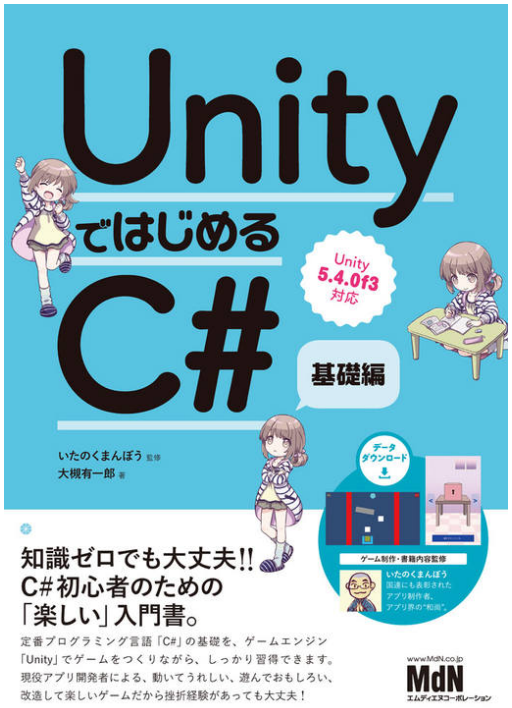
おすすめの書籍1

- 授業前半の参考書
  - とてもわかりやすい
  - 2,000円＋税 （安い！）
  - これを読みながらスライド資料を作成
    - ~~授業に来なくても大丈夫~~



おすすめの書籍2

- ステップバイステップのチュートリアル型
  - いずれも同程度にわかりやすい



おすすめのWeb上リソース

- C#のガイド
  - https://docs.microsoft.com/ja-jp/dotnet/csharp/
- Unityマニュアル
  - https://docs.unity3d.com/ja/current/Manual/
- Unity - Learn
  - https://unity3d.com/jp/learn

公式ドキュメントが最も信頼性が高く確実

本日の演習

- GitHubへのサインアップ&登録
  - 持ってる人はスキップ
- GitHubの練習
  - テキストファイルの編集と提出

課題の実施方法

- GitHub経由での配布＆提出
  - 提出物にコメントを付すため
- プログラミング環境は自由
  - MacOS でも Windows でもサポートします
    - 元企業人なので Windows のほうが得意
  - どちらも学生無償で揃います

GitHub練習

1. アカウント取得（必要な人のみ）
2. GitHubの設定と課題配布
   - 来週までに向井が頑張る
3. 課題実施
   - テキストを書き換えるだけ
4. 課題提出
   - Pull request

Unity試用

- 球転がしゲームを作ってみよう
  - 公式チュートリアル（Roll-a-Ball）
- 向井の説明は無視して進めてOK

プログラミング基礎(C/C++言語)履修者向け

C#プログラミング「出力と演算」

空のスクリプト

Unity C#

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

public class NewBehaviourScript :
MonoBehaviour {
    // Start is called before ...
    void Start() {

    }

    // Update is called once per frame
    void Update() {

    }
}
```

C++風 (あくまで”風”)

```
#include <System/Collections>
#include <System/Collections/Generic>
#include <UnityEngine>

class NewBehaviorScript :
MonoBehaviour {
    // Start is called before ...
    void Start() {

    }

    // Update is called once per frame
    void Update() {

    }
}
```

using ディレクティブ

言葉は忘れてOK

Unity C#

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

C++風 (あくまで"風")

```
#include <System/Collections>
#include <System/Collections/Generic>
#include <UnityEngine>
```

- 別のファイルに格納されているクラスを利用するための命令
  - using UnityEngine;
    ＝ UnityEngine ファイル内のクラスを利用したい
    https://docs.unity3d.com/ja/current/ScriptReference/
  - C/C++の「include」に意味的には近い

クラスの定義

Unity C#

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

public class NewBehaviourScript :
MonoBehaviour {
    // Start is called before ...
    void Start() {

    }

    // Update is called once per frame
    void Update() {

    }
}
```

C++風 (あくまで”風”)

```
#include <System/Collections>
#include <System/Collections/Generic>
#include <UnityEngine>

class NewBehaviorScript :
MonoBehaviour {
    // Start is called before ...
    void Start() {

    }

    // Update is called once per frame
    void Update() {

    }
}
```

- クラス＝複数のメンバーをパッケージ化したもの
  - NewBehaviourScript クラスを定義
  - その際、MonoBehaviour クラスを継承
  - 詳細は来週の講義

メソッド(関数)の定義

Unity C#

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

public class Test : MonoBehaviour {
    // Start is called before ...
    void Start() {

    }

    // Update is called once per frame
    void Update() {

    }
}
```

C++風 (あくまで"風")

```
#include <System/Collections>
#include <System/Collections/Generic>
#include <UnityEngine>

class Test : MonoBehaviour {
    // Start is called before ...
    void Start() {

    }

    // Update is called once per frame
    void Update() {

    }
}
```

- Startメソッド：引数なし、戻り値なし、処理なし
- Updateメソッド：引数なし、戻り値なし、処理なし

メソッドの基本形 (※プログラミング基礎の資料より改変)

戻り値の型名　メソッド名（パラメータ）　{　処理;　}

コメント

```
// 以降はコメント（和文OK） プログラム動作に影響しない
/*
長いコメントを書きたいときは
このように書いてもOK
*/
```

Unity C#

```
using System.Collections;
using UnityEngine;

public class NewBehaviourScript :
MonoBehaviour {
    // Start is called before ...
    void Start() {

    }

    // Update is called once per frame
    void Update() {

    }
}
```

C++風(めくるめく風)

```
#include <System/Collections>
#include <System/Collections/Generic>
#include <UnityEngine>


class NewBehaviorScript :
MonoBehaviour {
    // Start is called before ...
    void Start() {

    }

    // Update is called once per frame
    void Update() {

    }
}
```

最初の例：Hello World

Unity C#

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

public class NewBehaviourScript :
MonoBehaviour {
    // Start is called before ...
    void Start() {
      Debug.Log("Hello World");
    }
    // Update is called once per frame
    void Update() {
    }
}
```

C++風 (あくまで”風”)

```
#include <System/Collections>
#include <System/Collections/Generic>
#include <UnityEngine>
#include <iostream>
class NewBehaviorScript :
MonoBehaviour {
  // Start is called before ...
  void Start() {
    std::cout << "Hello World" << std::endl;
  }

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

- 画面出力は Debug.Log メソッドを利用
- 文字列は”“で囲まれる
- 命令文の終わりにはセミコロン；を付ける

記入場所を変えた Hello World

Unity C#

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

public class NewBehaviourScript :
MonoBehaviour {
    // Start is called before ...
    void Start() {
      // Debug.Log("Hello World");
    }

    // Update is called once per frame
    void Update() {
       Debug.Log("Hello World");
    }
}
```

C++風 (あくまで”風”)

```
#include <System/Collections>
#include <System/Collections/Generic>
#include <UnityEngine>
#include <iostream>
class NewBehaviorScript :
MonoBehaviour {
  // Start is called before ...
  void Start() {

  }

  // Update is called once per frame
  void Update() {
    std::cout << "Hello World" << std::endl;
  }
}
```

コンソール出力

Unity C#

```
using UnityEngine;

public class NewBehaviourScript :
MonoBehaviour {
  // Start is called before ...
  void Start() {
    Debug.Log(2 + 3 / 4);
    Debug.Log(2 + 3.0 / 4.0);
    Debug.Log((2 + 3.0) / 4.0);
    Debug.Log((2 + 3.0) % 4.0);
    Debug.Log(6.0 / 2.0 + 4.0 * 1.5);
    Debug.Log("3 + 4 = " + 3 + 4);
    Debug.Log("3 + 4 = " + (3 + 4));
  }
}
```

C++風 (あくまで”風”)

```
#include <iostream>
using namespace std;
class NewBehaviorScript :
MonoBehaviour {
  // Start is called before ...
  void Start() {
    cout << 2 + 3 / 4 << endl;
    cout << 2 + 3.0 / 4.0 << endl;
    cout << (2 + 3.0) / 4.0 << endl;
    cout << (2 + 3.0) % 4.0 << endl;
    cout << 6.0 / 2.0 + 4.0 * 1.5 << endl;
    cout << "3 + 4 = " << 3 << 4 << endl;
    cout << "3 + 4 = " << 3 + 4 << endl;
  }
}
```

ミニ演習

- ４つの数字「２」「２」「８」「９」を１回ずつ用いた四則演算の結果が「１０」になるような計算式をC#プログラミング
  - 小括弧（）は何個用いてもOK
  - 四則演算「+」「-」「*」「/」は各0個以上使用可
    - 2回以上使ってもOK
    - 使わなくてもOK
  - 「1」「2」「2」「3」の例：Debug.Log((3 + (1 * 2)) * 2);
- 余力があれば「3」「4」「7」「8」にも挑戦

プログラミング基礎(C/C++言語)履修者向け

C#プログラミング「変数と演算」

変数と代入

```
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour
{
    void Start() {
        int x = 0;
        x = 10 * 2;
        int y = x;
        Debug.Log(x + " " + y);
    }
}
```

注意：・全て半角文字で入力すること
・大文字／小文字は区別されるので間違えないように
・プログラム文の最後のセミコロン「;」を忘れずに

文法図解1

```
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour
{
    void Start() {
        int x = 0;
        x = 10 * 2;
        int y = 5;
        Debug.Log(x + " " + y);
    }
}
```

型名

初期値

変数名（プログラマが指定）

変数の宣言

- 指定された型の変数を，変数名「x」で利用開始
- 初期値0は省略可（「int x;」でもOK）

文法図解2

```
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour
{
    void Start() {
        int x = 0;
        x = 10 * 2;
        int y = x;
        Debug.Log(x + " " + y);
    }
}
```

②　①

変数への代入
2-1. 右辺の計算式の実行
2-2. 計算結果 20 によって
左辺の変数 x を上書き

文法図解3

```
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour
{
    void Start()
    {
        int x = 20;
        x = 10;
        int y = x;
        Debug.Log(x + " " + y);
    }
}
```

変数宣言時の代入
3-1. 右辺の変数 x から数値 20 を取り出す
3-2. 左辺の変数 y を宣言し、
3-3. 取り出した右辺の数値 20 を y に代入

② ③ ①

変数の型

https://docs.microsoft.com/ja-jp/dotnet/csharp/language-reference/keywords/types

- 整数型　　int
- 浮動小数点型　　float, double
- ブール型　　bool
  - その他 short, long, byte, uint, など

- 文字列型　　string
- ユーザー定義型クラス

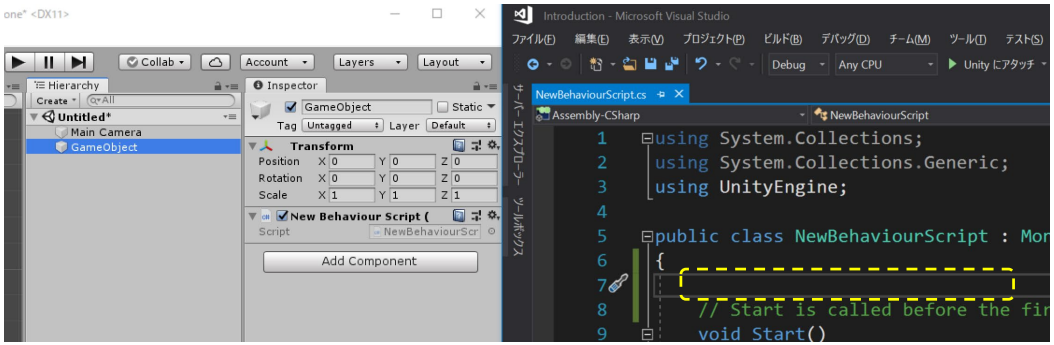
クイズ： 出力結果は？

```
using UnityEngine;
public class NewBehaviourScript :
MonoBehaviour {
    void Start() {
        int x = 0;
        float y = 10;
        x = 10 * 2;
        y = x / 3;
        Debug.Log(x + " " + y);
    }
}
```

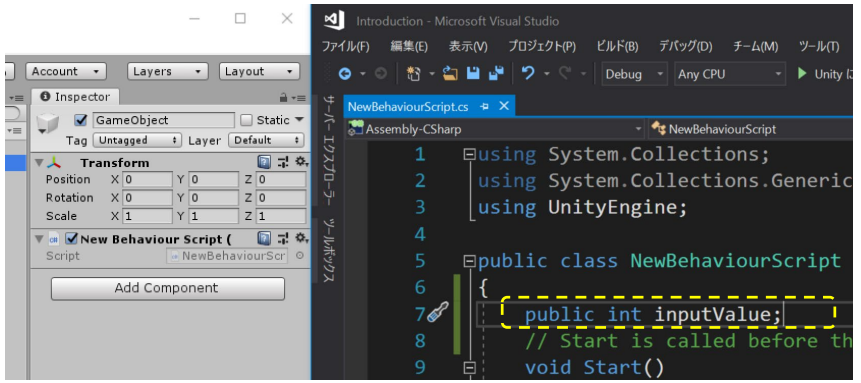
UNITY C#プログラミング「入力」

Unityからの入力

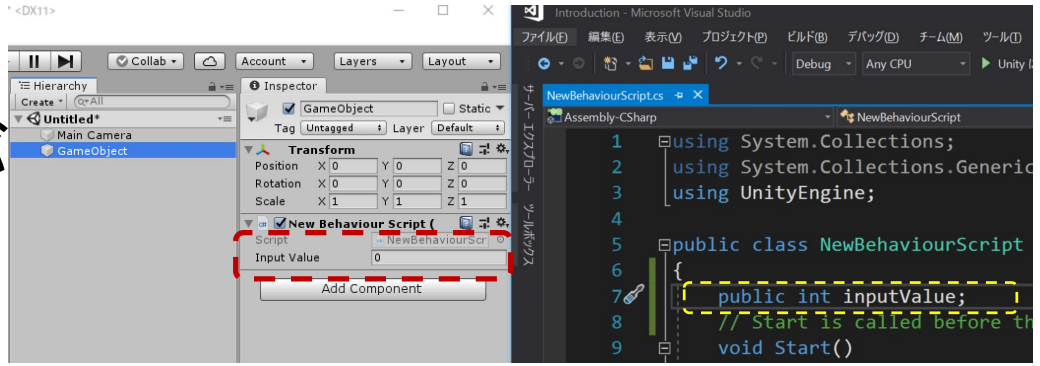
1. スクリプトクラスの上部に



2. public int inputValue;
と記入して、保存

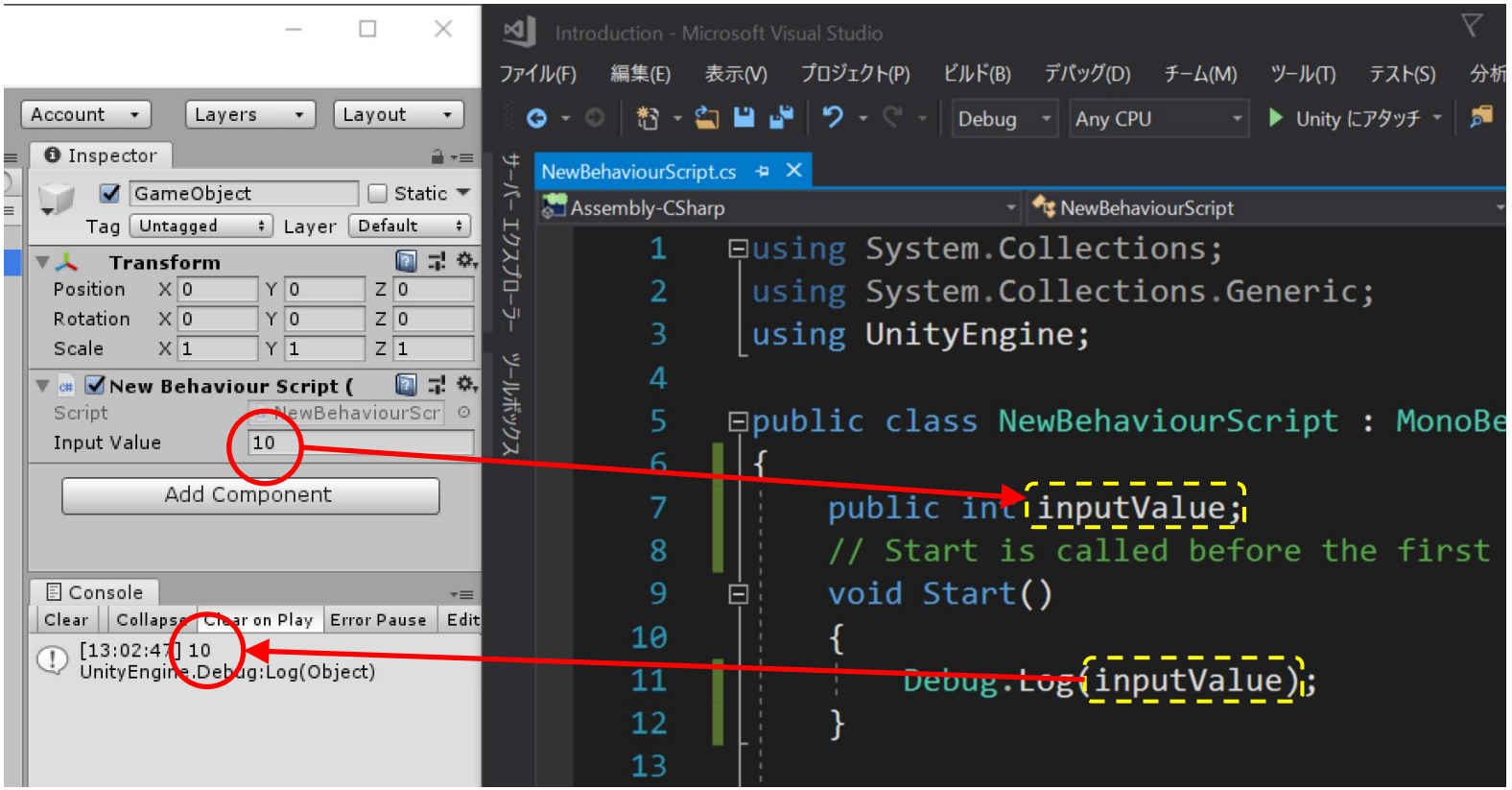


3. Unityに戻って
しばらく待つと、
「Input Value」欄が
追加される



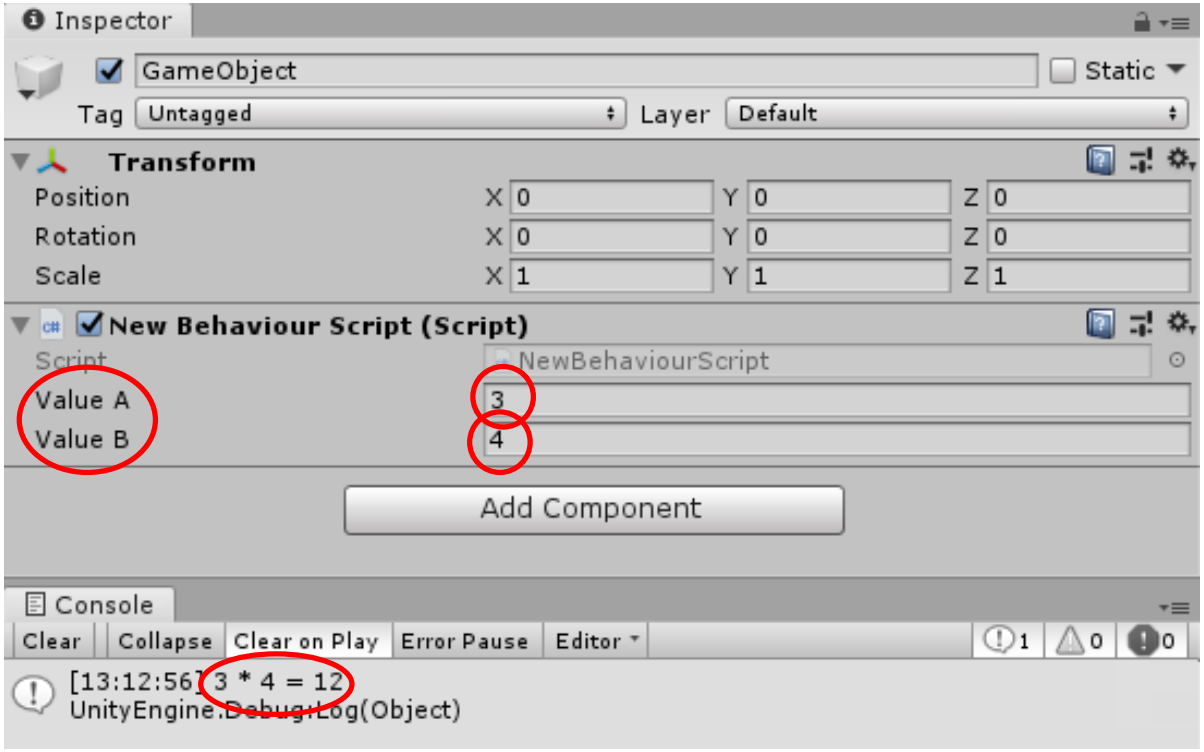
Unityからの入力 contd.

4. 「Input Value」に入力した数値はスクリプト実行時に inputValue に代入される



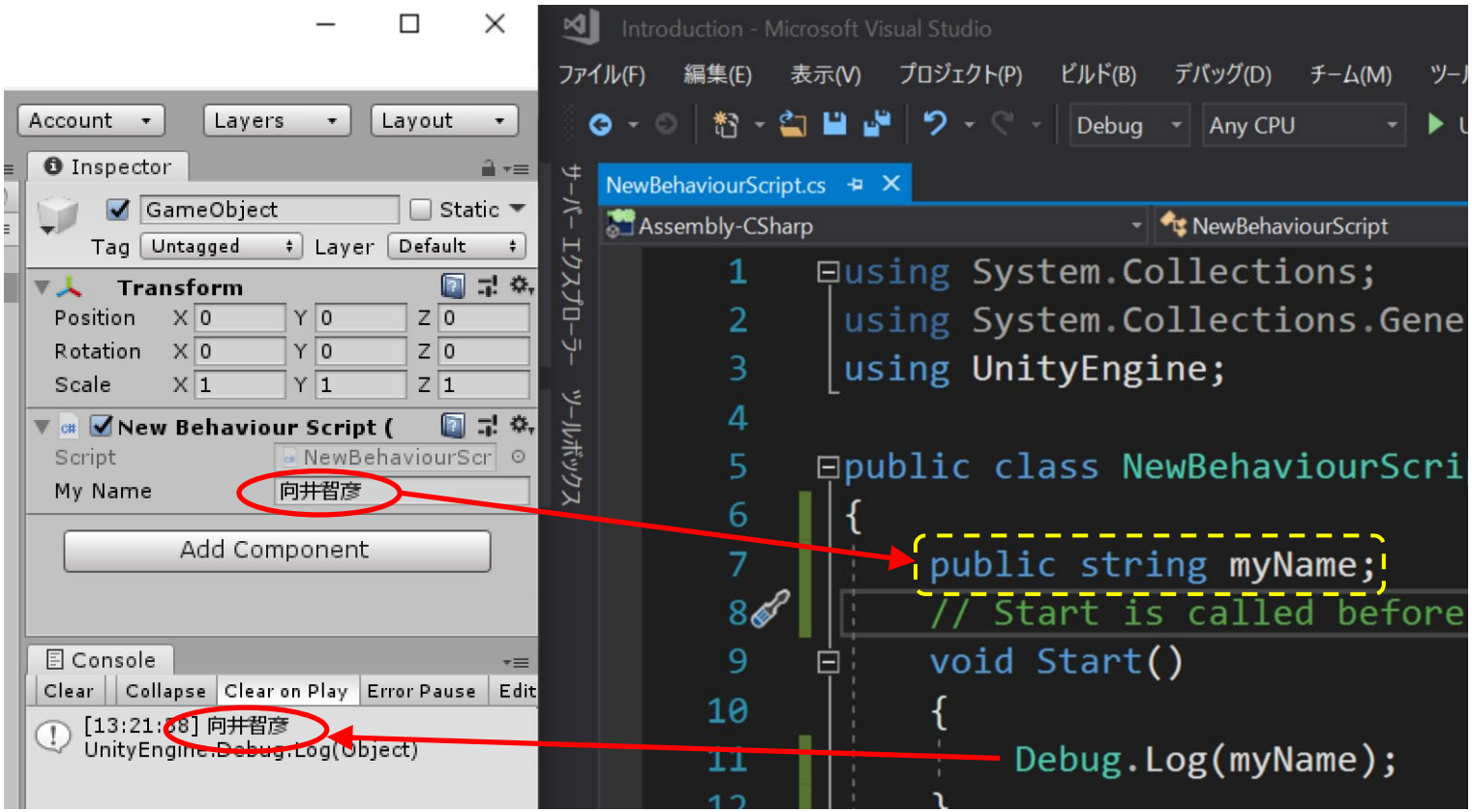
練習問題

- 下図の通り、かけ算を行うC#スクリプトを開発
  - 入力欄は「Value A」と「Value B」（いずれもint）
  - 出力は「*」「=」の前後それぞれに半角空白挿入



補足:文字列の入出力

- string型変数を介して日本語を扱える



プログラミング基礎(C/C++言語)履修者向け

C#プログラミング「条件分岐」

条件分岐 if & else

- 真偽値true/falseに応じて異なる処理を行う，処理分岐のための制御構文

```
void Start {
  int x = 5, y = 6;
  if (x > y) {
    Debug.Log(x + " is greater than " + y);
  }
  else {
    Debug.Log(x + " is smaller than " + y);
  }
}
```

条件式：真偽値：比較演算＆論理演算

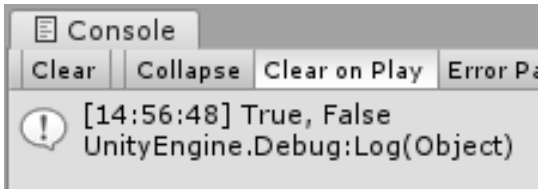
条件式がtrueの時に実行される文

条件式がfalseのときに実行される文

ブール（bool）型

- 真理値 とか 真偽値，ブーリアン型とも
  - 真：true あるいは偽：false
  - true の否定 は false，falseの否定はtrue
    - 数学「命題・論理」と関係

```
bool a, b;
a = true;
b = !a; //否定演算
Debug.Log(a + ", " + b);
```



比較演算：演算結果はbool型

※以下，a と b は任意の型（int, char, double, string,･･･）

- 一致「 a == b 」（代入「 = 」と混同しないよう注意）
  - a と b が同じ値であれば true, 違うなら false
- 非一致「 a != b」
  - a と b が異なる値であれば true，違うなら false
- 大なり「 a > b」，以上「 a >= b」
  - a が b より大きい値ならtrue，小さいなら false
- 未満「 a < b」，以下「 a <= b」
  - a が b より小さい値ならtrue，大きいなら false

真偽値の論理演算

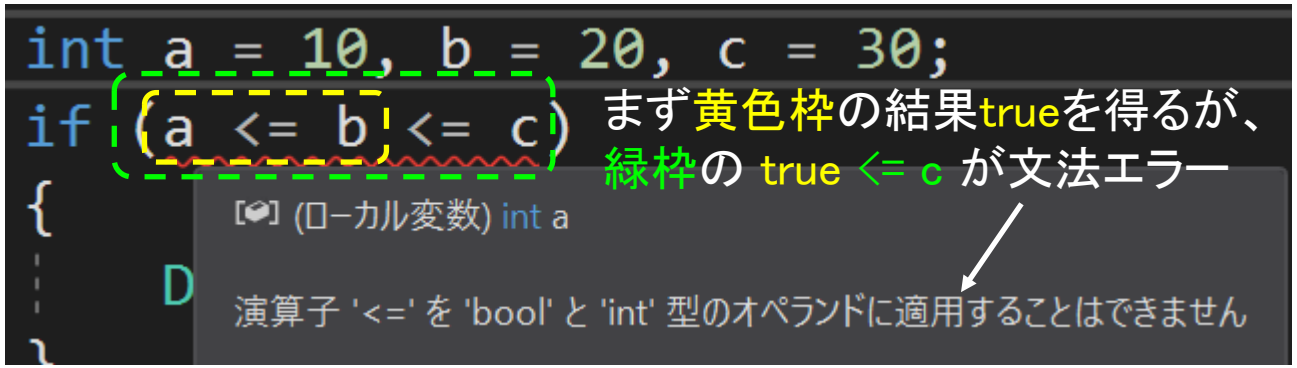
※ a と b はどちらもbool型

- 否定「 !a 」「 !b 」
  - true の否定は false，false の否定は true
- 論理和「 a || b 」
  - a と b のいずれか true なら，結果は true
  - a と b の両方が false の時，結果は false
- 論理積「 a && b 」
  - a と b の両方が true なら，結果は true
  - a と b のどちらかが false なら，結果は false

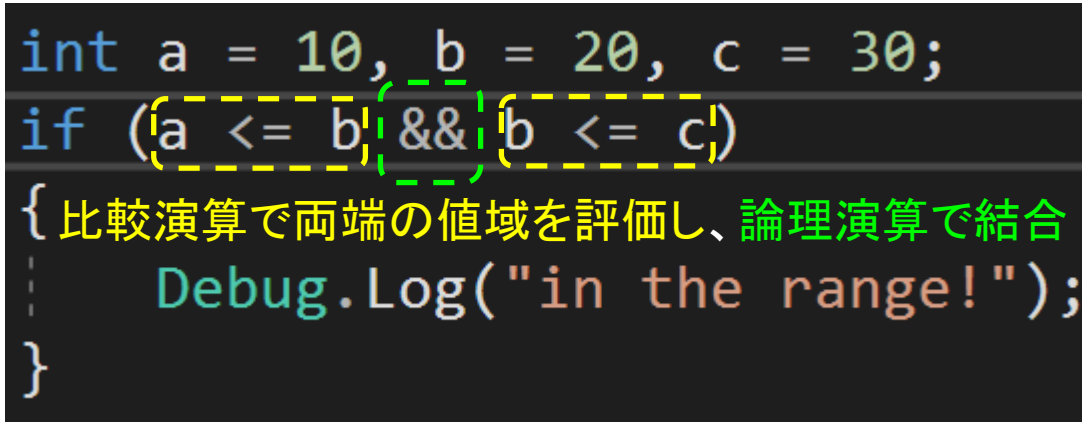
演算子の原則:
演算対象の数値は1つ(単項)か2つ

https://docs.microsoft.com/ja-jp/dotnet/csharp/language-reference/operators/

誤）



正）



本日のまとめ

- 授業ガイダンス
  - Unity C#スクリプト
  - 各回70%＋最終課題30%
- Unity & C# 導入
  - Roll-a-Ball
  - 演算、入出力、変数、条件分岐

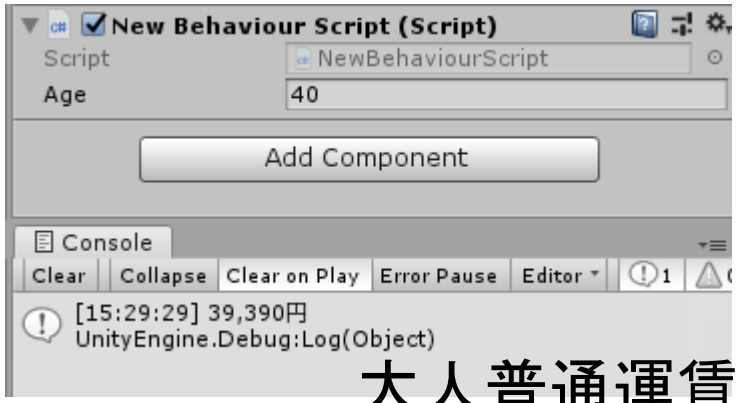
本日の課題1：航空機運賃計算

※提出方法・期限の説明は来週

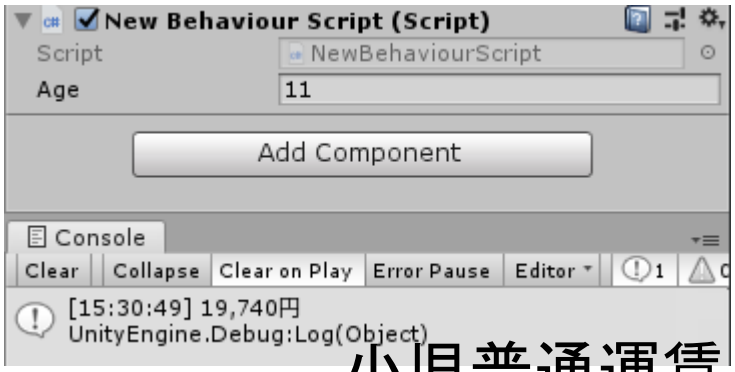
前提：　今年の5月5日朝に東京から長崎まで航空機利用で帰省します。ある運航会社の大人普通運賃は39,390 円ですが、年齢によってさまざまな割引があります。まず、満3歳以上満12歳未満の小児運賃は19,740 円です。満3歳未満の幼児は無料です。また、65歳以上は15,290 円、満12歳以上満22歳未満も15,290 円です。

課題：　入力された年齢に応じて運賃を計算するC#スクリプトを作成

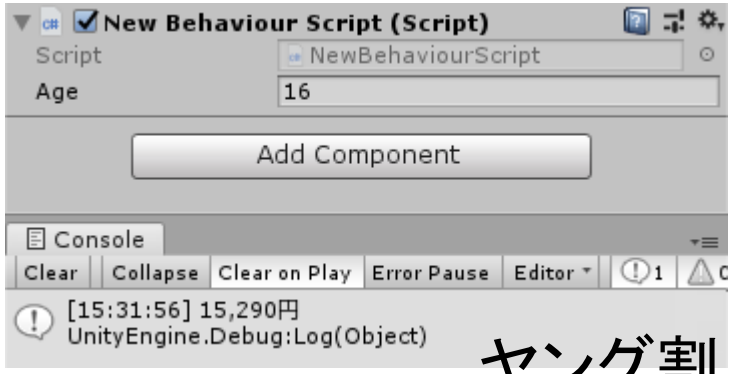
実行結果例



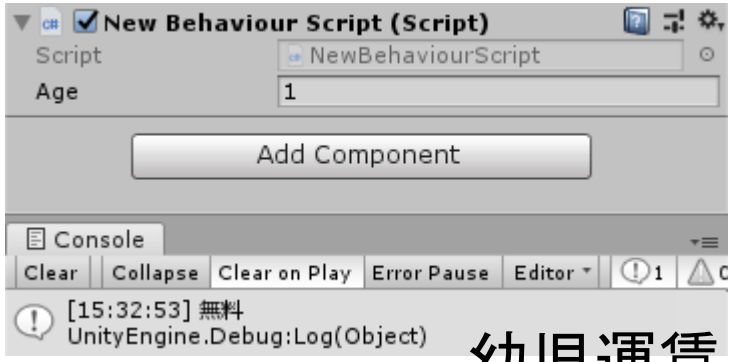
大人普通運賃



小児普通運賃



ヤング割



幼児運賃

本日説明しなかったこと

- class
  - Debugクラス
  - MonoBehaviourクラス
- public
- StartメソッドとUpdateメソッド

来週の予告

- C#プログラミング 配列
- C#プログラミング 反復
- C#プログラミング メソッド
- Unity C#プログラミング Mathfクラス