

# 2020年度 Unity講座(基礎編)



04回目

講師：幸田 将伍 (@MagurodonDev)

# 今回の講義の目的

2

- ▶ 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

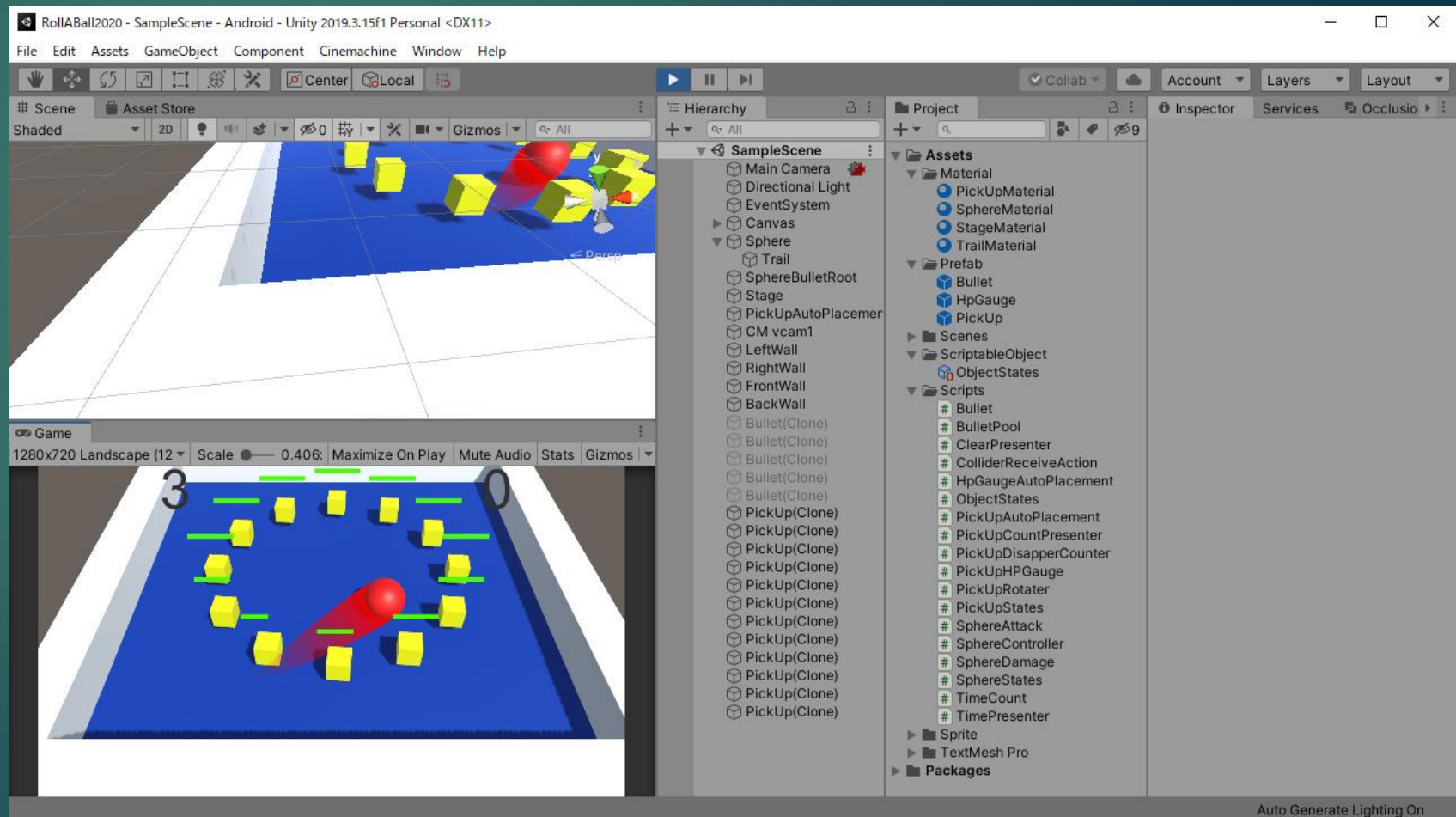
一緒にレベルアップして行きましょう！

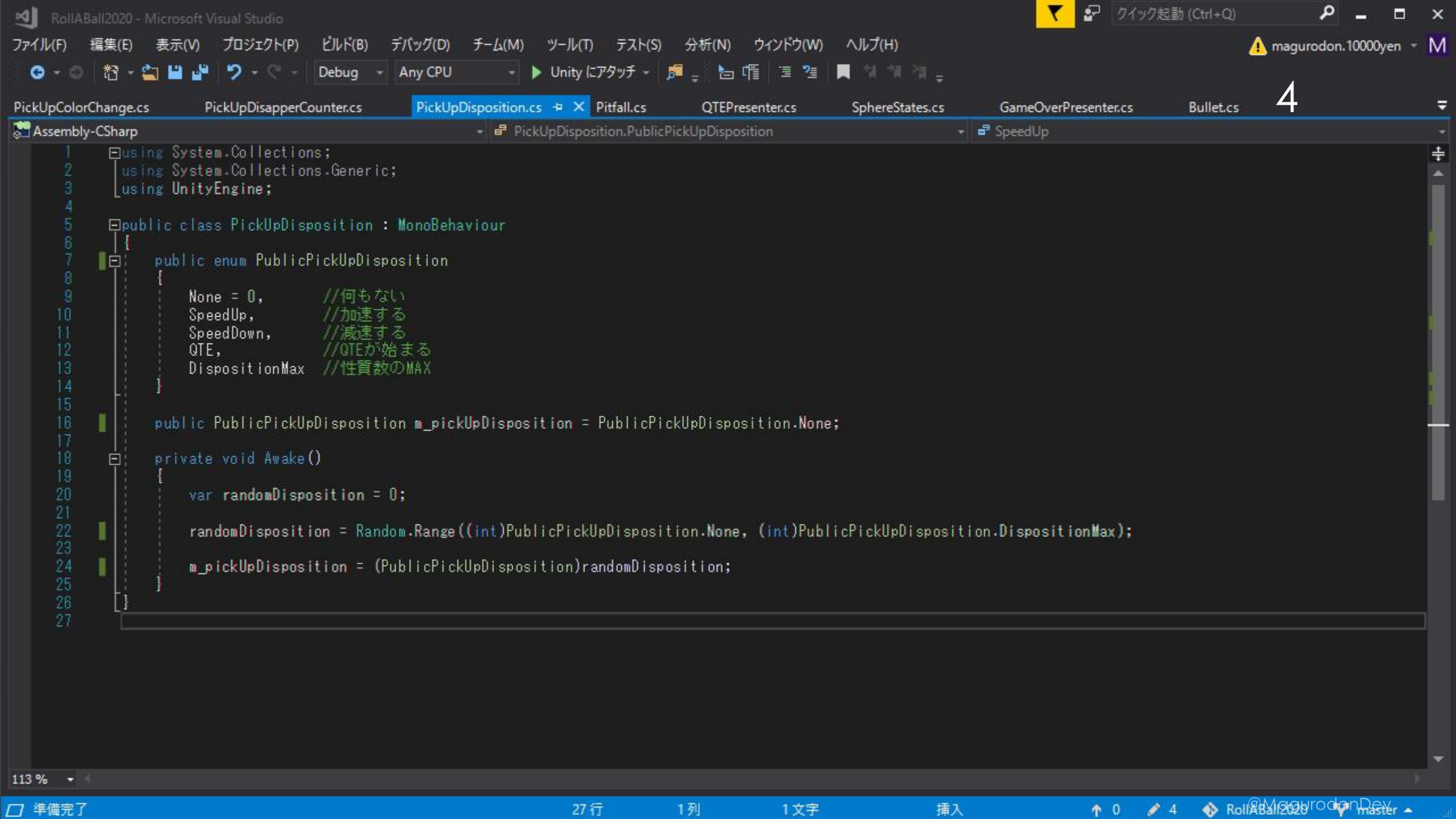
# Unity

3

## RollABall

- 前はSphere側の仕様は全て満たしました
- 今回はPickUp側の仕様を完成させていきましょう
- ランダムで自分の色を変え、その色によって特定の効果を与える
- →QTEが始まったり(2秒以内に特定の文字を入力しろ的な)
- →加速したり、減速したり
- ランダムで自分の色を変え、その色によって得点が変わる
- このあたりを実装していきます
- PickUpDispositionを作成します





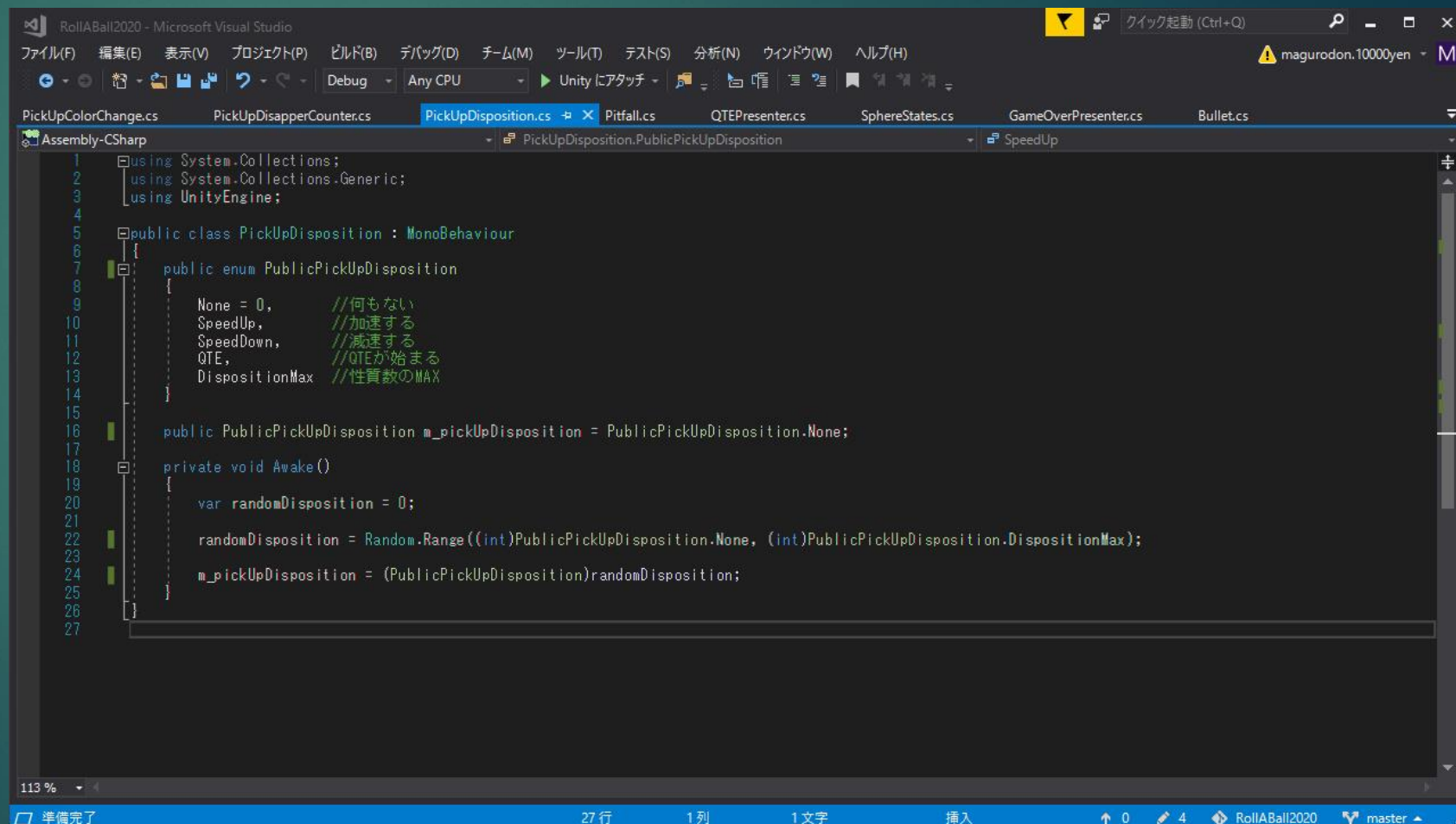


# Unity

## RollABall

5

- ※Random.Range(Min,max)
- ランダムはゲームの世界でとてもよく使われます
- この関数の取り扱いですが少し注意をしなければいけません
- Int型の場合は、Minは大丈夫なのですがMaxが指定した値の-1になります
- Float型の場合はそんなことはないので気をつけましょう
- 今回はenumでPickupのもつ性質を、ゲーム開始時にランダムで決めています



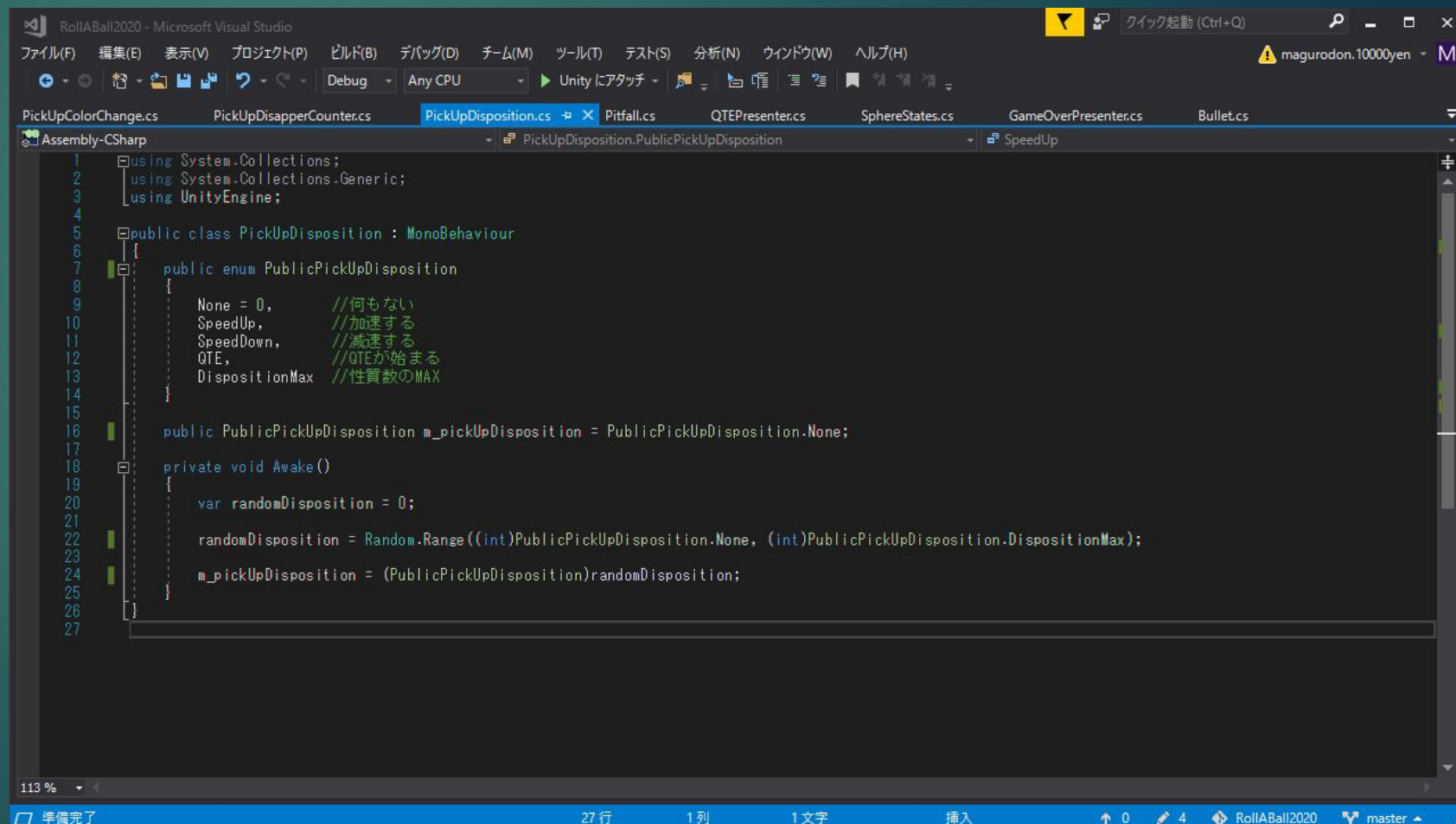
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PickUpDisposition : MonoBehaviour
6 {
7     public enum PublicPickUpDisposition
8     {
9         None = 0,           //何もない
10        SpeedUp,            //加速する
11        SpeedDown,          //減速する
12        QTE,                //QTEが始まる
13        DispositionMax       //性質数のMAX
14    }
15
16    public PublicPickUpDisposition m_pickUpDisposition = PublicPickUpDisposition.None;
17
18    private void Awake()
19    {
20        var randomDisposition = 0;
21
22        randomDisposition = Random.Range((int)PublicPickUpDisposition.None, (int)PublicPickUpDisposition.DispositionMax);
23
24        m_pickUpDisposition = (PublicPickUpDisposition)randomDisposition;
25    }
26
27 }
```

# Unity

## RollABall

6

- ※(Hogehoge)hogehoge
- 値を違う型に変換したいときにつかわれます
- 今回でいうと  
(int)PublicPickupDisposition.Noneは0になりますし、例えば  
(PublicPickupDisposition)2は  
PublicPickupDisposition.SpeedDownになります
- ここでクイズです
- (int)  
PublicPickupDisposition.QTEは  
何になるでしょうか
- こういったことを、キャストする、やコンバートする、等言います



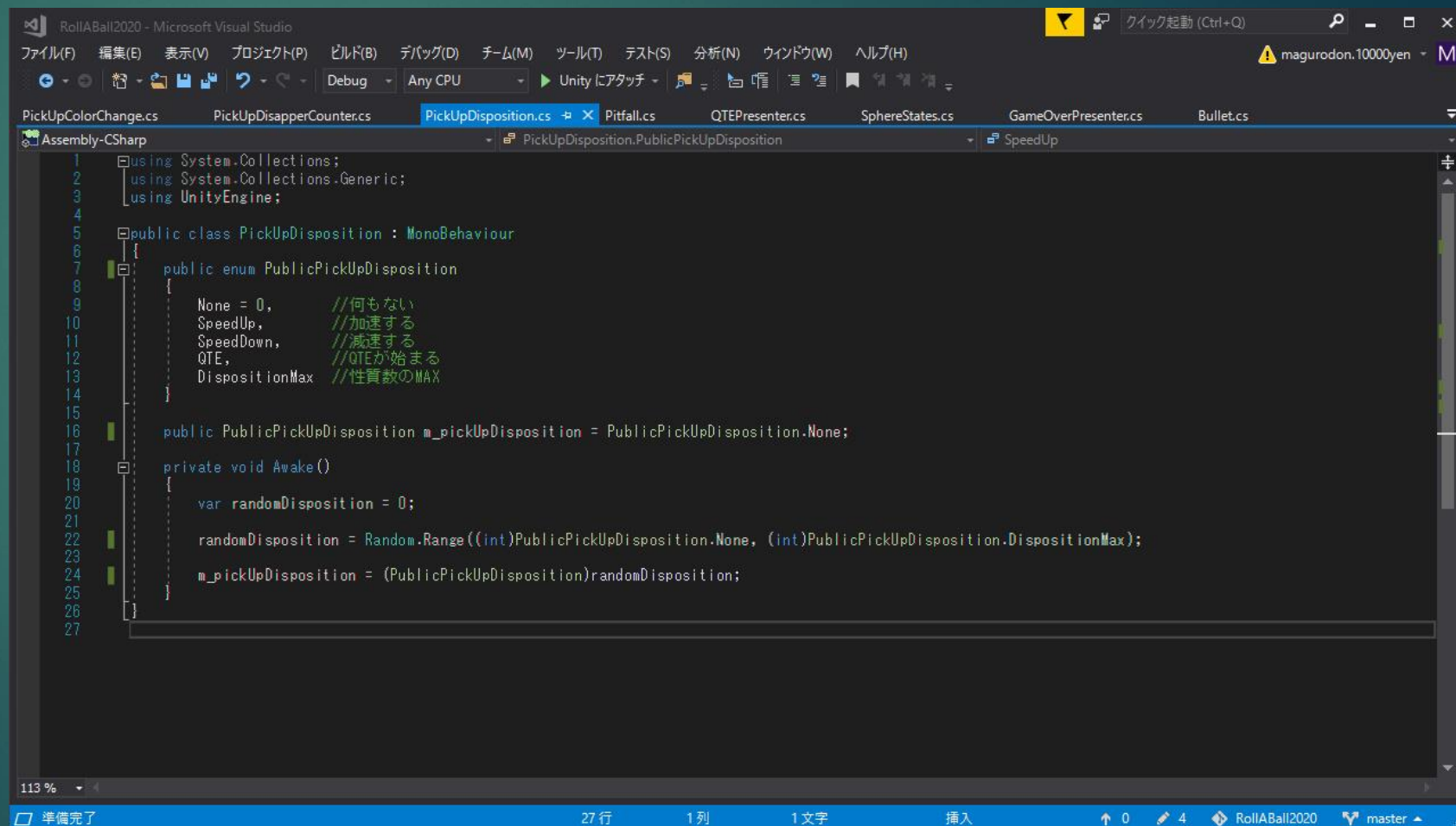
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PickUpDisposition : MonoBehaviour
6 {
7     public enum PublicPickupDisposition
8     {
9         None = 0,           //何もない
10        SpeedUp,            //加速する
11        SpeedDown,          //減速する
12        QTE,                //QTEが始まる
13        DispositionMax       //性質数のMAX
14    }
15
16    public PublicPickupDisposition m_pickUpDisposition = PublicPickupDisposition.None;
17
18    private void Awake()
19    {
20        var randomDisposition = 0;
21
22        randomDisposition = Random.Range((int)PublicPickupDisposition.None, (int)PublicPickupDisposition.DispositionMax);
23
24        m_pickUpDisposition = (PublicPickupDisposition)randomDisposition;
25    }
26
27 }
```

# Unity

## RollABall

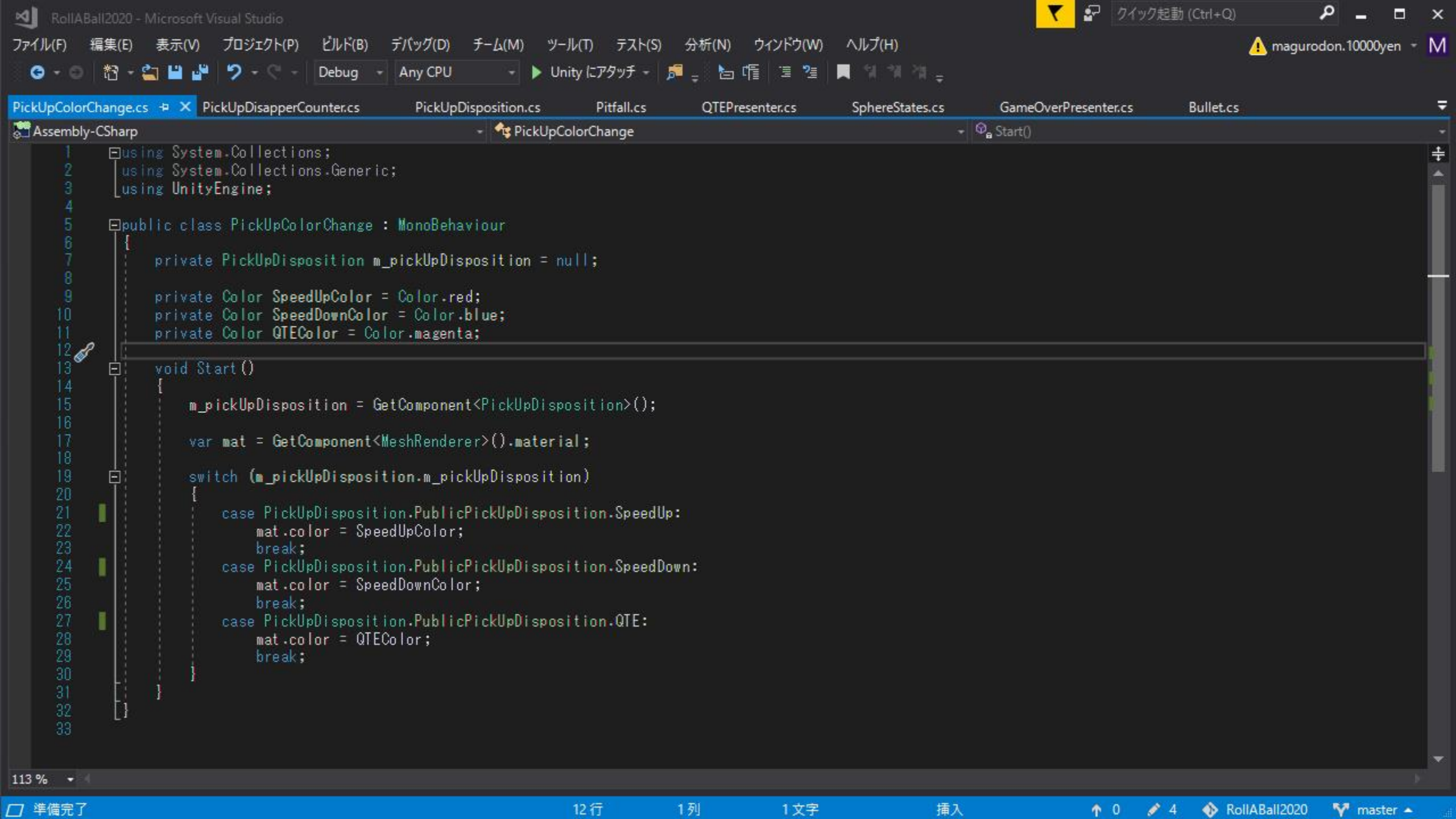
7

- さて、次にPickUpDispositionによって変化を起こすスクリプトを書いていきます
- PickUpColorChangeとPickUpDispositionEffectを作成しましょう



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PickUpDisposition : MonoBehaviour
6 {
7     public enum PublicPickUpDisposition
8     {
9         None = 0,           //何もない
10        SpeedUp,            //加速する
11        SpeedDown,          //減速する
12        QTE,                 //QTEが始まる
13        DispositionMax       //性質数のMAX
14    }
15
16    public PublicPickUpDisposition m_pickUpDisposition = PublicPickUpDisposition.None;
17
18    private void Awake()
19    {
20        var randomDisposition = 0;
21
22        randomDisposition = Random.Range((int)PublicPickUpDisposition.None, (int)PublicPickUpDisposition.DispositionMax);
23
24        m_pickUpDisposition = (PublicPickUpDisposition)randomDisposition;
25    }
26
27 }
```



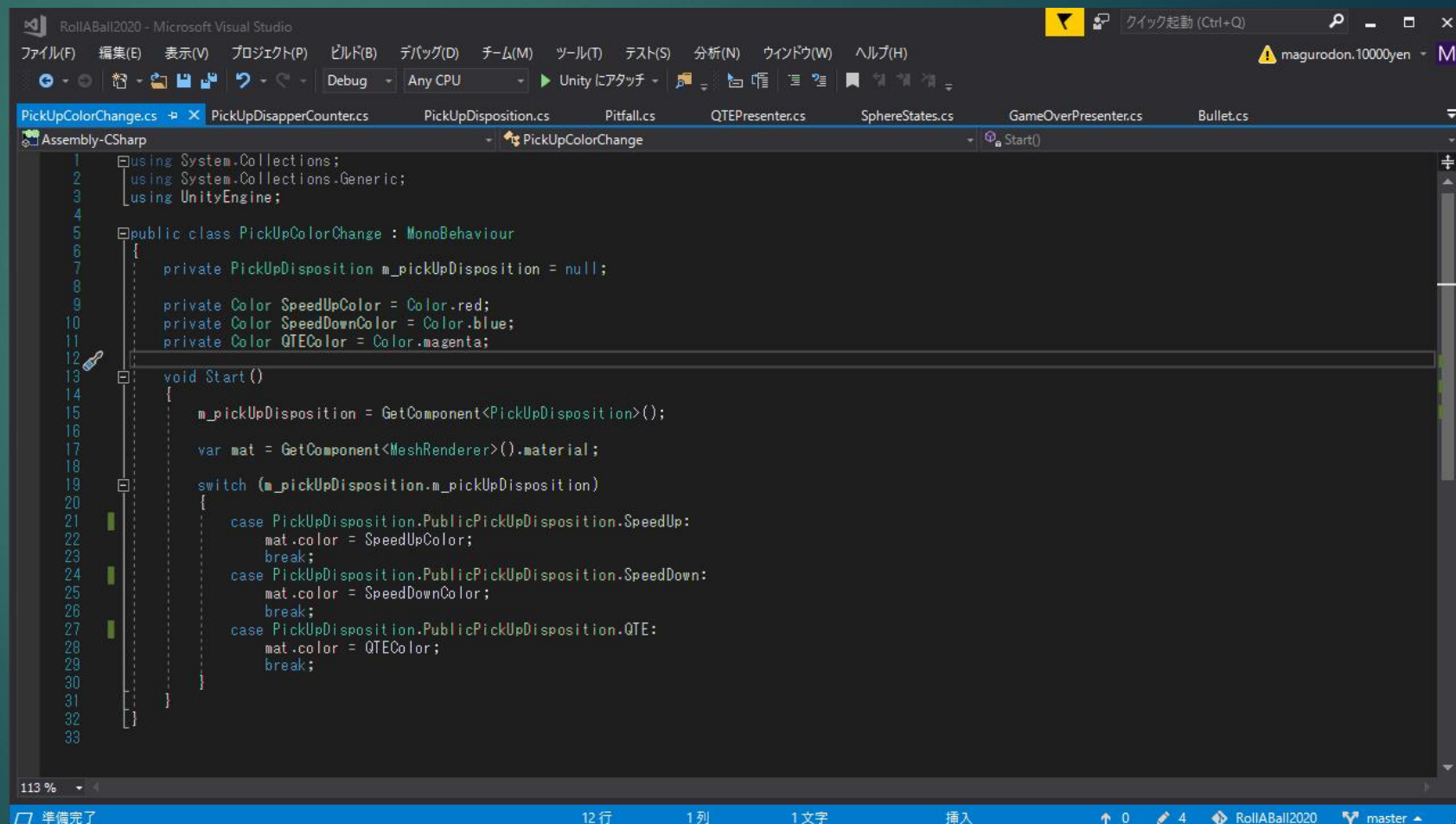




```
1 using UnityEngine;
2
3 public class PickupDispositionEffect : MonoBehaviour
4 {
5     private PickupDisposition m_pickUpDisposition = null;
6
7     private QTEPresenter m_QTEPresenter = null;
8
9     private void Start()
10    {
11        m_pickUpDisposition = GetComponent<PickupDisposition>();
12    }
13
14    private void OnTriggerEnter(Collider other)
15    {
16        if (other.CompareTag("Player"))
17        {
18            DispositionEffect(other);
19        }
20    }
21
22    private void DispositionEffect(Collider other)
23    {
24        var vec = Vector3.Normalize(transform.position - other.transform.position);
25
26        switch (m_pickUpDisposition.m_pickUpDisposition)
27        {
28            case PickupDisposition.PublicPickUpDisposition.SpeedUp:
29                other.GetComponent<Rigidbody>().AddForce(vec * 5f, ForceMode.Impulse);
30                break;
31            case PickupDisposition.PublicPickUpDisposition.SpeedDown:
32                other.GetComponent<Rigidbody>().AddForce(vec * -100f, ForceMode.Acceleration);
33                break;
34        }
35    }
36 }
```

## RollABall

- ※Switch文について
- 今回は新しい文法、Switch文について解説していきます
- Switch文とは一つの値に対して条件を分岐する場合にif文の代わりに使われる文法です
- 今まではif文によって、条件を分岐してきました
- ただし、if文はもし～であれば、という意味を、同じ値に対して基本的には一つしか持てません
- 無理をしてif文を重ねることもできるのですが、可読性が高くないので、一つの値に対して条件を分岐する場合はSwitch文などを使うと見やすく書けます



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PickUpColorChange : MonoBehaviour
6 {
7     private PickupDisposition m_pickUpDisposition = null;
8
9     private Color SpeedUpColor = Color.red;
10    private Color SpeedDownColor = Color.blue;
11    private Color QTEColor = Color.magenta;
12
13    void Start()
14    {
15        m_pickUpDisposition = GetComponent<PickUpDisposition>();
16
17        var mat = GetComponent<MeshRenderer>().material;
18
19        switch (m_pickUpDisposition.m_pickUpDisposition)
20        {
21            case PickupDisposition.PublicPickUpDisposition.SpeedUp:
22                mat.color = SpeedUpColor;
23                break;
24            case PickupDisposition.PublicPickUpDisposition.SpeedDown:
25                mat.color = SpeedDownColor;
26                break;
27            case PickupDisposition.PublicPickUpDisposition.QTE:
28                mat.color = QTEColor;
29                break;
30        }
31    }
32
33 }
```

# Unity

## RollABall

11

- ※Break;
- Switch文の中でよく出てきます
- これがでてきた場合、CPUはこのSwitch文の中の処理を止めて次の処理に向かいます
- その処理の流れを壊して(Break)して外に出るというイメージで覚えましょう

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PickUpColorChange : MonoBehaviour
6 {
7     private PickUpDisposition m_pickUpDisposition = null;
8
9     private Color SpeedUpColor = Color.red;
10    private Color SpeedDownColor = Color.blue;
11    private Color QTEColor = Color.magenta;
12
13    void Start()
14    {
15        m_pickUpDisposition = GetComponent<PickUpDisposition>();
16        var mat = GetComponent<MeshRenderer>().material;
17
18        switch (m_pickUpDisposition.m_pickUpDisposition)
19        {
20            case PickUpDisposition.PublicPickUpDisposition.SpeedUp:
21                mat.color = SpeedUpColor;
22                break;
23            case PickUpDisposition.PublicPickUpDisposition.SpeedDown:
24                mat.color = SpeedDownColor;
25                break;
26            case PickUpDisposition.PublicPickUpDisposition.QTE:
27                mat.color = QTEColor;
28                break;
29        }
30    }
31
32
33 }
```

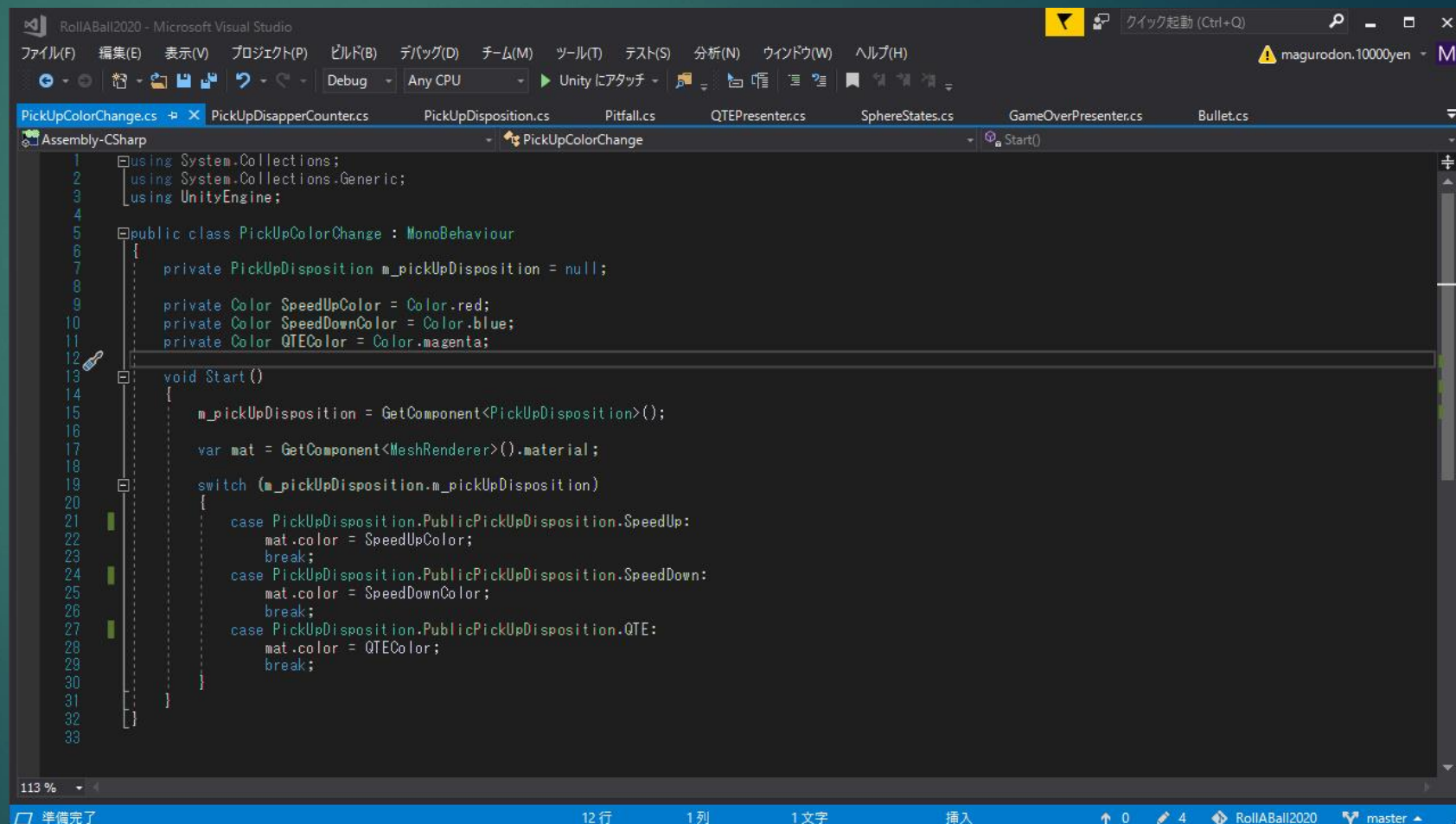


# Unity

12

## RollABall

- ※Color型
- 色もプログラムで制御することが可能です
- 今回はUnity側で用意してあるColor型の定数を使いました
- もちろん、Color型(r,g,b,a)の変数を新しく定義すれば好きな色に変換することも可能です
- 色について少し深くお話しします



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PickUpColorChange : MonoBehaviour
6 {
7     private PickUpDisposition m_pickUpDisposition = null;
8
9     private Color SpeedUpColor = Color.red;
10    private Color SpeedDownColor = Color.blue;
11    private Color QTEColor = Color.magenta;
12
13    void Start()
14    {
15        m_pickUpDisposition = GetComponent<PickUpDisposition>();
16
17        var mat = GetComponent<MeshRenderer>().material;
18
19        switch (m_pickUpDisposition.m_pickUpDisposition)
20        {
21            case PickUpDisposition.PublicPickUpDisposition.SpeedUp:
22                mat.color = SpeedUpColor;
23                break;
24            case PickUpDisposition.PublicPickUpDisposition.SpeedDown:
25                mat.color = SpeedDownColor;
26                break;
27            case PickUpDisposition.PublicPickUpDisposition.QTE:
28                mat.color = QTEColor;
29                break;
30        }
31    }
32
33 }
```

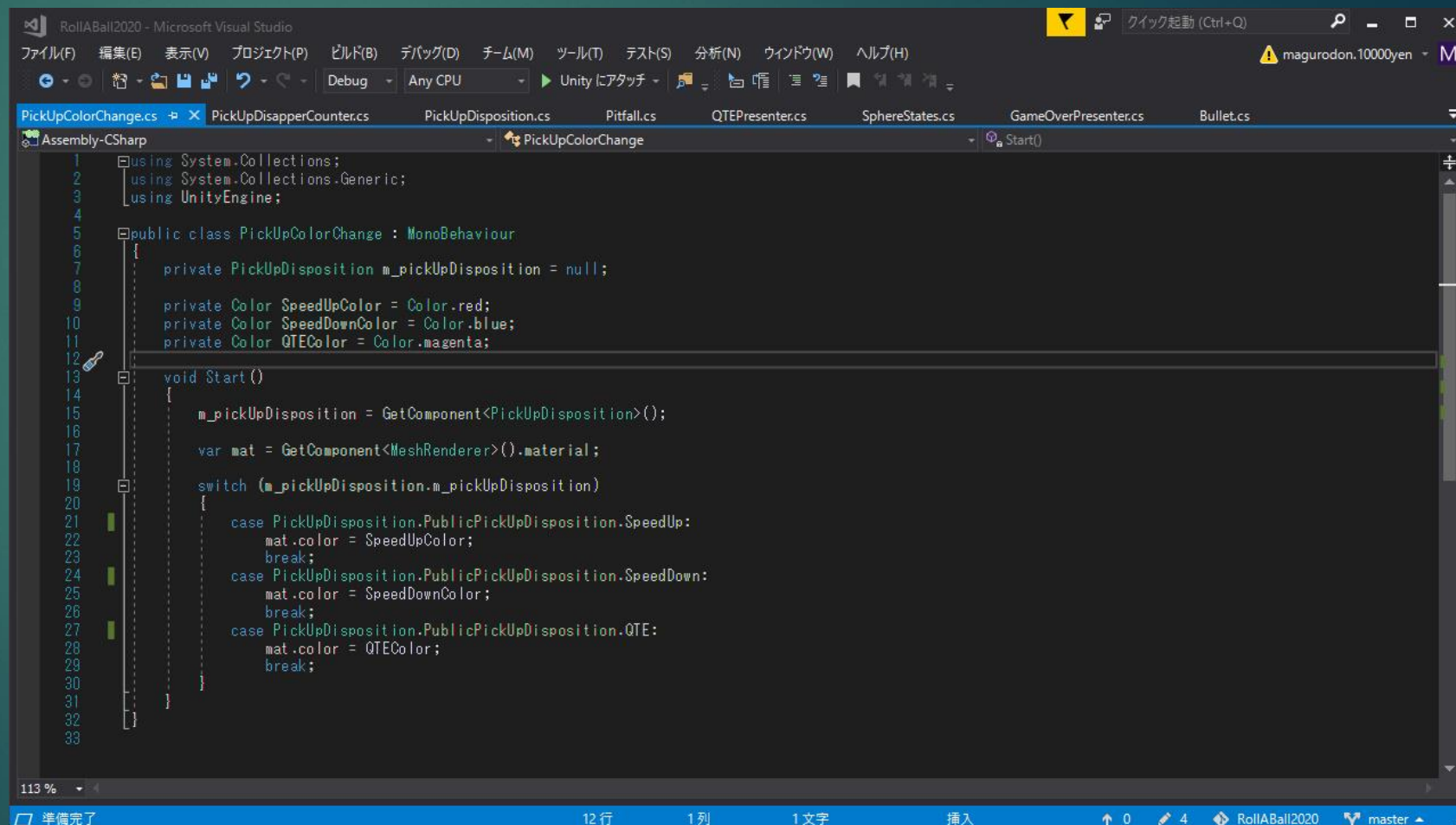


# Unity

13

## RollABall

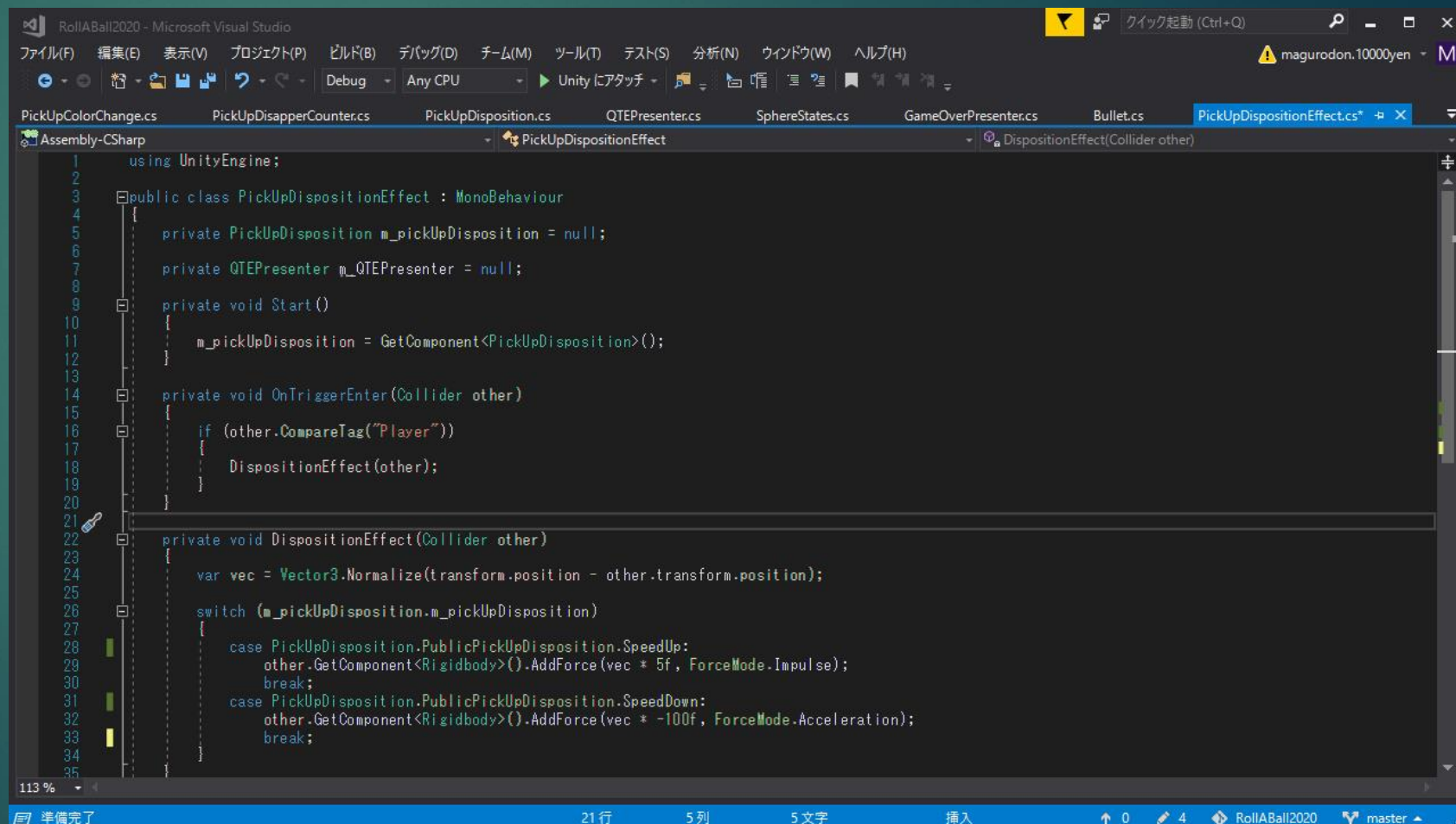
- プログラムにとって色を表現する時に使われる値は、基本的には0~255までの数値をR,G,Bの3つの数値を掛け合わせたものを使います
- 例えばColor(255,0,0)は赤ですし、Color(0,255,0)は緑、Color(0,0,255)は青です
- 何か自分の使いたい色を表現したい場合は、Unityのカラーパレットに表示されているR,G,Bの数値をよく見てみましょう
- その3つの数値をColor(r,g,b)に当てはめれば好きな色に変えることができます



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PickUpColorChange : MonoBehaviour
6 {
7     private PickupDisposition m_pickUpDisposition = null;
8
9     private Color SpeedUpColor = Color.red;
10    private Color SpeedDownColor = Color.blue;
11    private Color QTEColor = Color.magenta;
12
13    void Start()
14    {
15        m_pickUpDisposition = GetComponent<PickUpDisposition>();
16
17        var mat = GetComponent<MeshRenderer>().material;
18
19        switch (m_pickUpDisposition.m_pickUpDisposition)
20        {
21            case PickupDisposition.PublicPickUpDisposition.SpeedUp:
22                mat.color = SpeedUpColor;
23                break;
24            case PickupDisposition.PublicPickUpDisposition.SpeedDown:
25                mat.color = SpeedDownColor;
26                break;
27            case PickupDisposition.PublicPickUpDisposition.QTE:
28                mat.color = QTEColor;
29                break;
30        }
31    }
32
33 }
```

## RollABall

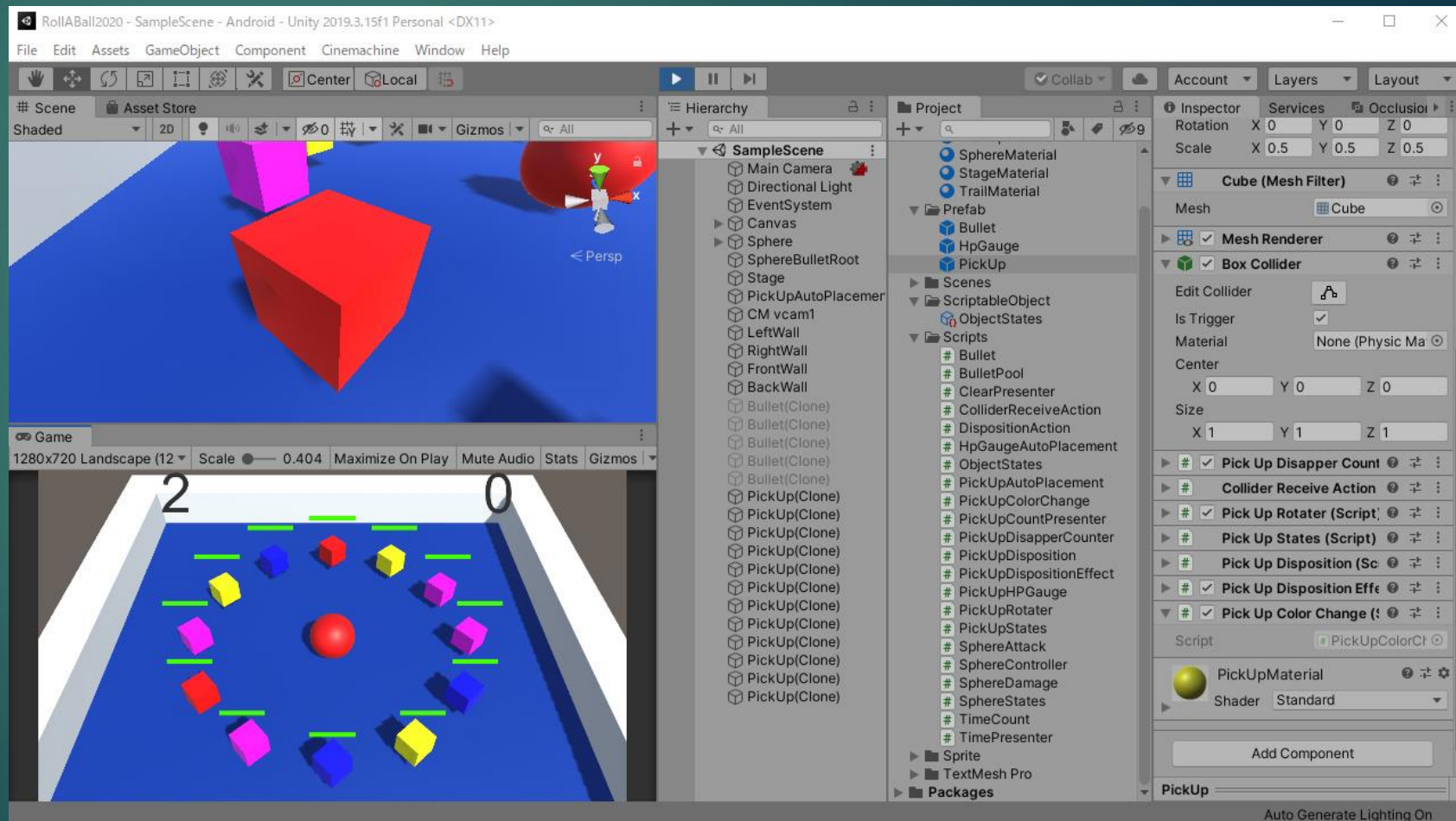
- さてPickUpDispositionEffectについて解説していきます
- ※Vector3.Normalize(Vector3)
- 長さを1.0fとして引数に入れた方向のベクトルを返します
- 今回はPickUpからSphereの距離の差分を出し、それをもとにその方向のベクトルを出しています
- お気づきの方もいるかもしれませんが、QTEの分岐分けがまだないです
- 理由はQTEは毛色の違うゲームシステムを突っ込むことになるので、今は開けておきます



```
1 using UnityEngine;
2
3 public class PickUpDispositionEffect : MonoBehaviour
4 {
5     private PickUpDisposition m_pickUpDisposition = null;
6     private QTEPresenter m_QTEPresenter = null;
7
8     private void Start()
9     {
10         m_pickUpDisposition = GetComponent<PickUpDisposition>();
11     }
12
13     private void OnTriggerEnter(Collider other)
14     {
15         if (other.CompareTag("Player"))
16         {
17             DispositionEffect(other);
18         }
19     }
20
21     private void DispositionEffect(Collider other)
22     {
23         var vec = Vector3.Normalize(transform.position - other.transform.position);
24
25         switch (m_pickUpDisposition.m_pickUpDisposition)
26         {
27             case PickUpDisposition.PublicPickUpDisposition.SpeedUp:
28                 other.GetComponent<Rigidbody>().AddForce(vec * 5f, ForceMode.Impulse);
29                 break;
30             case PickUpDisposition.PublicPickUpDisposition.SpeedDown:
31                 other.GetComponent<Rigidbody>().AddForce(vec * -100f, ForceMode.Acceleration);
32                 break;
33         }
34     }
35 }
```

## RollABall

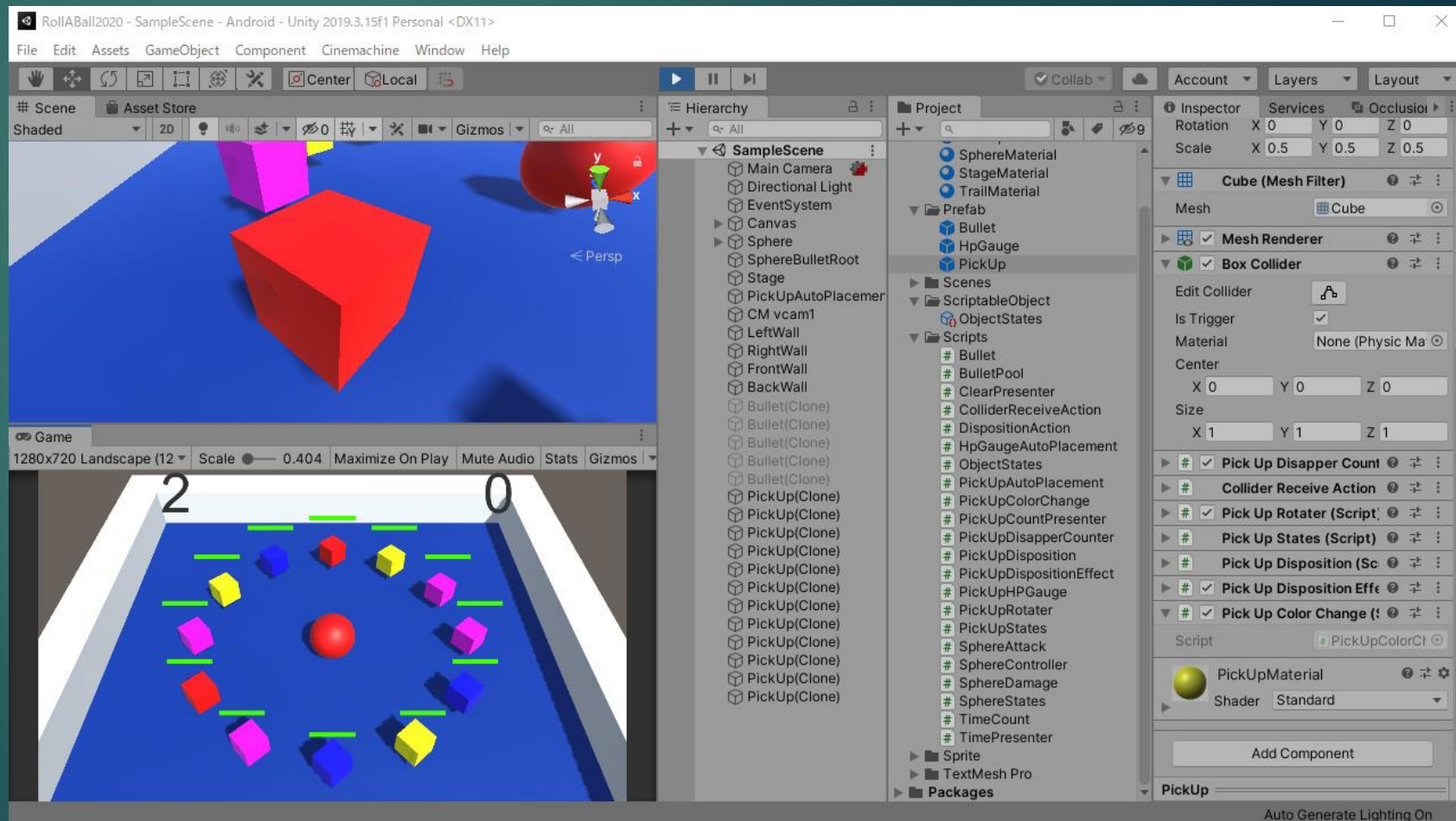
- Unityに戻って確認します
- Prefabフォルダ内にあるPickUpに今回作成したPickupColorChange, PickupDisposition, PickupDispositionをD&Dしていきます
- プレイして確認してみましょう
- 右図のようにカラフルになれば成功です
- さて、それではQTEを実装していきましょう





## RollABall

- QTEとはクイックタイムイベントの略です
- このゲームシステムを使っているゲームは有名どころでいうと「龍が如く」シリーズや「バイオハザード」シリーズ等があげられます
- 時間制限付きで、プレイヤーに入力させ、その可否でゲームに影響を与えるイベントです
- ゲームデザインとしては賛否両論あるシステムですので、利用は計画的に行ってください
- さて、QTEPresenterを作成してください
- SphereDamageも修正していきます

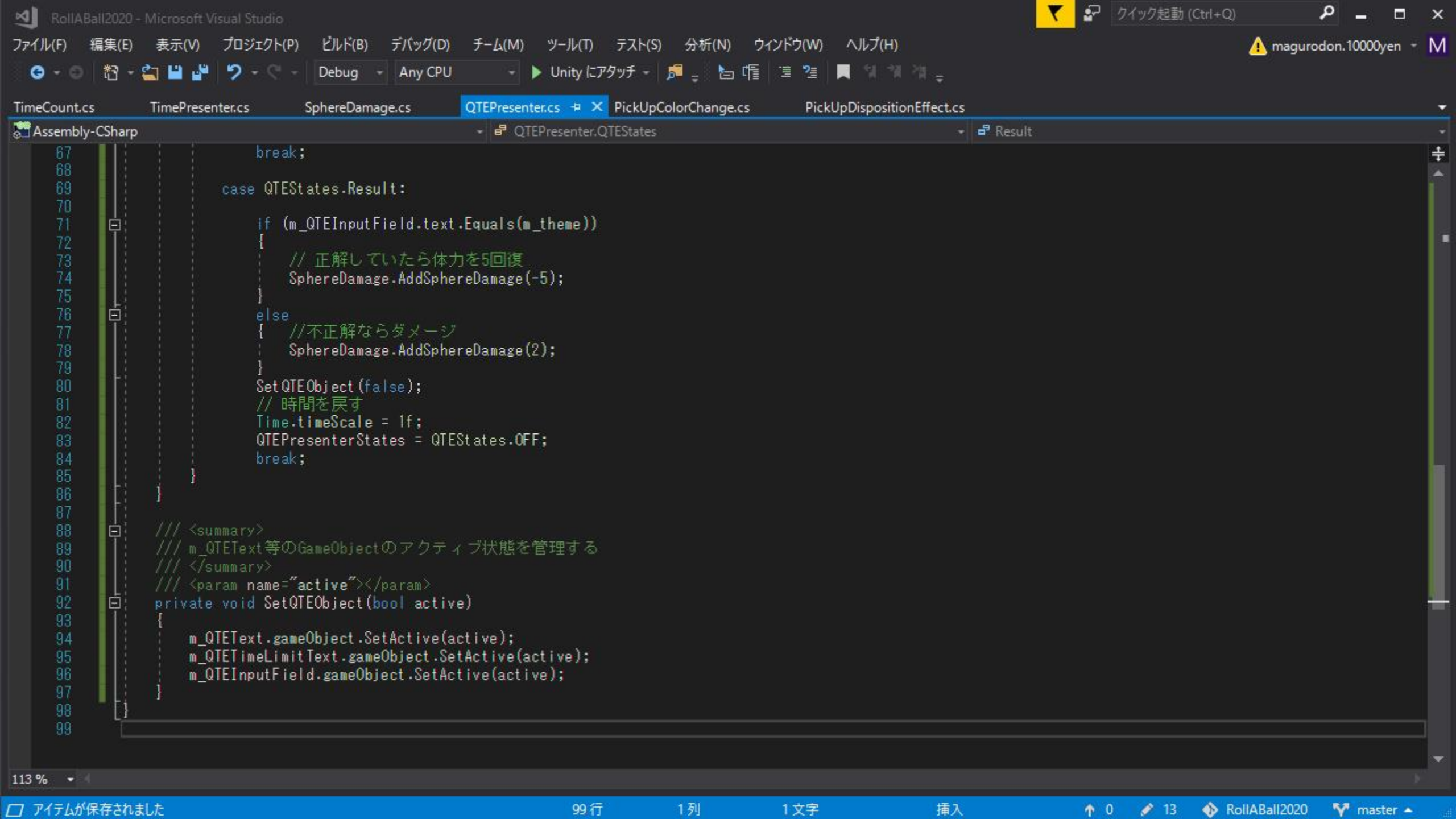




17

```
1 using UnityEngine;
2 using UnityEngine.UI;
3
4 public class QTEPresenter : MonoBehaviour
5 {
6     [SerializeField] private Text m_QTEText = null;
7
8     [SerializeField] private Text m_QTETimeLimitText = null;
9
10    [SerializeField] private InputField m_QTEInputField = null;
11
12    [SerializeField] private float m_timeLimit = 5f;
13
14    [SerializeField] private string m_theme = "SUSHI";
15
16    public SphereDamage SphereDamage = null;
17
18    public enum QTEStates
19    {
20        OFF = 0,
21        ON,
22        Input,
23        Result,
24    }
25
26    public QTEStates QTEPresenterStates = QTEStates.OFF;
27
28    private void Start()
29    {
30        SetQTEObject(false);
31    }
32
33
34    private void Update()
35    {
```

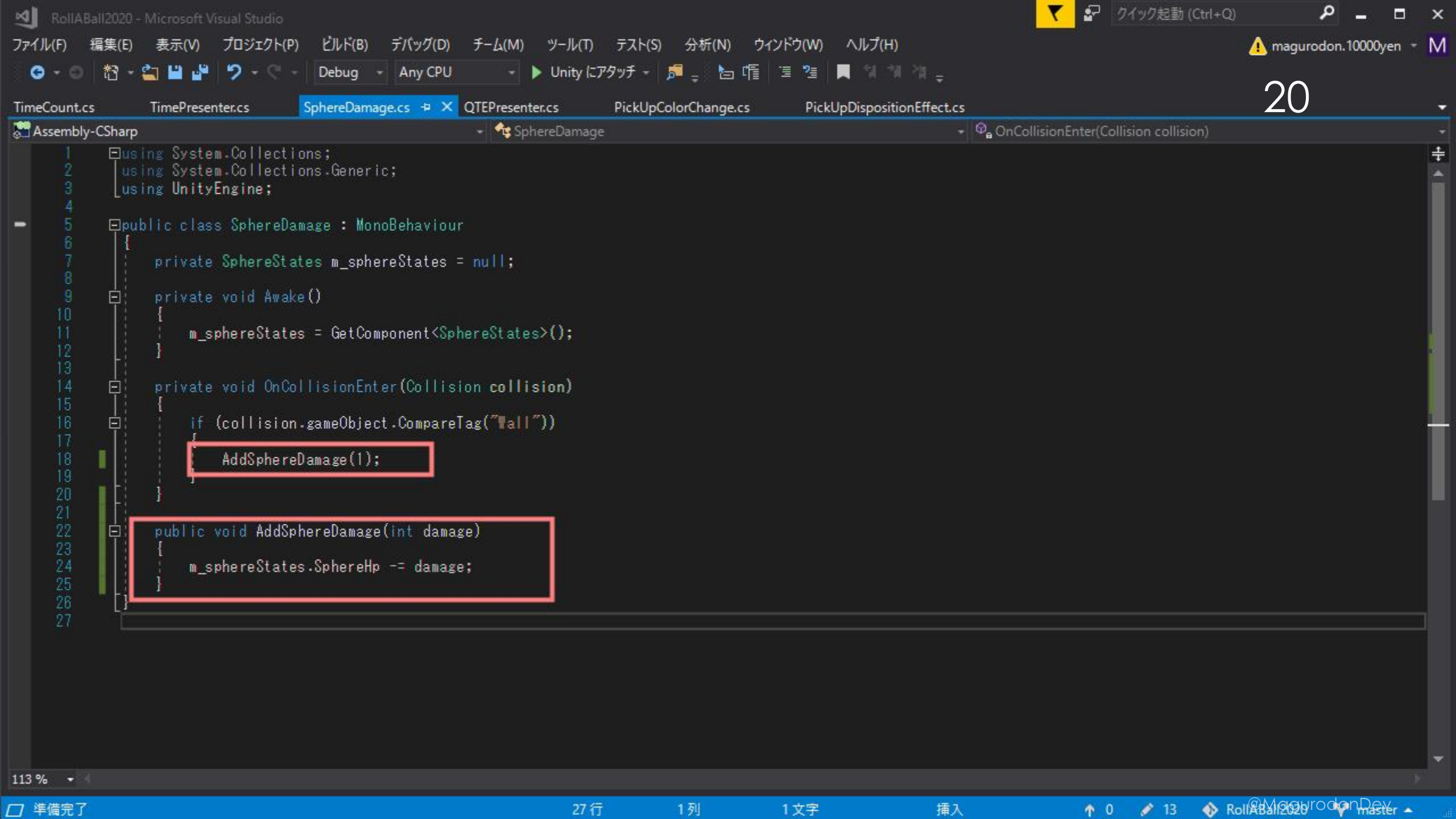
```
34 private void Update()
35 {
36     switch (QTEPresenterStates)
37     {
38         case QTEStates.OFF:
39             return;
40
41         case QTEStates.ON:
42             // 時間を止める
43             Time.timeScale = 0f;
44             m_timeLimit = 5f;
45
46             SetQTEObject(true);
47
48             // QTEの命令をユーザーに伝える
49             m_QTEText.text = $"{m_theme}と入力しろ!";
50             m_QTEInputField.text = string.Empty;
51
52             QTEPresenterStates = QTEStates.Input;
53             break;
54
55         case QTEStates.Input:
56             // 入力にはタイムリミットを設ける
57             m_timeLimit -= Time.unscaledDeltaTime;
58
59             m_QTETimeLimitText.text = $"{m_timeLimit}";
60
61             // タイムリミットが来たらResultに移行
62             if (m_timeLimit < 0)
63             {
64                 QTEPresenterStates = QTEStates.Result;
65             }
66             break;
67     }
68 }
```



```
RollABall2020 - Microsoft Visual Studio
クイック起動 (Ctrl+Q)
magurodon.10000yen

ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) チーム(M) ツール(T) テスト(S) 分析(N) ウィンドウ(W) ヘルプ(H)
Debug Any CPU Unity にアタッチ
TimeCount.cs TimePresenter.cs SphereDamage.cs QTEPresenter.cs PickUpColorChange.cs PickUpDispositionEffect.cs
Assembly-CSharp QTEPresenter.QTEStates Result
67 break;
68
69 case QTEStates.Result:
70
71     if (m_QTEInputField.text.Equals(m_theme))
72     {
73         // 正解していたら体力を5回復
74         SphereDamage.AddSphereDamage(-5);
75     }
76     else
77     {
78         //不正解ならダメージ
79         SphereDamage.AddSphereDamage(2);
80     }
81     SetQTEObject(false);
82     // 時間を戻す
83     Time.timeScale = 1f;
84     QTEPresenterStates = QTEStates.OFF;
85     break;
86 }
87
88 /// <summary>
89 /// m_QTEText等のGameObjectのアクティブ状態を管理する
90 /// </summary>
91 /// <param name="active"></param>
92 private void SetQTEObject(bool active)
93 {
94     m_QTEText.gameObject.SetActive(active);
95     m_QTETimeLimitText.gameObject.SetActive(active);
96     m_QTEInputField.gameObject.SetActive(active);
97 }
98
99
113 %
99行 1列 1文字 挿入 0 13 RollABall2020 master
```

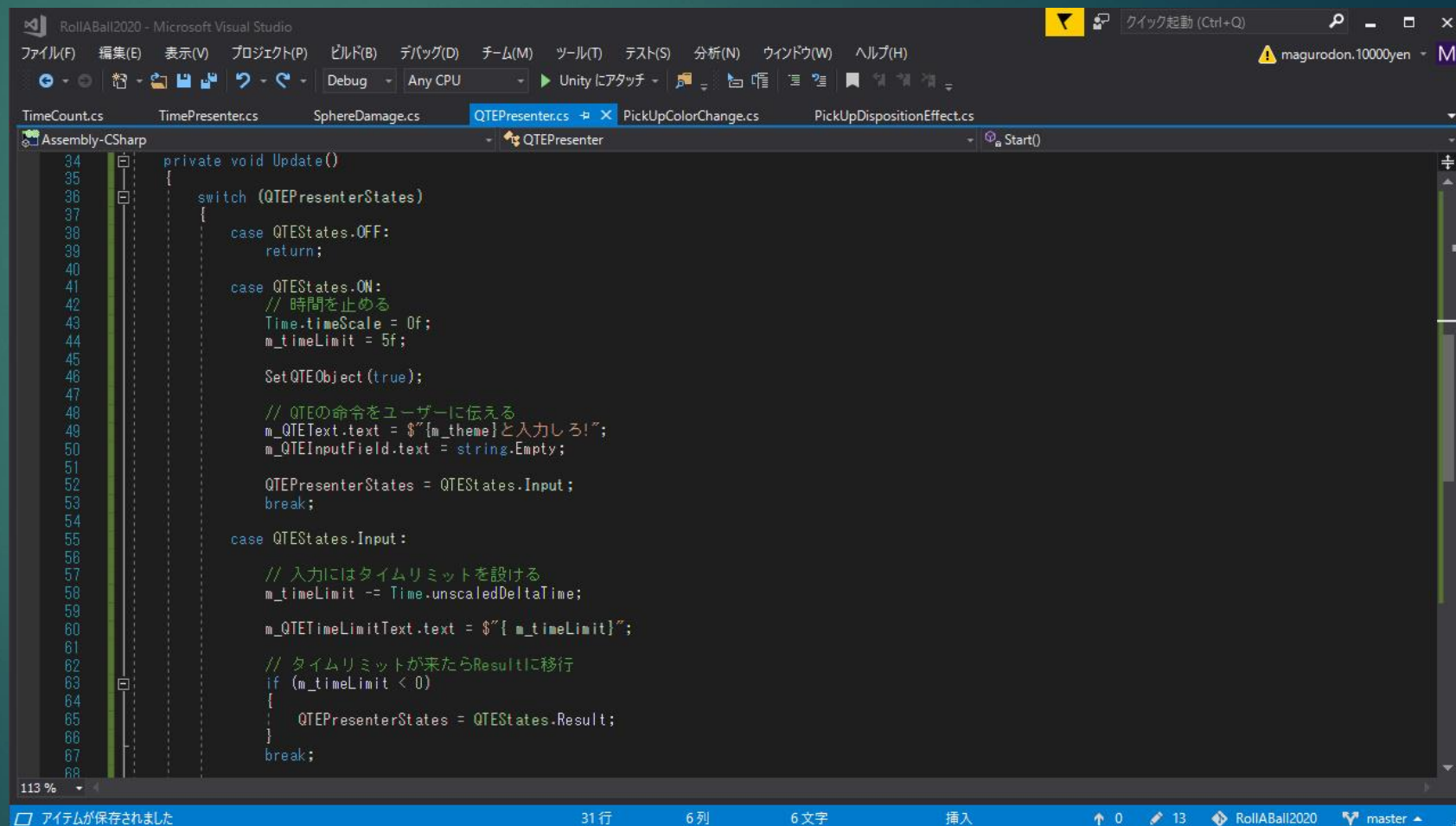






## RollABall

- ※TimeScaleとは
- スローモーションなどによく使われる関数です
- TimeScaleが0になった場合、FixedUpdate関数は呼び出されません
- つまり物理制御は全て止まります
- フレームレートに依存していない関数も全て止まります
- 今回はunscaledDeltaTimeというTimeScaleに干渉されないdeltaTimeを使用しています
- deltaTimeとはフレームとフレームの間の時間でしたね、思い出してください



```
private void Update()
{
    switch (QTEPresenterStates)
    {
        case QTEStates.OFF:
            return;

        case QTEStates.ON:
            // 時間を止める
            Time.timeScale = 0f;
            m_timeLimit = 5f;

            SetQTEObject(true);

            // QTEの命令をユーザーに伝える
            m_QTEText.text = $"[{m_theme}]と入力しろ!";
            m_QTEInputField.text = string.Empty;

            QTEPresenterStates = QTEStates.Input;
            break;

        case QTEStates.Input:
            // 入力にはタイムリミットを設ける
            m_timeLimit -= Time.unscaledDeltaTime;

            m_QTETimeLimitText.text = $"[{m_timeLimit}]";

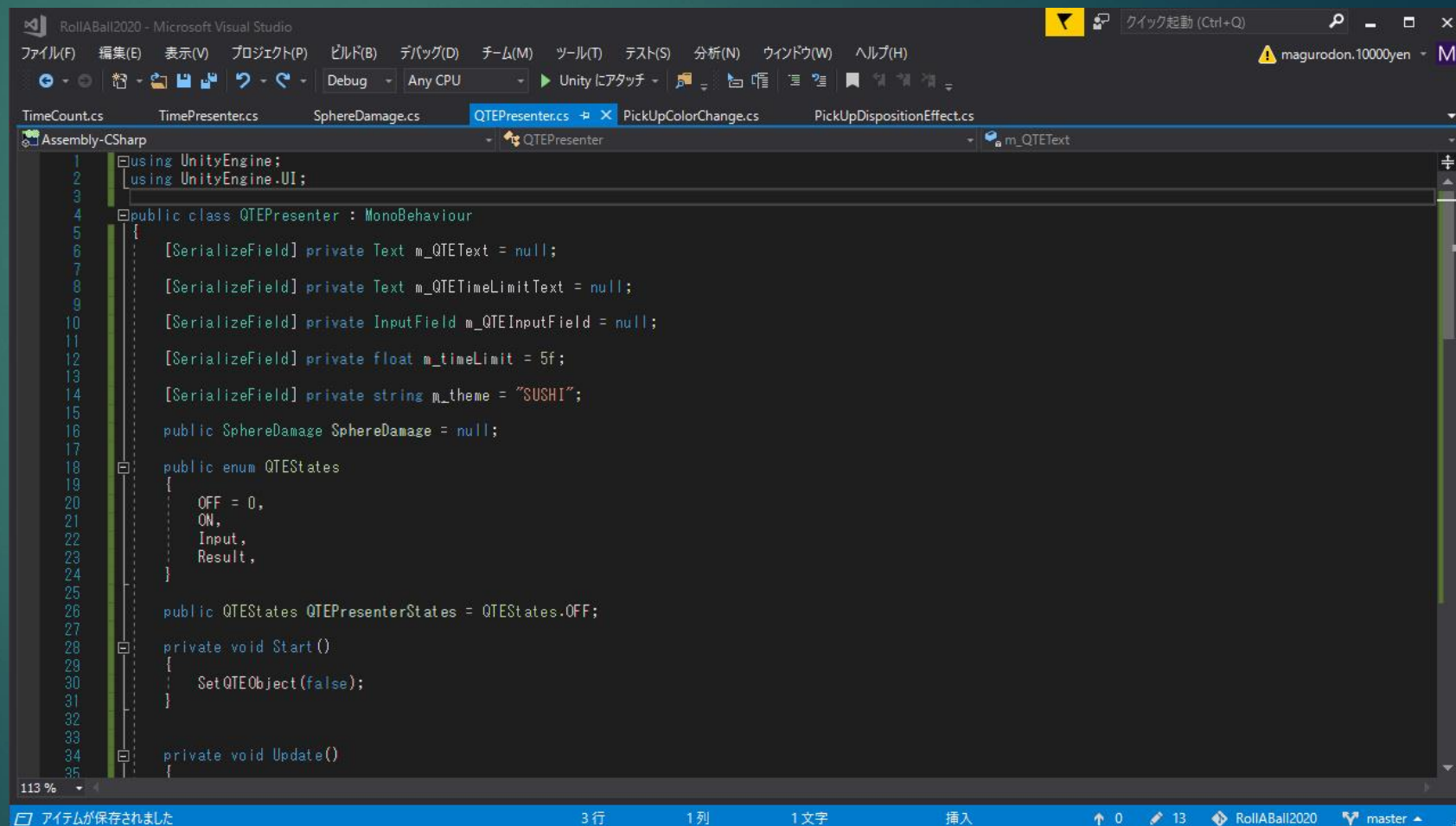
            // タイムリミットが来たらResultに移行
            if (m_timeLimit < 0)
            {
                QTEPresenterStates = QTEStates.Result;
            }
            break;
    }
}
```

# Unity

22

## RollABall

- ざっくり、今の時点で質問ある方おられますか
- 正味、今何を書いているかわからない方、正直に挙手してください
- そろそろ皆さんにコツを伝授していきたいと思います
- 日本語に変換していくので、コメントに書いて下さい
- プログラムも言語のひとつです
- 日本語で書かれているものをプログラムにしているだけなので、逆をしてあげればいいです
- C#は高級言語なのでそれが可能です



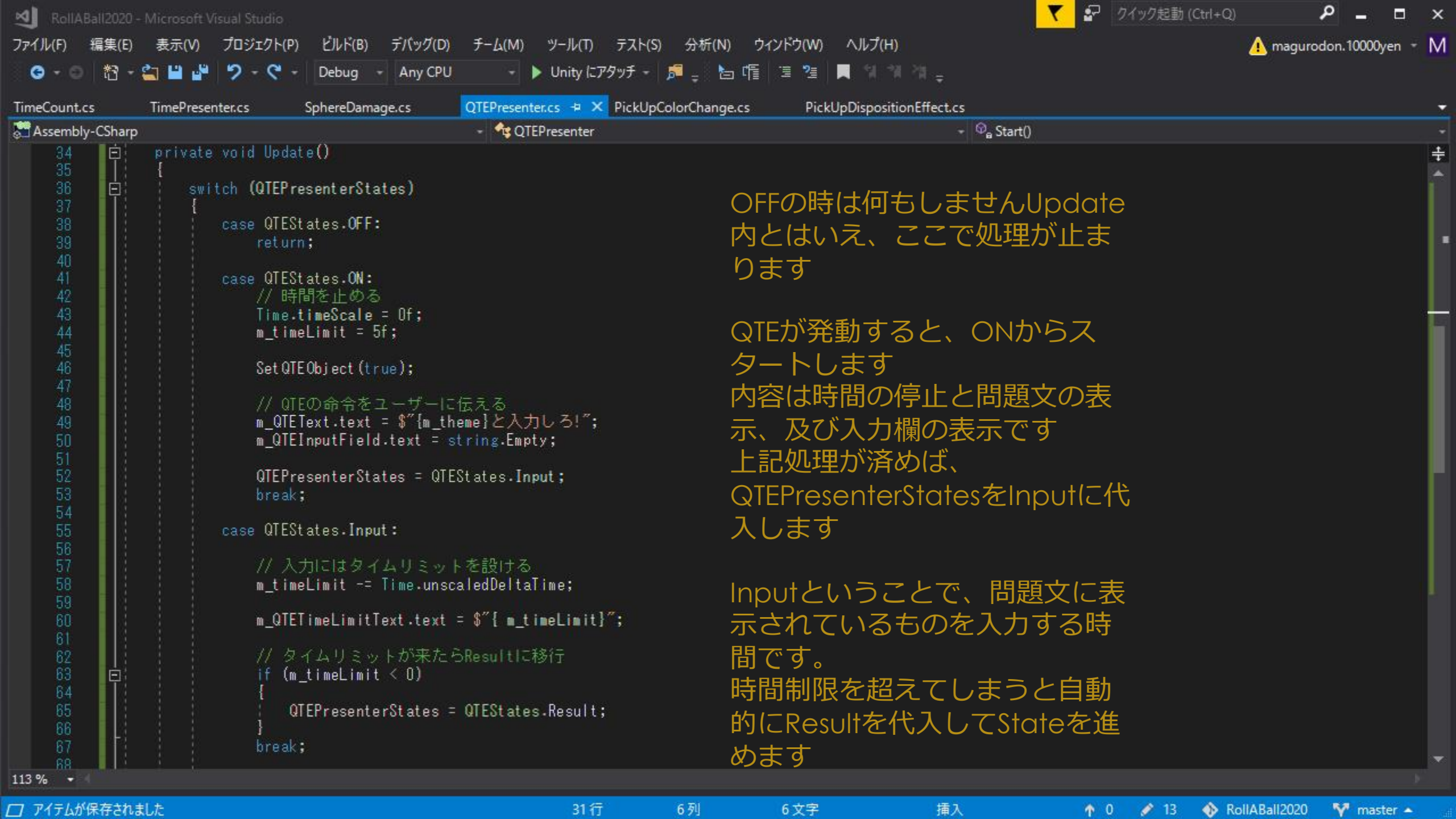
```
1 using UnityEngine;
2 using UnityEngine.UI;
3
4 public class QTEPresenter : MonoBehaviour
5 {
6     [SerializeField] private Text m_QTEText = null;
7
8     [SerializeField] private Text m_QTETimeLimitText = null;
9
10    [SerializeField] private InputField m_QTEInputField = null;
11
12    [SerializeField] private float m_timeLimit = 5f;
13
14    [SerializeField] private string m_theme = "SUSHI";
15
16    public SphereDamage SphereDamage = null;
17
18    public enum QTEStates
19    {
20        OFF = 0,
21        ON,
22        Input,
23        Result,
24    }
25
26    public QTEStates QTEPresenterStates = QTEStates.OFF;
27
28    private void Start()
29    {
30        SetQTEObject(false);
31    }
32
33
34    private void Update()
35    {
```

```
1 using UnityEngine;
2 using UnityEngine.UI;
3
4 public class QTEPresenter : MonoBehaviour
5 {
6     [SerializeField] private Text m_QTEText = null;
7
8     [SerializeField] private Text m_QTETimeLimitText = null;
9
10    [SerializeField] private InputField m_QTEInputField = null;
11
12    [SerializeField] private float m_timeLimit = 5f;
13
14    [SerializeField] private string m_theme = "SUSHI";
15
16    public SphereDamage SphereDamage = null;
17
18    public enum QTEStates
19    {
20        OFF = 0,
21        ON,
22        Input,
23        Result,
24    }
25
26    public QTEStates QTEPresenterStates = QTEStates.OFF;
27
28    private void Start()
29    {
30        SetQTEObject(false);
31    }
32
33
34    private void Update()
35    {
```

QTEの状態を管理するenumです  
このenumをupdateの中で判別  
させることで状態を遷移させて  
いきます

最初はOFFにしておきます





RollABall2020 - Microsoft Visual Studio

クイック起動 (Ctrl+Q)

magurodon.10000yen

ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) チーム(M) ツール(T) テスト(S) 分析(N) ウィンドウ(W) ヘルプ(H)

Debug Any CPU Unity にアタッチ

TimeCount.cs TimePresenter.cs SphereDamage.cs QTEPresenter.cs PickUpColorChange.cs PickUpDispositionEffect.cs

Assembly-CSharp QTEPresenter Start()

```
34 private void Update()
35 {
36     switch (QTEPresenterStates)
37     {
38         case QTEStates.OFF:
39             return;
40
41         case QTEStates.ON:
42             // 時間を止める
43             Time.timeScale = 0f;
44             m_timeLimit = 5f;
45
46             SetQTEObject(true);
47
48             // QTEの命令をユーザーに伝える
49             m_QTEText.text = $"{m_theme}と入力しろ!";
50             m_QTEInputField.text = string.Empty;
51
52             QTEPresenterStates = QTEStates.Input;
53             break;
54
55         case QTEStates.Input:
56
57             // 入力にはタイムリミットを設ける
58             m_timeLimit -= Time.unscaledDeltaTime;
59
60             m_QTETimeLimitText.text = $"{m_timeLimit}";
61
62             // タイムリミットが来たらResultに移行
63             if (m_timeLimit < 0)
64             {
65                 QTEPresenterStates = QTEStates.Result;
66             }
67             break;
68     }
```

OFFの時は何もしませんUpdate内とはいえ、ここで処理が止まります

QTEが発動すると、ONからスタートします  
内容は時間の停止と問題文の表示、及び入力欄の表示です  
上記処理が済めば、QTEPresenterStatesをInputに代入します

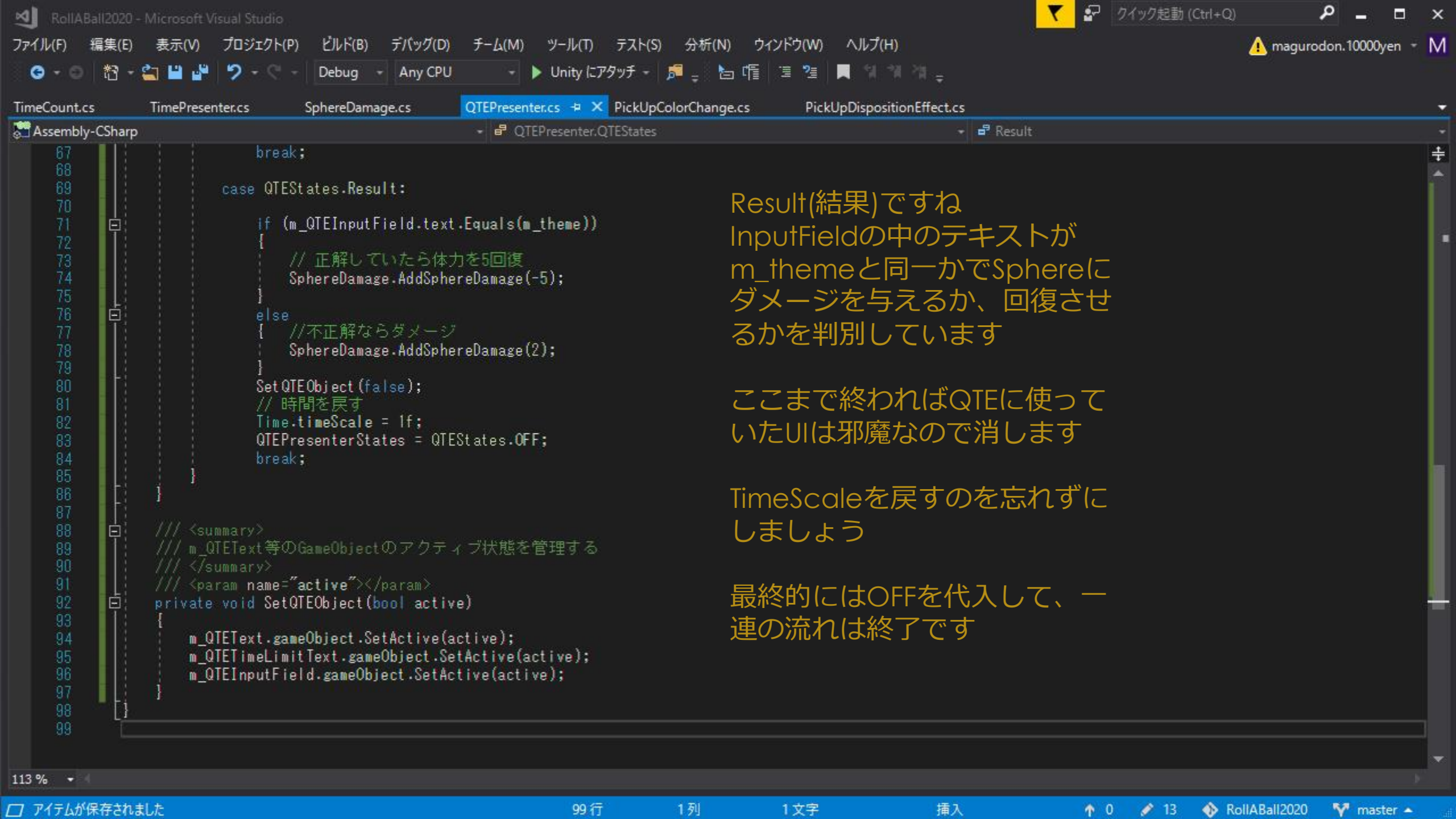
Inputということで、問題文に表示されているものを入力する時間です。  
時間制限を超えてしまうと自動的にResultを代入してStateを進めます

113 %

アイテムが保存されました

31行 6列 6文字 挿入 0 13 RollABall2020 master



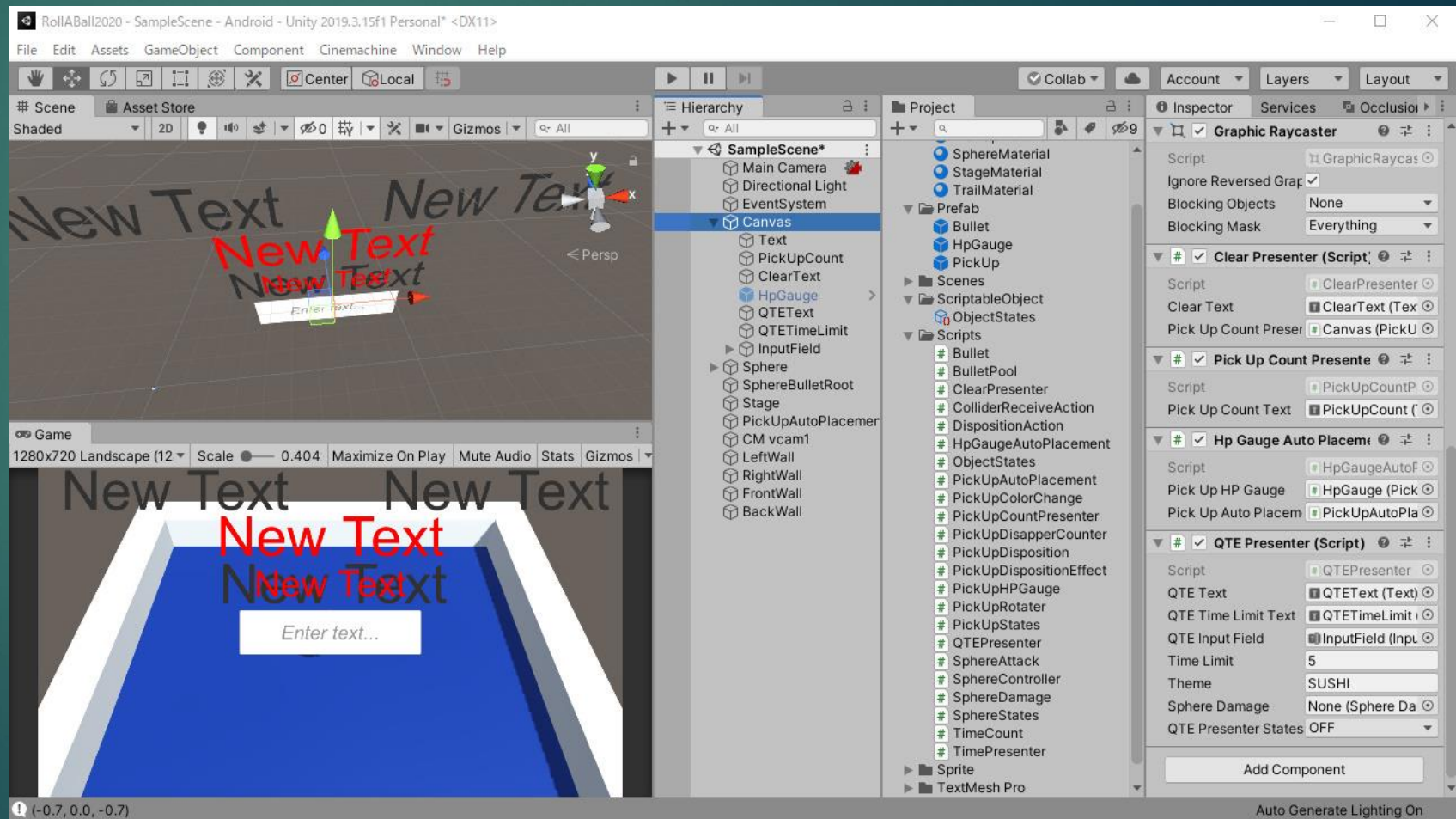


# Unity

26

## RollABall

- Unityに戻って設定をしていきましょう
- Canvas直下にTextを2つ、InputFieldを1つ作成してみてください
- UIを作成する場合、すべて右クリック→Create→UIの一覧の中にあります
- 作成したtextを右図のような位置に調整して、色も変えておいてください
- Google検索していただいてもかまいません
- 自力でやってみましょう



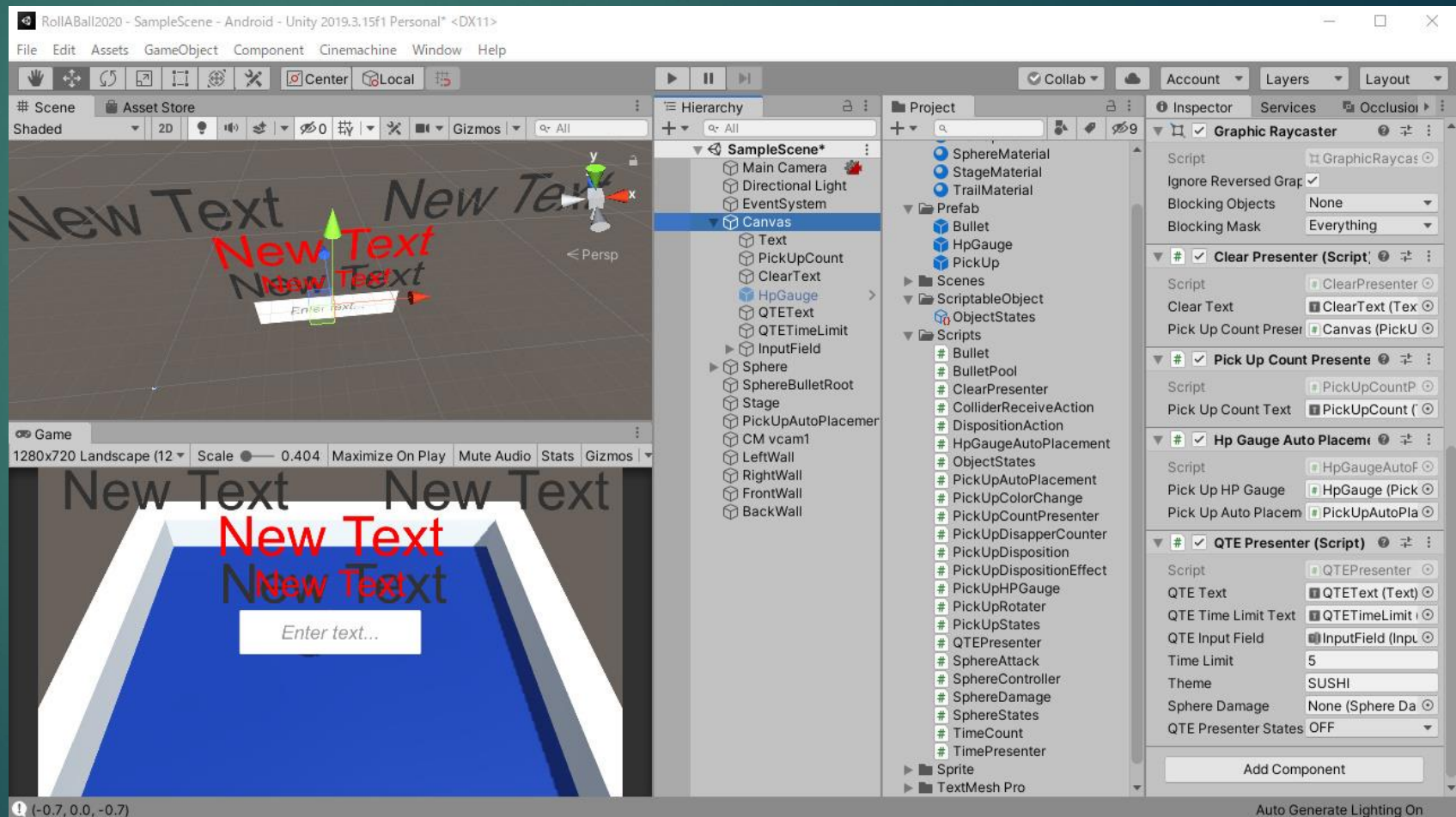


# Unity

## RollABall

27

- 作成が終わりましたら、CanvasにQTETPresenterをD&Dして設定していきます
- SphereDamageは空のままで大丈夫です
- ここまで出来ましたら、最初で作成したPickUpDispositionの中身もQTEを追加していきます





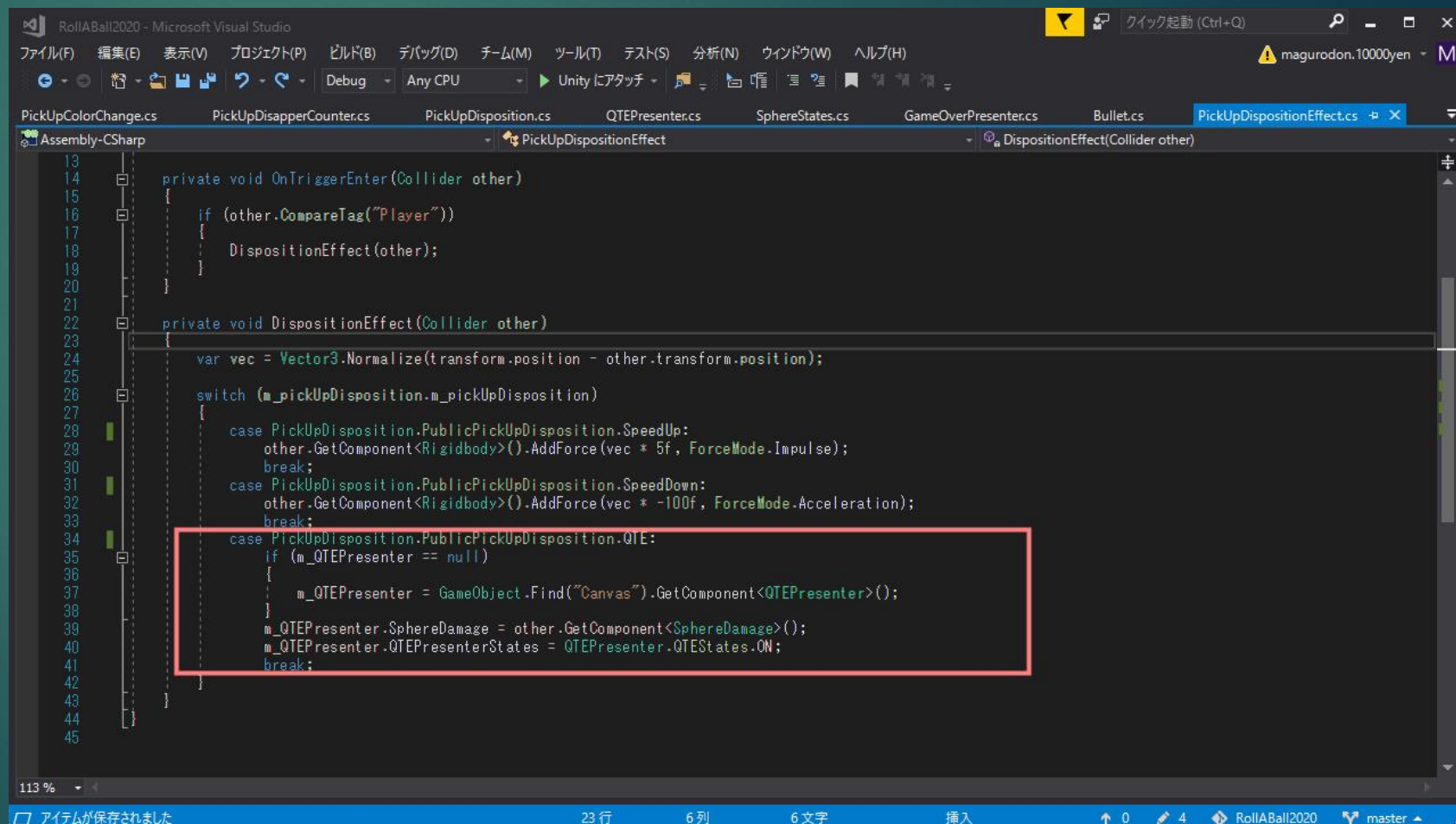
```
1 using UnityEngine;
2
3 public class PickupDispositionEffect : MonoBehaviour
4 {
5     private PickupDisposition m_pickUpDisposition = null;
6     private QTEPresenter m_QTEPresenter = null;
7
8     private void Start()
9     {
10         m_pickUpDisposition = GetComponent<PickupDisposition>();
11     }
12
13     private void OnTriggerEnter(Collider other)
14     {
15         if (other.CompareTag("Player"))
16         {
17             DispositionEffect(other);
18         }
19     }
20
21     private void DispositionEffect(Collider other)
22     {
23         var vec = Vector3.Normalize(transform.position - other.transform.position);
24
25         switch (m_pickUpDisposition.m_pickUpDisposition)
26         {
27             case PickupDisposition.PublicPickUpDisposition.SpeedUp:
28                 other.GetComponent<Rigidbody>().AddForce(vec * 5f, ForceMode.Impulse);
29                 break;
30             case PickupDisposition.PublicPickUpDisposition.SpeedDown:
31                 other.GetComponent<Rigidbody>().AddForce(vec * -100f, ForceMode.Acceleration);
32                 break;
33             case PickupDisposition.PublicPickUpDisposition.QTE:
34                 if (m_QTEPresenter == null)
```

```
13
14 private void OnTriggerEnter(Collider other)
15 {
16     if (other.CompareTag("Player"))
17     {
18         DispositionEffect(other);
19     }
20 }
21
22 private void DispositionEffect(Collider other)
23 {
24     var vec = Vector3.Normalize(transform.position - other.transform.position);
25
26     switch (m_pickUpDisposition.m_pickUpDisposition)
27     {
28     case PickUpDisposition.PublicPickUpDisposition.SpeedUp:
29         other.GetComponent<Rigidbody>().AddForce(vec * 5f, ForceMode.Impulse);
30         break;
31     case PickUpDisposition.PublicPickUpDisposition.SpeedDown:
32         other.GetComponent<Rigidbody>().AddForce(vec * -100f, ForceMode.Acceleration);
33         break;
34     case PickUpDisposition.PublicPickUpDisposition.QTE:
35         if (m_QTEPresenter == null)
36         {
37             m_QTEPresenter = GameObject.Find("Canvas").GetComponent<QTEPresenter>();
38         }
39         m_QTEPresenter.SphereDamage = other.GetComponent<SphereDamage>();
40         m_QTEPresenter.QTEPresenterStates = QTEPresenter.QTEStates.ON;
41         break;
42     }
43 }
44
45
```



## RollABall

- ※GameObject.Find("hoge")
- 英語のままですね
- 引数に入れられた名前のGameObjectを見つけてくれます
- ただし、この処理自体はScene内のGameObjectすべてを走査するので、CPU的に重たいです
- なのでなるべくクラス内の変数にキャッシュすることをお勧めします
- ※キャッシュとは
- 毎回取得しに行くのではなく、変数に格納しておき、一回目以降はその変数を使うことです



```
13 private void OnTriggerEnter(Collider other)
14 {
15     if (other.CompareTag("Player"))
16     {
17         DispositionEffect(other);
18     }
19 }
20
21 private void DispositionEffect(Collider other)
22 {
23     var vec = Vector3.Normalize(transform.position - other.transform.position);
24
25     switch (_pickUpDisposition._pickUpDisposition)
26     {
27         case PickUpDisposition.PublicPickUpDisposition.SpeedUp:
28             other.GetComponent<Rigidbody>().AddForce(vec * 5f, ForceMode.Impulse);
29             break;
30         case PickUpDisposition.PublicPickUpDisposition.SpeedDown:
31             other.GetComponent<Rigidbody>().AddForce(vec * -100f, ForceMode.Acceleration);
32             break;
33         case PickUpDisposition.PublicPickUpDisposition.QTE:
34             if (_QTEPresenter == null)
35             {
36                 _QTEPresenter = GameObject.Find("Canvas").GetComponent<QTEPresenter>();
37             }
38             _QTEPresenter.SphereDamage = other.GetComponent<SphereDamage>();
39             _QTEPresenter.QTEPresenterStates = QTEPresenter.QTEStates.ON;
40             break;
41     }
42 }
43
44
45
```

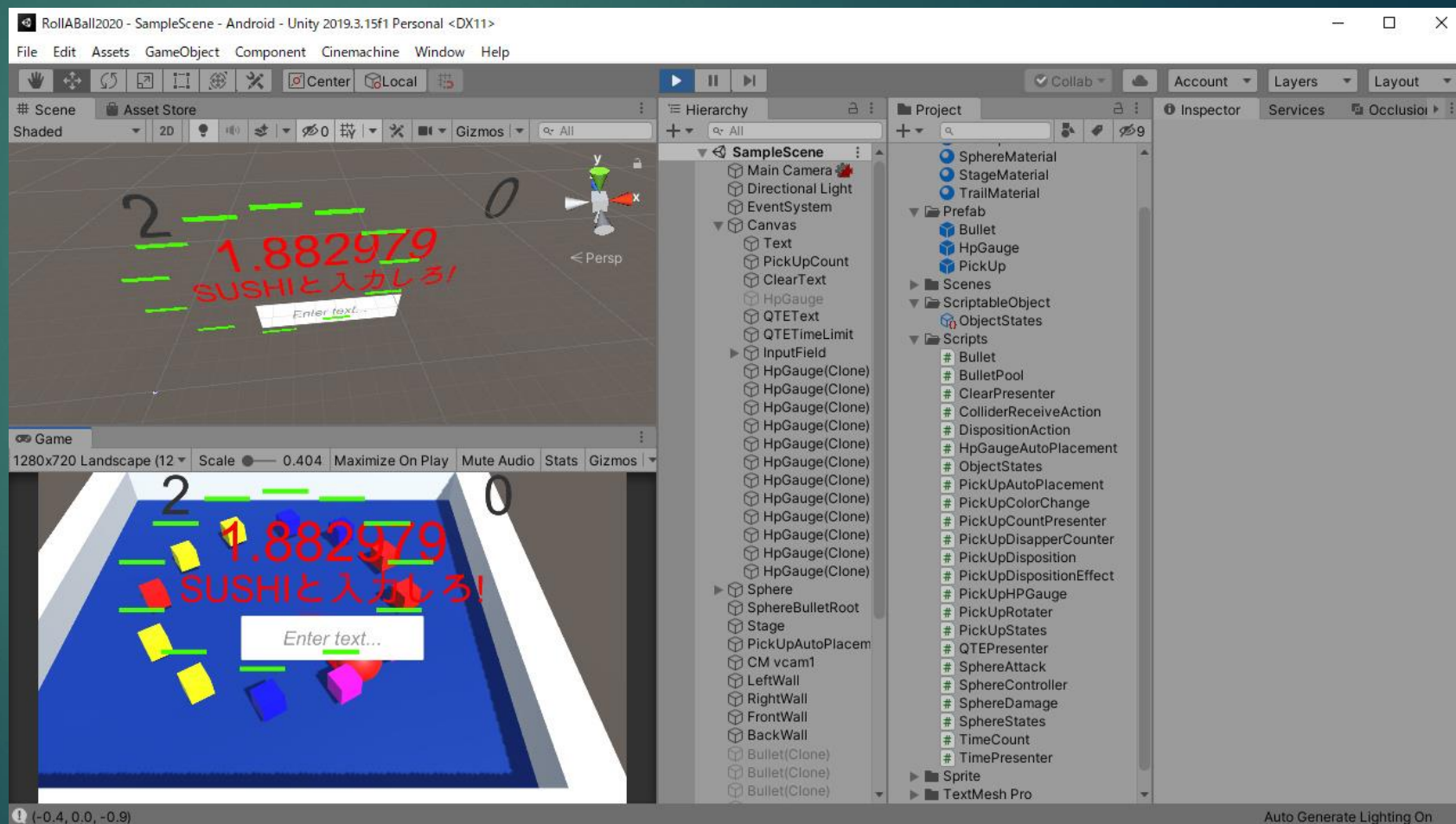


# Unity

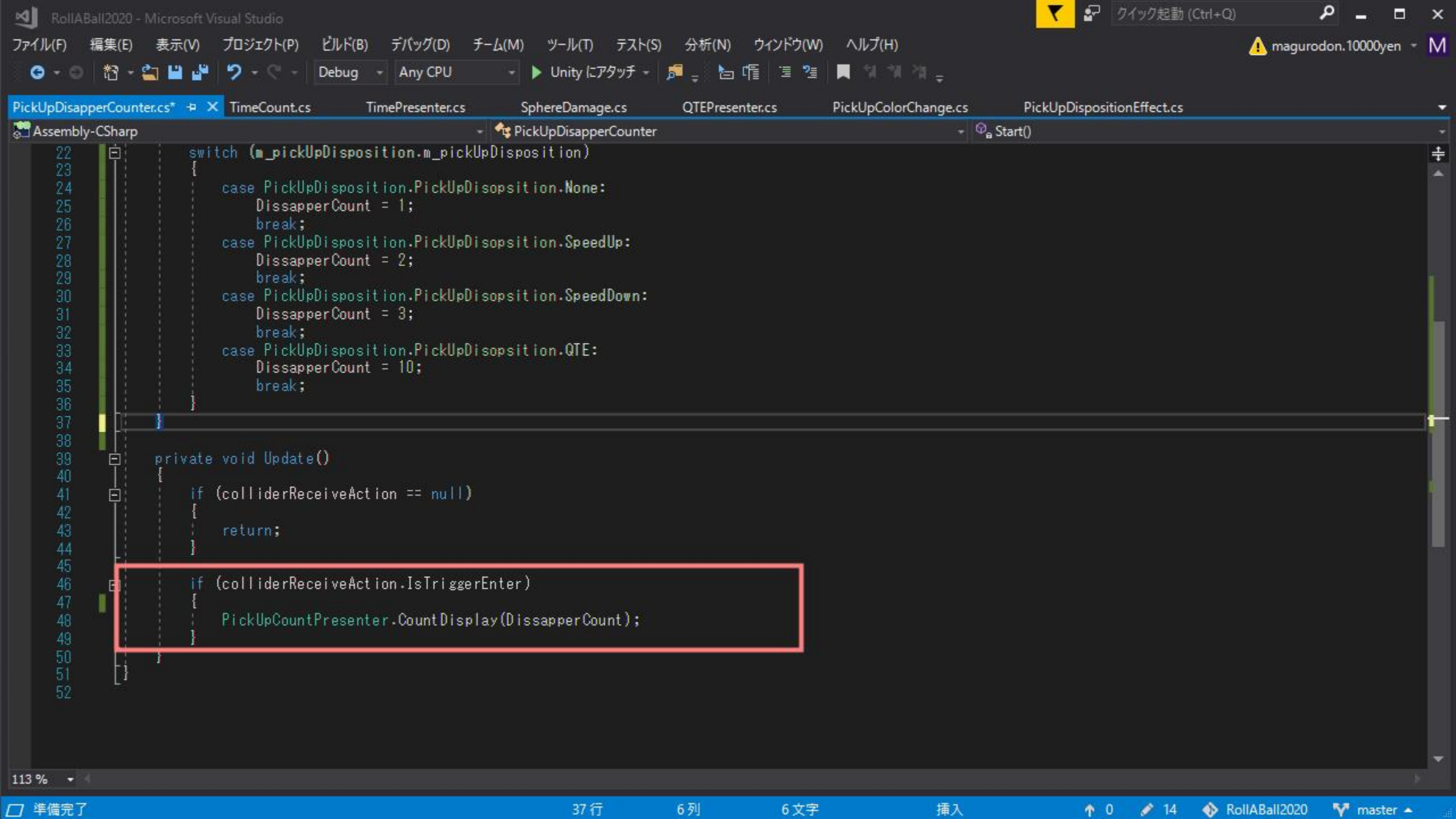
31

## RollABall

- これでQTEを発動させる準備が整いました
- Unityに戻ってプレイしてみましよう
- SUSHIと入力すれば体力が回復し、間違えば体力が減ります
- さて、色によって得点を変更していきましょう
- PickupDisapperCounterを改修していきます



```
10 public int DissapperCount { get; private set; } = 0;
11
12 /// <summary>
13 /// UnityEditorで設定するColliderReceiveAction
14 /// </summary>
15 [SerializeField] private ColliderReceiveAction colliderReceiveAction = null;
16
17 private PickUpDisposition m_pickUpDisposition = null;
18
19 private void Start()
20 {
21     m_pickUpDisposition = GetComponent<PickUpDisposition>();
22     switch (m_pickUpDisposition.m_pickUpDisposition)
23     {
24         case PickUpDisposition.PickUpDisposition.None:
25             DissapperCount = 1;
26             break;
27         case PickUpDisposition.PickUpDisposition.SpeedUp:
28             DissapperCount = 2;
29             break;
30         case PickUpDisposition.PickUpDisposition.SpeedDown:
31             DissapperCount = 3;
32             break;
33         case PickUpDisposition.PickUpDisposition.QTE:
34             DissapperCount = 10;
35             break;
36     }
37
38
39
40 private void Update()
41 {
42     if (colliderReceiveAction == null)
43     {
44         return;
```

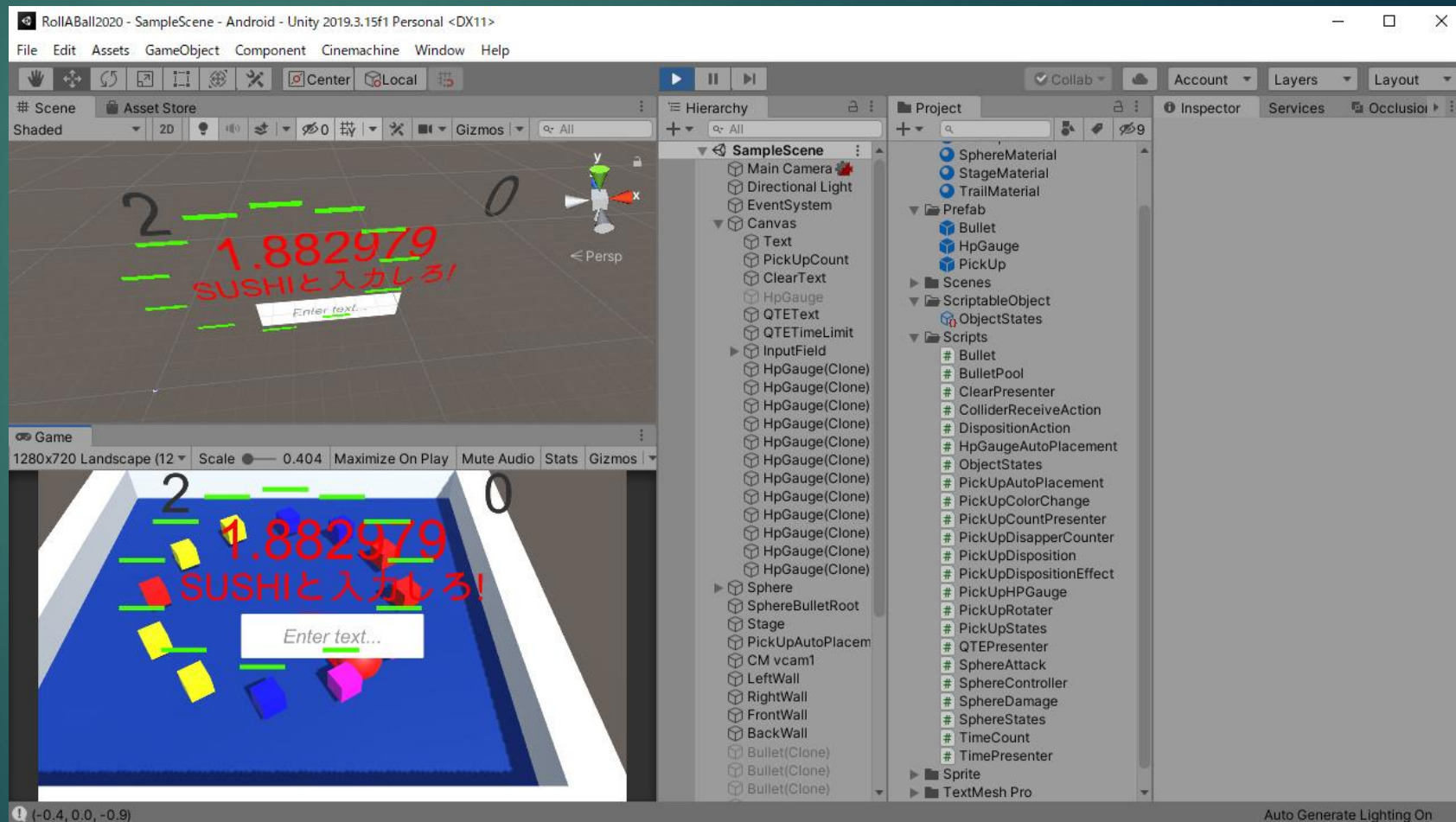


```
22 switch (m_pickUpDisposition.m_pickUpDisposition)
23 {
24     case PickupDisposition.PickUpDisopsition.None:
25         DissapperCount = 1;
26         break;
27     case PickupDisposition.PickUpDisopsition.SpeedUp:
28         DissapperCount = 2;
29         break;
30     case PickupDisposition.PickUpDisopsition.SpeedDown:
31         DissapperCount = 3;
32         break;
33     case PickupDisposition.PickUpDisopsition.QTE:
34         DissapperCount = 10;
35         break;
36 }
37
38
39 private void Update()
40 {
41     if (colliderReceiveAction == null)
42     {
43         return;
44     }
45
46     if (colliderReceiveAction.IsTriggerEnter)
47     {
48         PickUpCountPresenter.CountDisplay(DissapperCount);
49     }
50 }
51
52
```



## RollABall

- プレイしてみると、色によって得点が変わっていると思います
- さて、これでPickUp側の仕様は全て満たしたことになります
- 後半は妨害要素をいれていきましょう
- まずはプレイヤーを襲う弾を作っていきます
- DisturbanceBulletというスクリプトを作成してください



35

ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) チーム(M) ツール(T) テスト(S) 分析(N) ウィンドウ(W) ヘルプ(H)

Debug Any CPU Unity にアタッチ

SphereAttack.cs DisturbanceBullet.cs PickUpDispositionEffect.cs

Assembly-CSharp

DisturbanceBullet

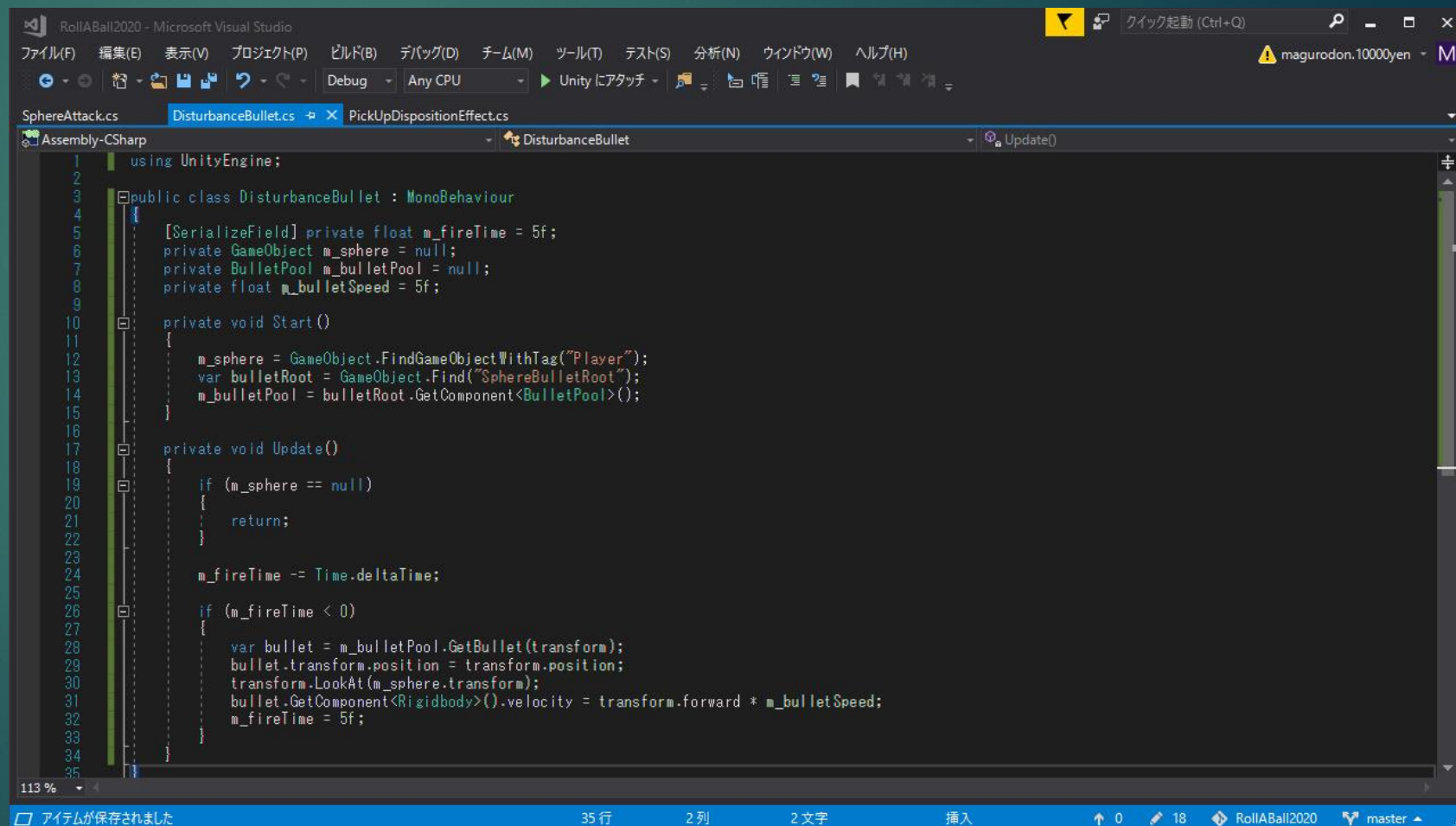
Update()

```
1 using UnityEngine;
2
3 public class DisturbanceBullet : MonoBehaviour
4 {
5     [SerializeField] private float m_fireTime = 5f;
6     private GameObject m_sphere = null;
7     private BulletPool m_bulletPool = null;
8     private float m_bulletSpeed = 5f;
9
10    private void Start()
11    {
12        m_sphere = GameObject.FindGameObjectWithTag("Player");
13        var bulletRoot = GameObject.Find("SphereBulletRoot");
14        m_bulletPool = bulletRoot.GetComponent<BulletPool>();
15    }
16
17    private void Update()
18    {
19        if (m_sphere == null)
20        {
21            return;
22        }
23
24        m_fireTime -= Time.deltaTime;
25
26        if (m_fireTime < 0)
27        {
28            var bullet = m_bulletPool.GetBullet(transform);
29            bullet.transform.position = transform.position;
30            transform.LookAt(m_sphere.transform);
31            bullet.GetComponent<Rigidbody>().velocity = transform.forward * m_bulletSpeed;
32            m_fireTime = 5f;
33        }
34    }
35 }
```

113 %

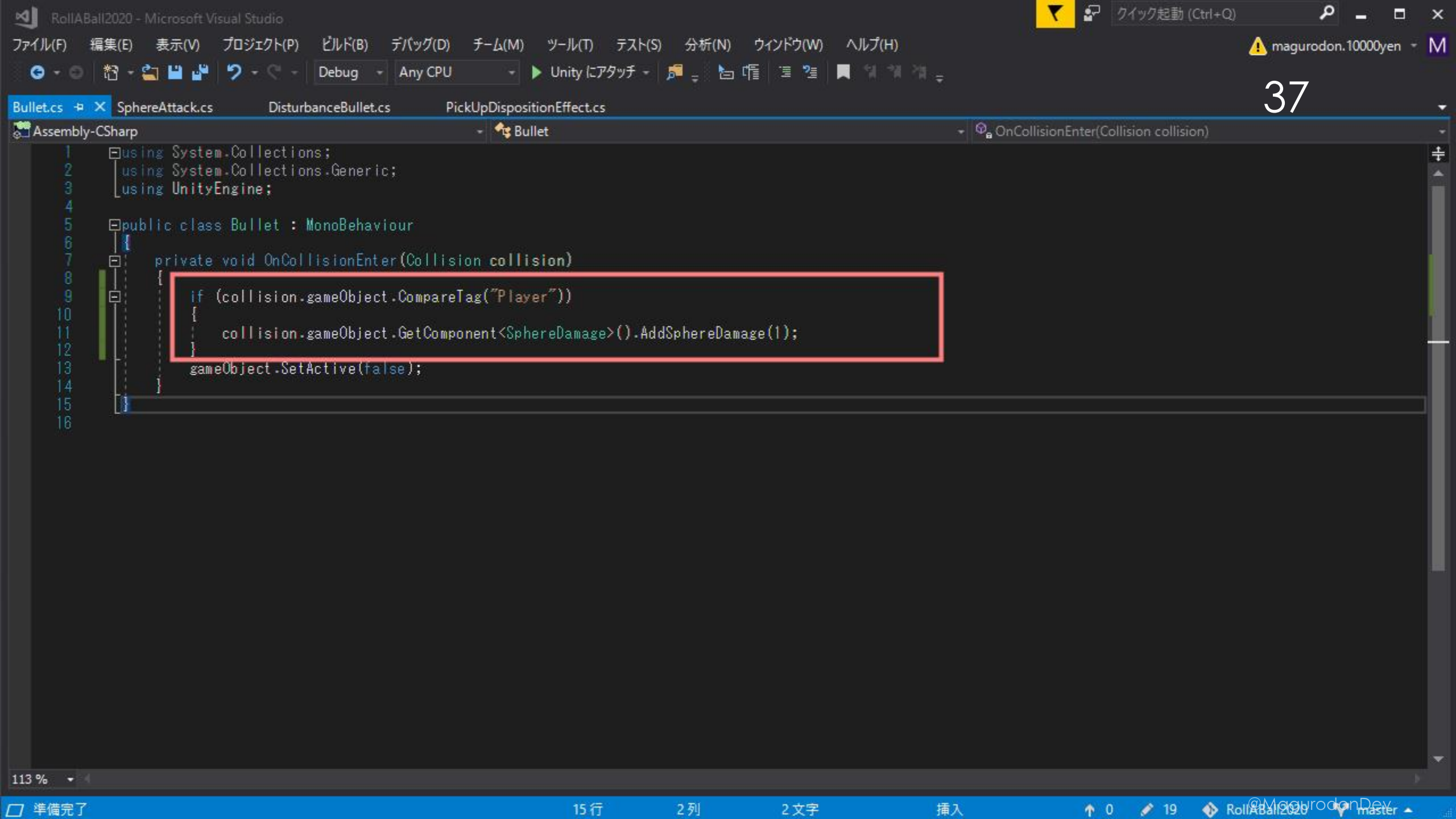
## RollABall

- ※GameObject.FindGameObjectWithTag("Hogehoge")
- GameObjectをファインド(見つける)方法は主に名前で検索するか、GameObjectに設定されているTagを検索する方法があります
- 処理が軽く、CPUに優しいのはFindGameObjectWithTagです
- 次はBulletがSphereにダメージを与えられるように少し編集しましょう



```
1 using UnityEngine;
2
3 public class DisturbanceBullet : MonoBehaviour
4 {
5     [SerializeField] private float m_fireTime = 5f;
6     private GameObject m_sphere = null;
7     private BulletPool m_bulletPool = null;
8     private float m_bulletSpeed = 5f;
9
10    private void Start()
11    {
12        m_sphere = GameObject.FindGameObjectWithTag("Player");
13        var bulletRoot = GameObject.Find("SphereBulletRoot");
14        m_bulletPool = bulletRoot.GetComponent<BulletPool>();
15    }
16
17    private void Update()
18    {
19        if (m_sphere == null)
20        {
21            return;
22        }
23
24        m_fireTime -= Time.deltaTime;
25
26        if (m_fireTime < 0)
27        {
28            var bullet = m_bulletPool.GetBullet(transform);
29            bullet.transform.position = transform.position;
30            transform.LookAt(m_sphere.transform);
31            bullet.GetComponent<Rigidbody>().velocity = transform.forward * m_bulletSpeed;
32            m_fireTime = 5f;
33        }
34    }
35 }
```





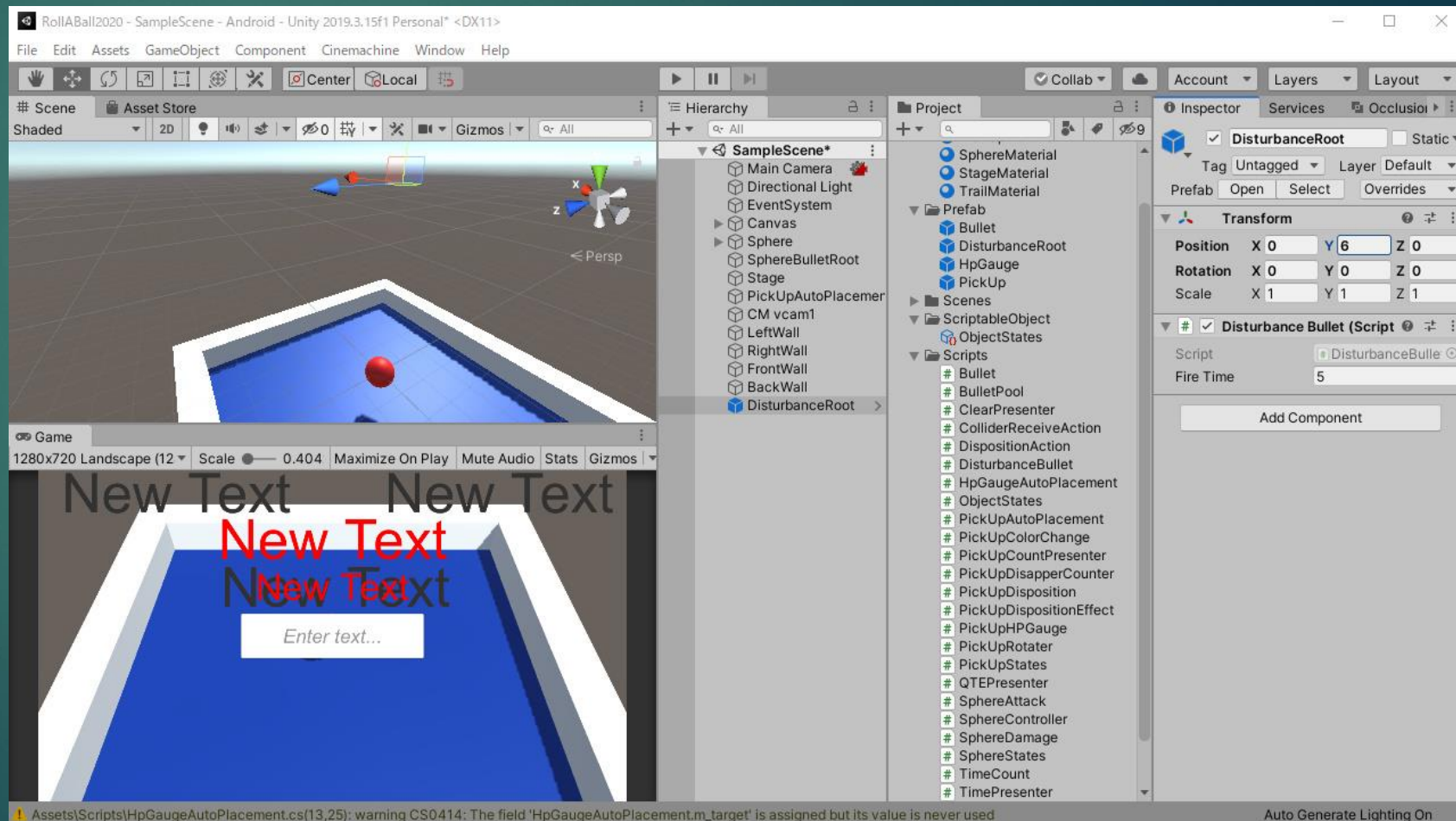
37

# Unity

## RollABall

38

- Unityに戻ってCreateEmptyから[DisturbanceRoot]を作成してください
- それをPrefabにさせていただき、今回作成したDisturbanceBulletをD&Dします
- Hierarchy欄に残っているDisturbanceRootのY軸に6を設定してください
- ここまでできたらプレイしてみましょう
- 上からSphereに向けて弾が降ってくるはずですが

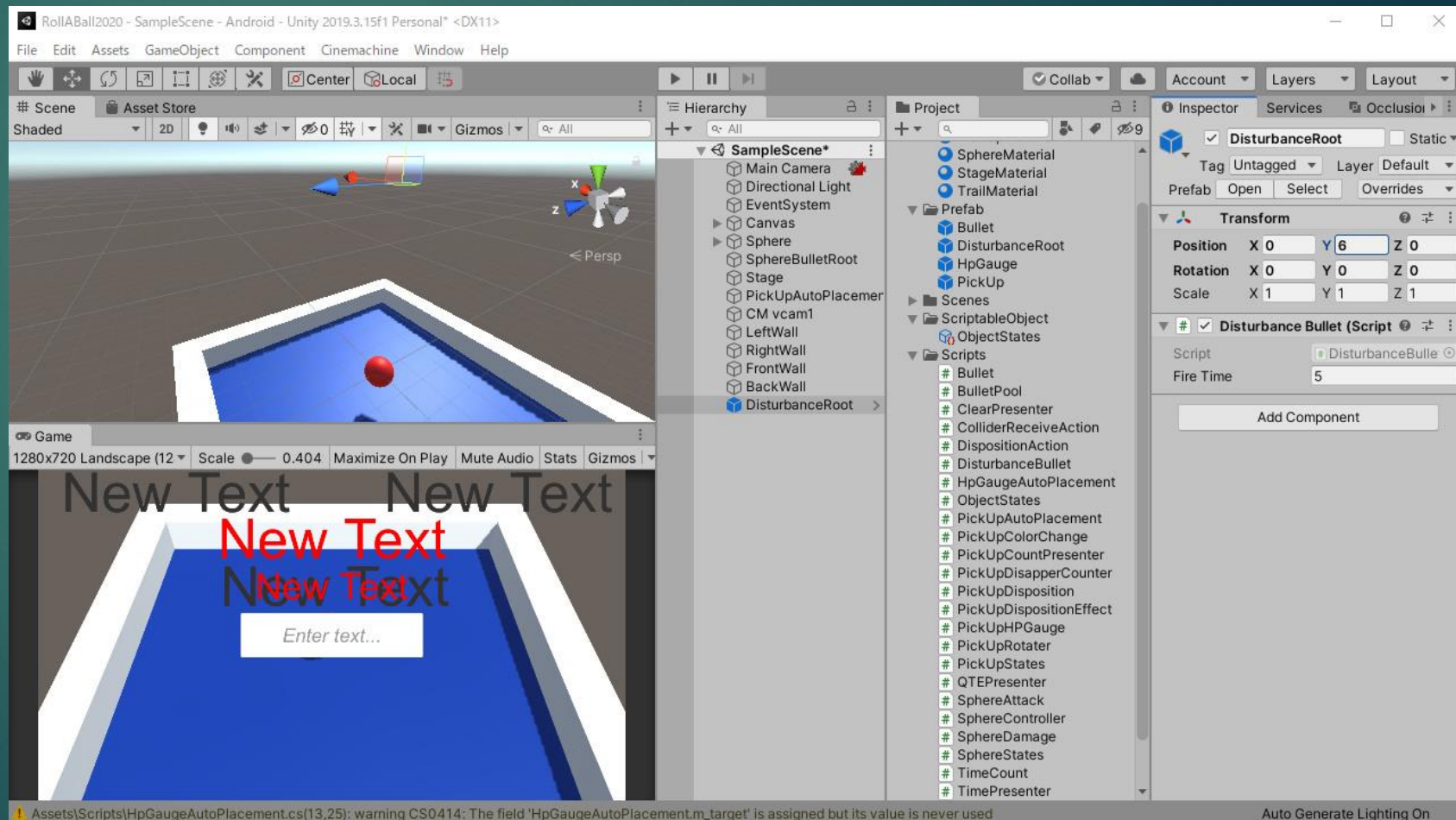


# Unity

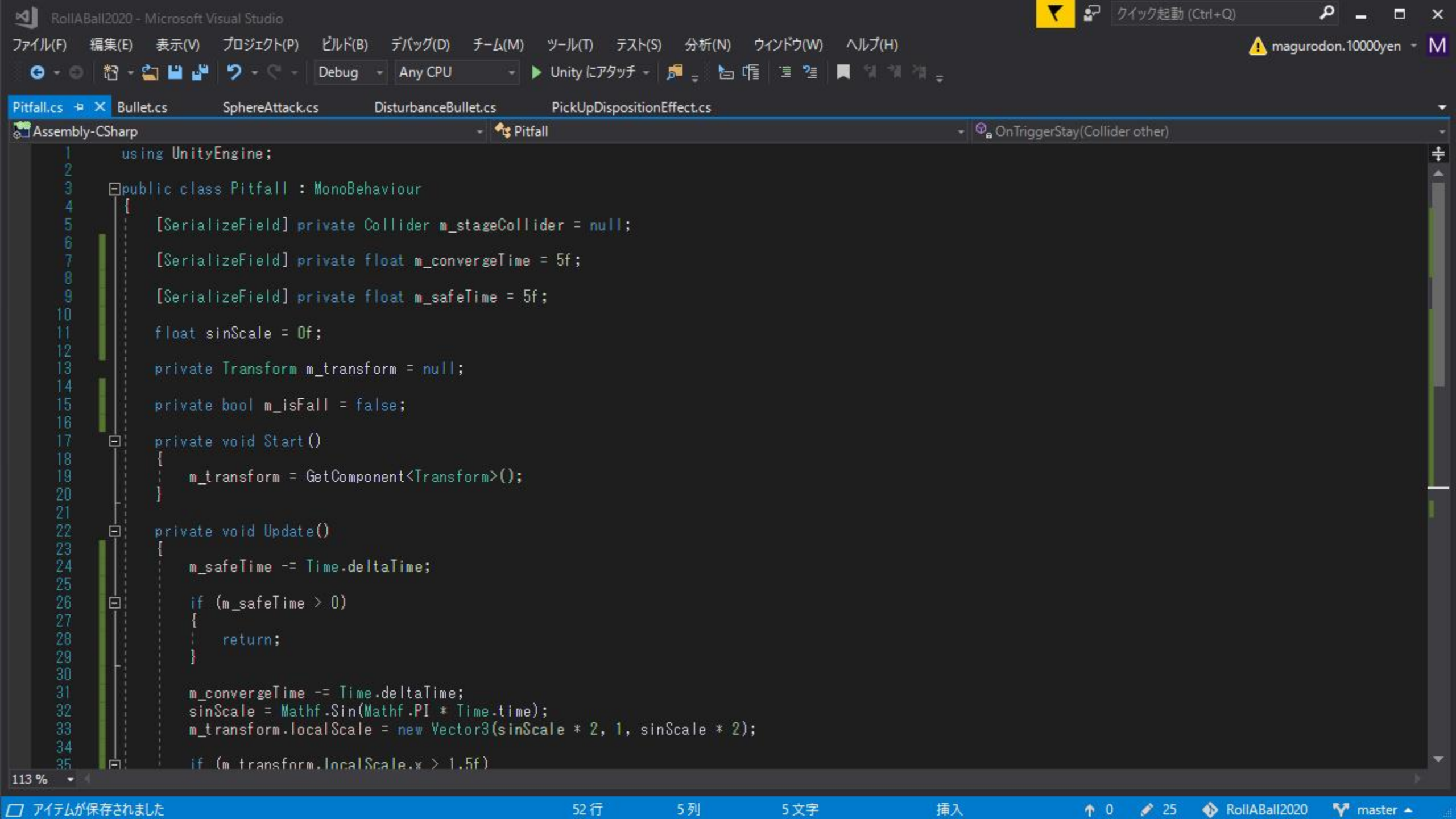
## RollABall

39

- 次に落とし穴を作っていきます
- Pitfallというスクリプトを作成してください
- SphereにあたるとStageのColliderを消して、Sphereが落下するようにします







```
19     m_transform = GetComponent<Transform>();
20 }
21
22 private void Update()
23 {
24     m_safeTime -= Time.deltaTime;
25
26     if (m_safeTime > 0)
27     {
28         return;
29     }
30
31     m_convergeTime -= Time.deltaTime;
32     sinScale = Mathf.Sin(Mathf.PI * Time.time);
33     m_transform.localScale = new Vector3(sinScale * 2, 1, sinScale * 2);
34
35     if (m_transform.localScale.x > 1.5f)
36     {
37         m_isFall = true;
38     }
39     else
40     {
41         m_isFall = false;
42     }
43
44     if (m_convergeTime > 0)
45     {
46         return;
47     }
48     m_transform.localScale = new Vector3(0, 0, 0);
49     m_convergeTime = 5f;
50     m_safeTime = 5f;
51 }
52
53 private void OnTriggerStay(Collider other)
```

```
43  
44     if (m_convergeTime > 0)  
45     {  
46         return;  
47     }  
48     m_transform.localScale = new Vector3(0, 0, 0);  
49     m_convergeTime = 5f;  
50     m_safeTime = 5f;  
51 }  
52  
53 private void OnTriggerStay(Collider other)  
54 {  
55     if (other.gameObject.CompareTag("Player") && m_isFall)  
56     {  
57         m_stageCollider.enabled = false;  
58     }  
59 }  
60  
61
```



## RollABall

- ※Mathf.Sin(float)
- はい、またやってまいりましたね三角関数のSin
- Sinにpiを用いることにより、-1～1の間で新円を真上からみた円運動を取得することができます
- 今回は滑らかに円が閉じたり開いたりするのに使いました
- 他にもいろいろと活用方法がありますので、数学の学びなおしはやっておいて損ではないです

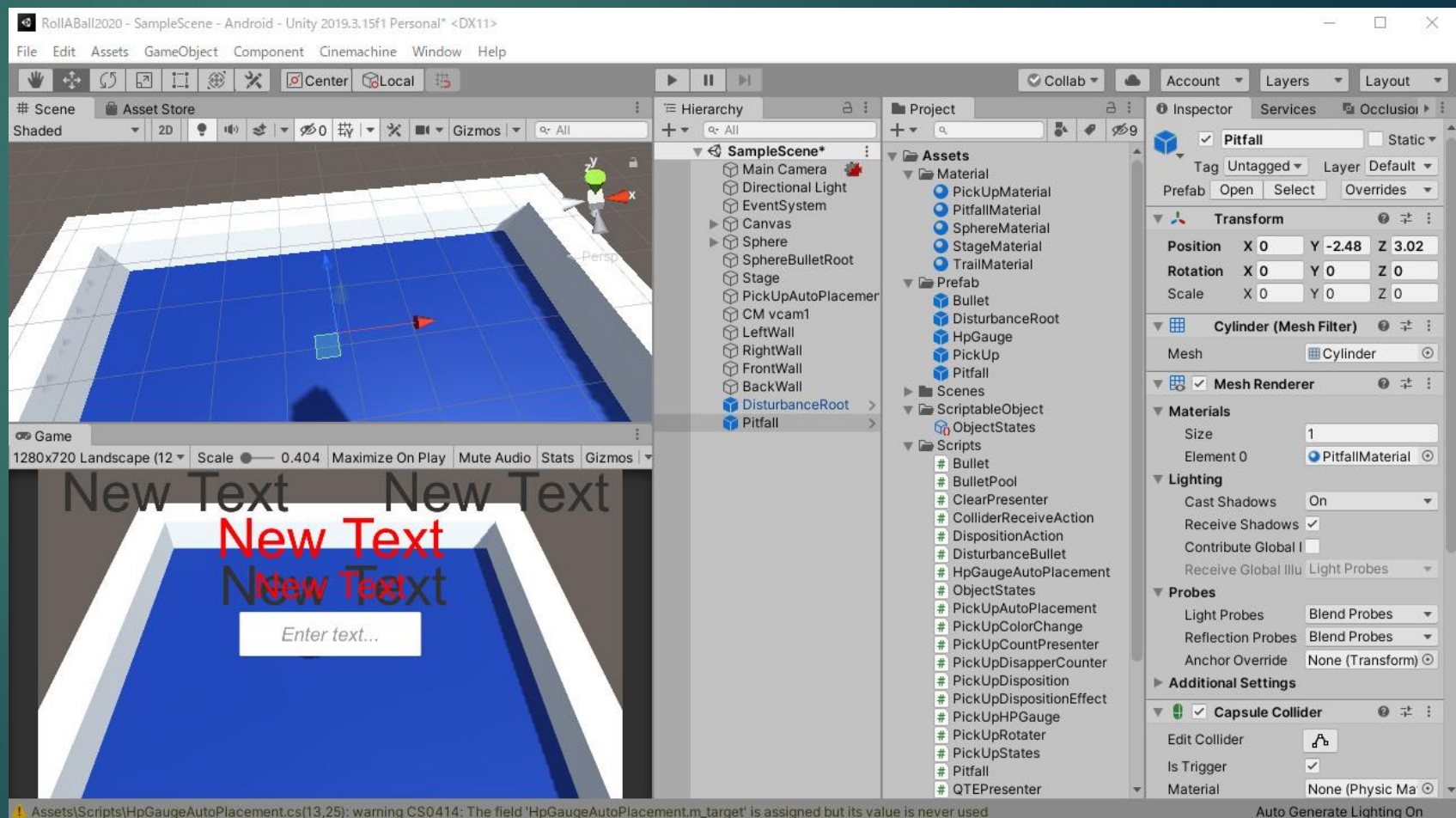
```
RollABall2020 - Microsoft Visual Studio
ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) チーム(M) ツール(T) テスト(S) 分析(N) ウィンドウ(W) ヘルプ(H)
magurodon.10000yen
Pitfall.cs x Bullet.cs SphereAttack.cs DisturbanceBullet.cs PickupDispositionEffect.cs
Assembly-CSharp Pitfall
19: m_transform = GetComponent<Transform>();
20: }
21:
22: private void Update()
23: {
24:     m_safeTime -= Time.deltaTime;
25:
26:     if (m_safeTime > 0)
27:     {
28:         return;
29:     }
30:
31:     m_convergeTime -= Time.deltaTime;
32:     sinScale = Mathf.Sin(Mathf.PI * Time.time);
33:     m_transform.localScale = new Vector3(sinScale * 2, 1, sinScale * 2);
34:
35:     if (m_transform.localScale.x > 1.5f)
36:     {
37:         m_isFall = true;
38:     }
39:     else
40:     {
41:         m_isFall = false;
42:     }
43:
44:     if (m_convergeTime > 0)
45:     {
46:         return;
47:     }
48:     m_transform.localScale = new Vector3(0, 0, 0);
49:     m_convergeTime = 5f;
50:     m_safeTime = 5f;
51:
52: private void OnTriggerEnter(Collider other)
53: {
54: }
```

# Unity

## RollABall

44

- ではUnityに戻って確認をします
- Hierarchy欄から  
Create→3DObject→Cylinderを  
作成します
- これが落とし穴の変わりになる  
穴に見立てたオブジェクトです
- 落とし穴を置きたい位置に移動  
させてください
- Pitfallと名付けてください
- 今回のスクリプト[Pitfall]をD&D  
します

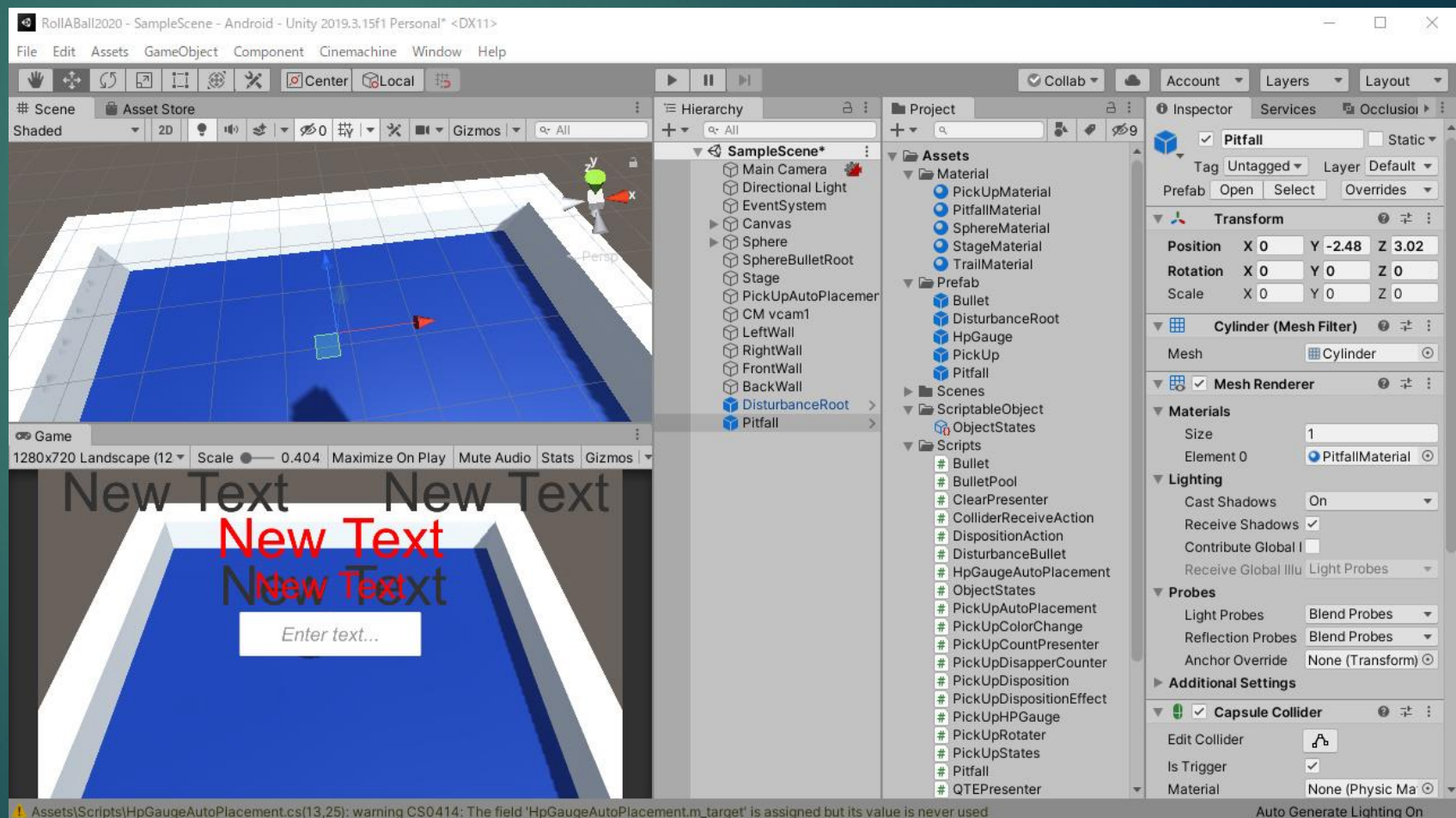


# Unity

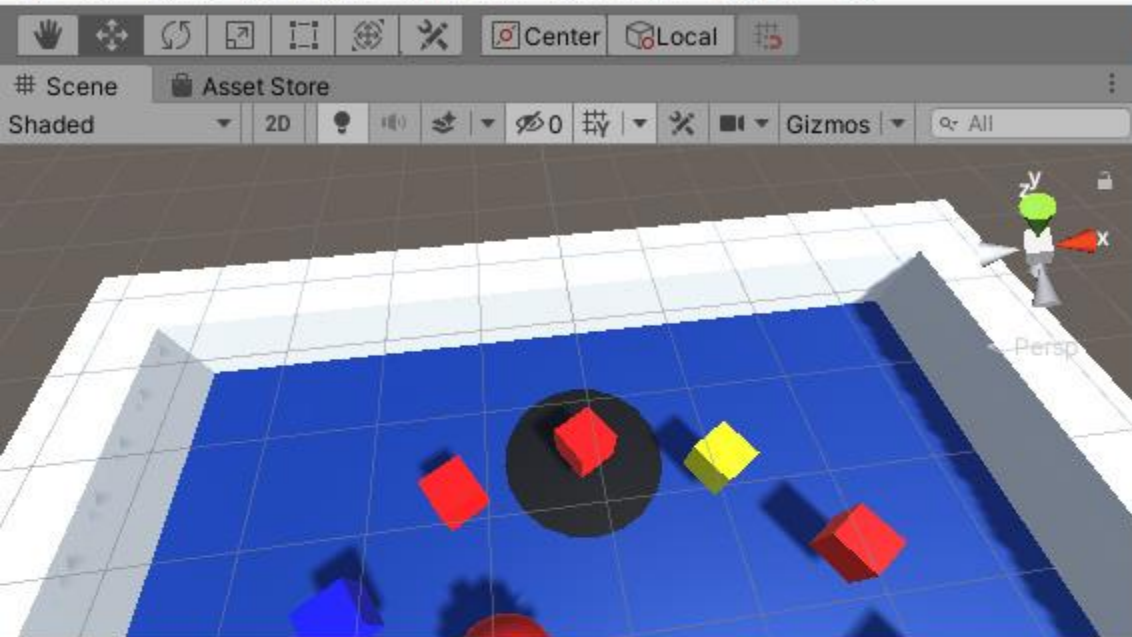
## RollABall

45

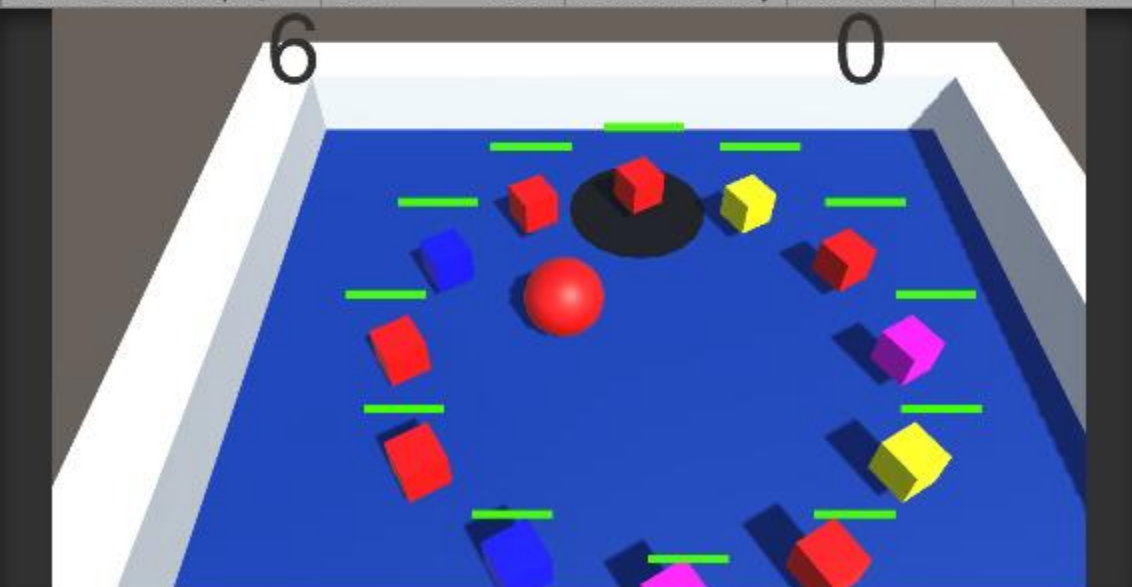
- これもPrefab化しておきましょう
- また、今は真っ白だと思いますので、Project欄からCreate→Materialで[PitfallMaterial]というMaterialを作成してください
- 色を黒にしたら、PrefabのPitfallにD&Dします
- これで落とし穴完成です
- プレイして確認してみましょう







Game 1280x720 Landscape (12) Scale 0.404 Maximize On Play Mute Audio Stats Gizmos

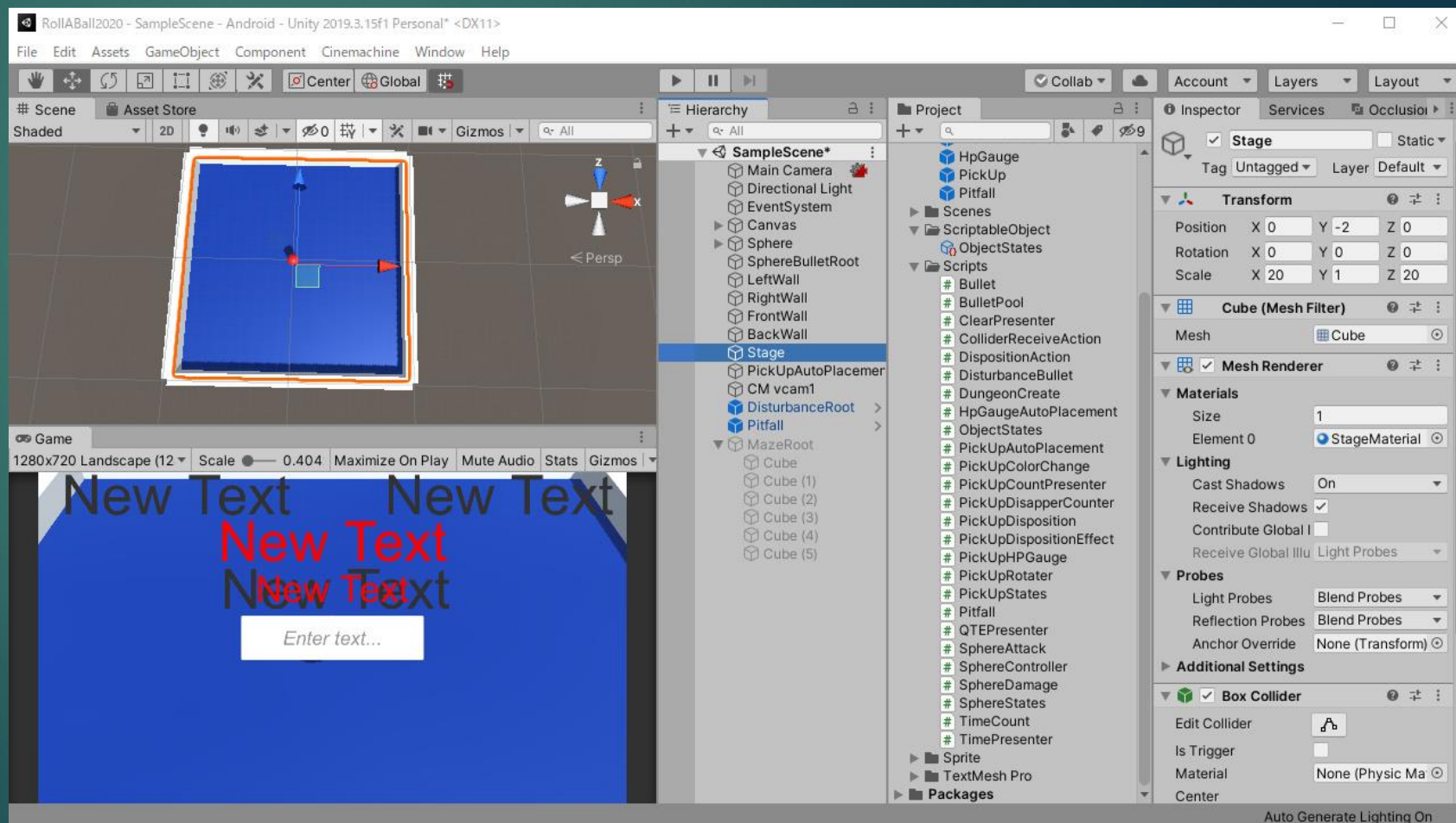


# Unity

## RollABall

47

- これで一通り、妨害の仕様は満たせたと思います
- 最後に、迷路を作っていきます
- その前にステージが手狭になってきたと思いますので、少しスケールをいじってください
- StageのXとZのScaleを20にしてください
- 以前、好きに作っていただいた壁を周りに合わせてください
- また、PickupAutoPlacementのRadiusも8に変更してください



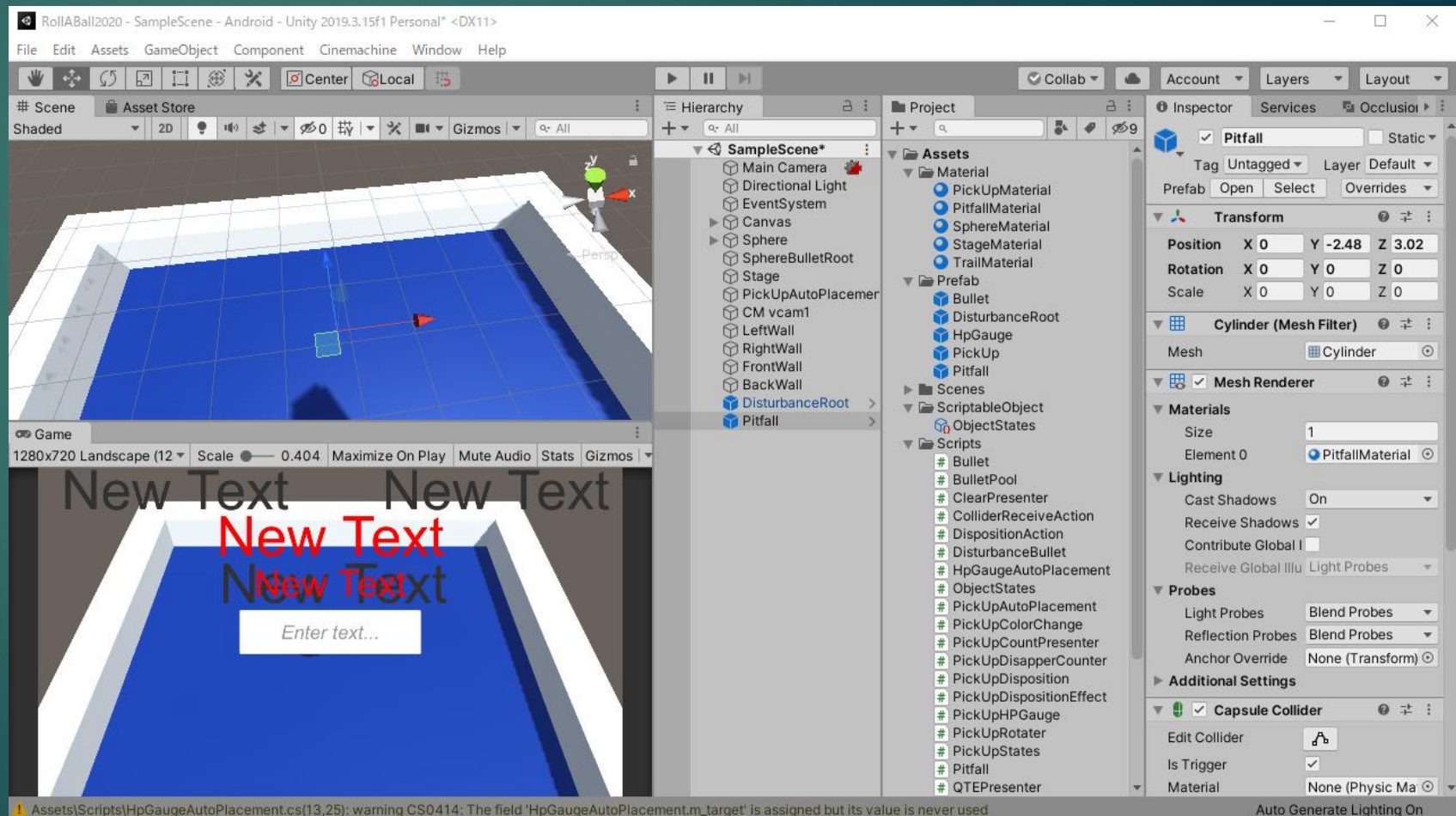


# Unity

## RollABall

48

- 迷路に関してなのですが、一個一個Cubeを置いていくのもいいのですが、やっぱり面倒臭いので再帰処理を使って、と思ったのですが、そのあたりは応用編でやろうと思います
- Create→3DObject→Cubeを各自作成し、TagをWallに変えてください
- また、CreateEmptyで空のGameObjectを作成し、MazeRootと名付けてください



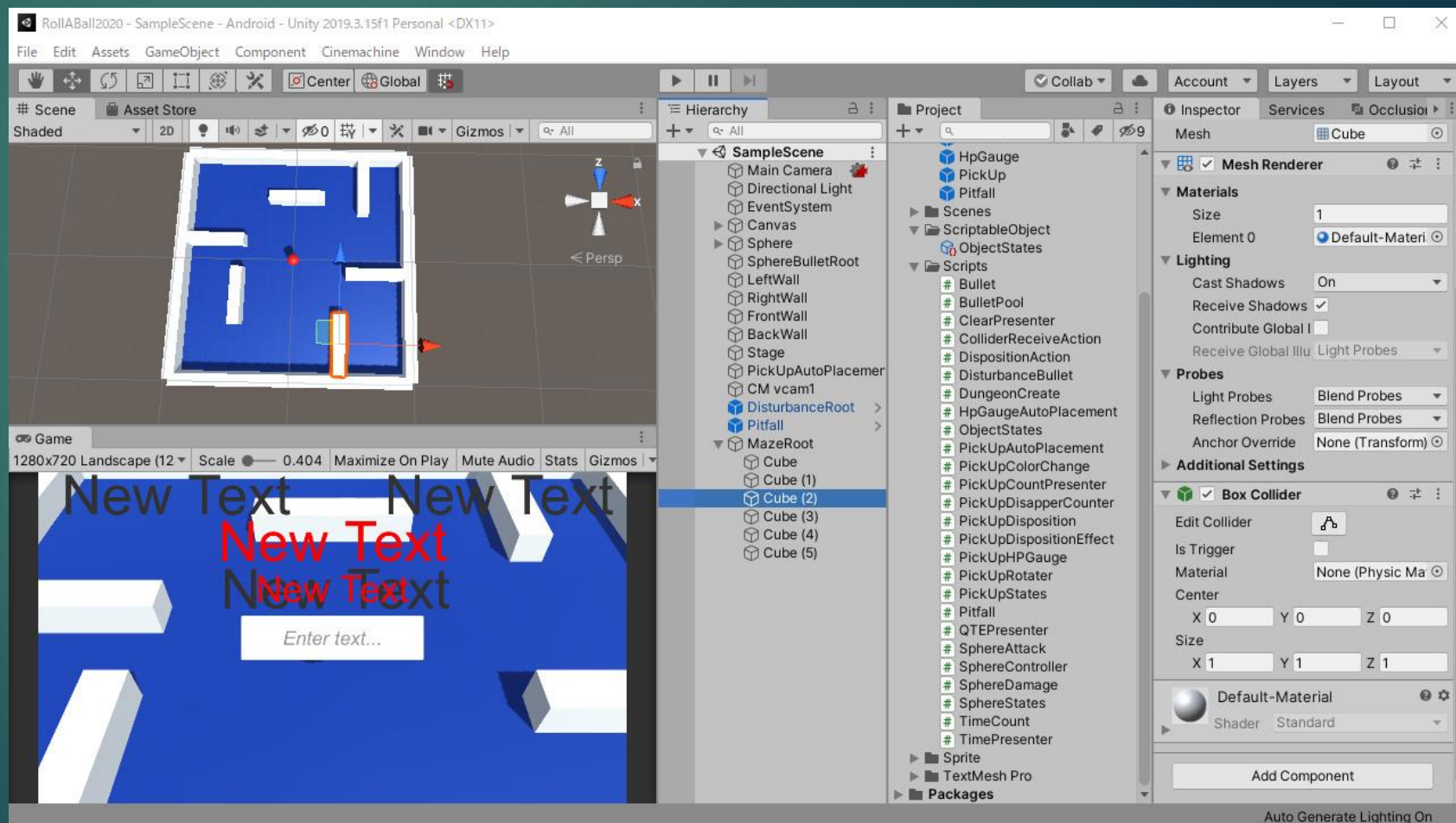


# Unity

## RollABall

49

- MazeRootの子階層にCubeを配置してください
- Scale等を変更して6つくらい自由に置いて迷路を作ってください
- 少し時間を取りますので、ユーザーが面白いと思う迷路に仕上げてってください

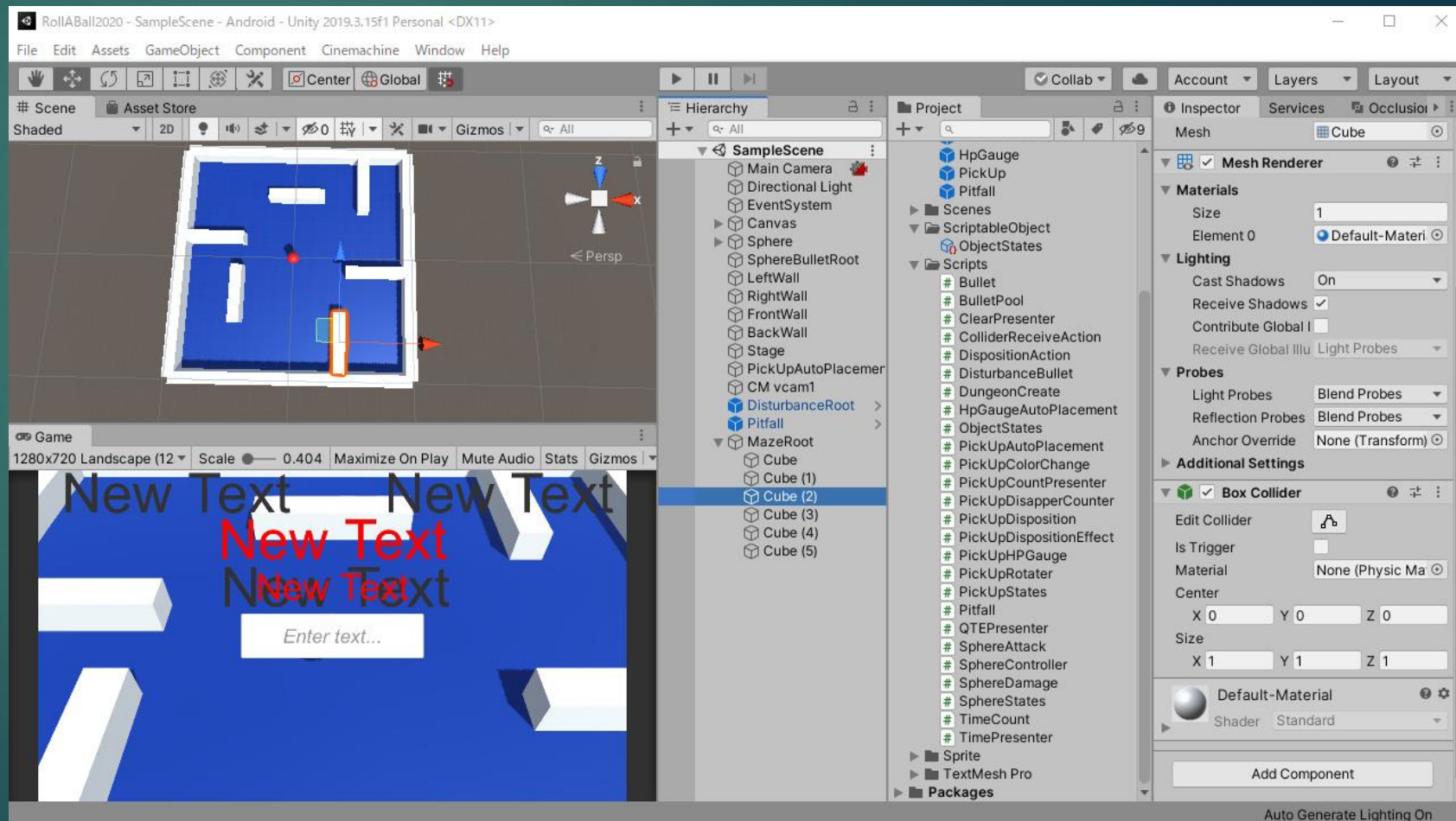


# Unity

## RollABall

50

- そういえば、ゲームのクリアは表示されてもゲームオーバーの演出がなかったので、ここで作成したいと思います
- GameOverPresenterをスクリプトで作成してください
- このスクリプトの役目はSphereのHpが0になったら、ゲームオーバーと画面に表示することです
- ちょうどTimeScaleを使いましたので、ゲームオーバーのタイミングで時間を止めてみましょう
- 今回は各自、書いてみましょうか

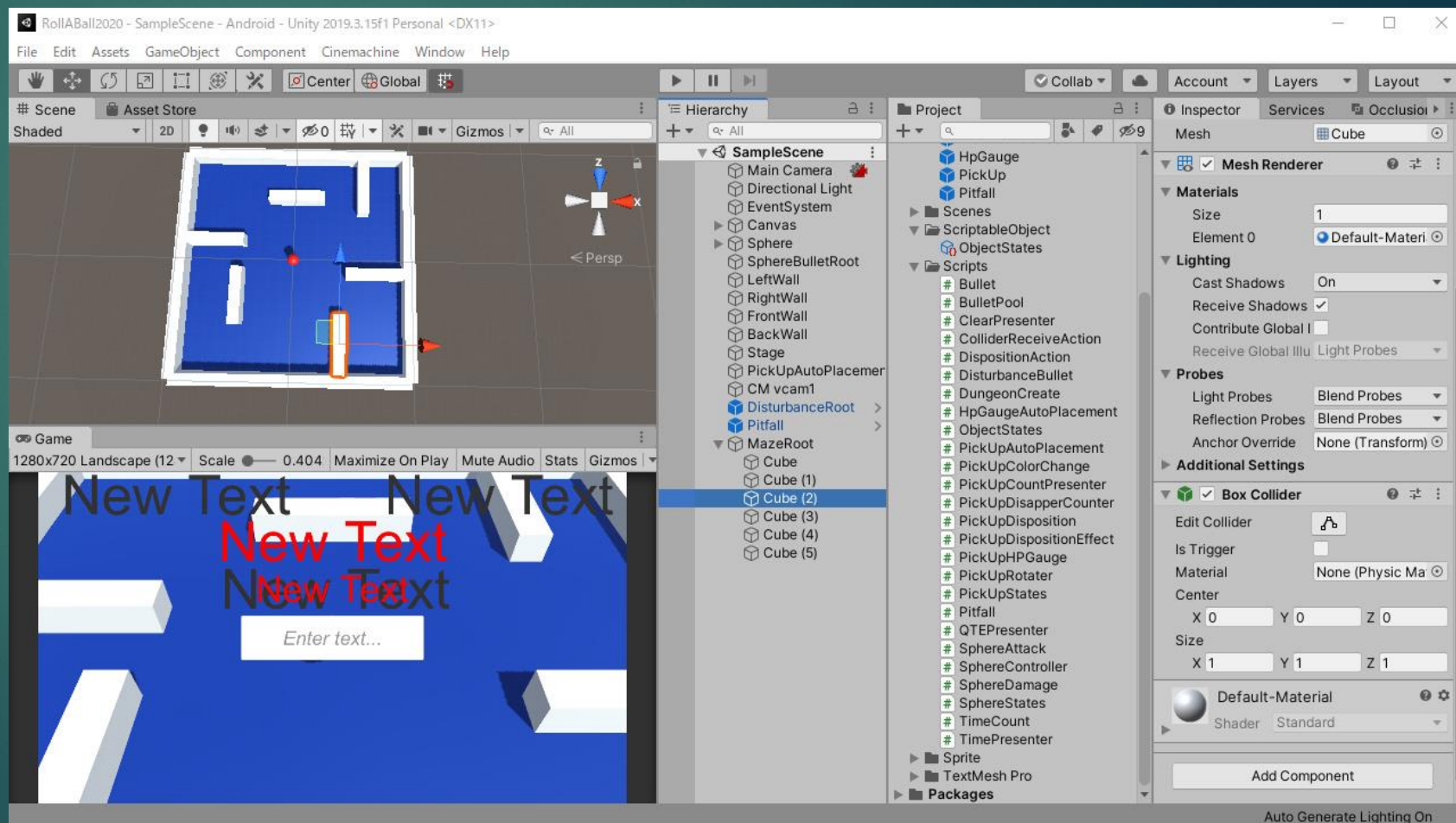


# Unity

## RollABall

51

- ※ヒント
- SphereStatesの中にSphereのHpがあります
- 時間を止める処理はQTEPresenterの中にあります
- Textに表示する方法は色々ありますので、各自見つけてください
- あと、Pitfallのスク립トに追加で、落とし穴に落ちたとき、SphereのHpにAddDamageで10のダメージを与えてみてください



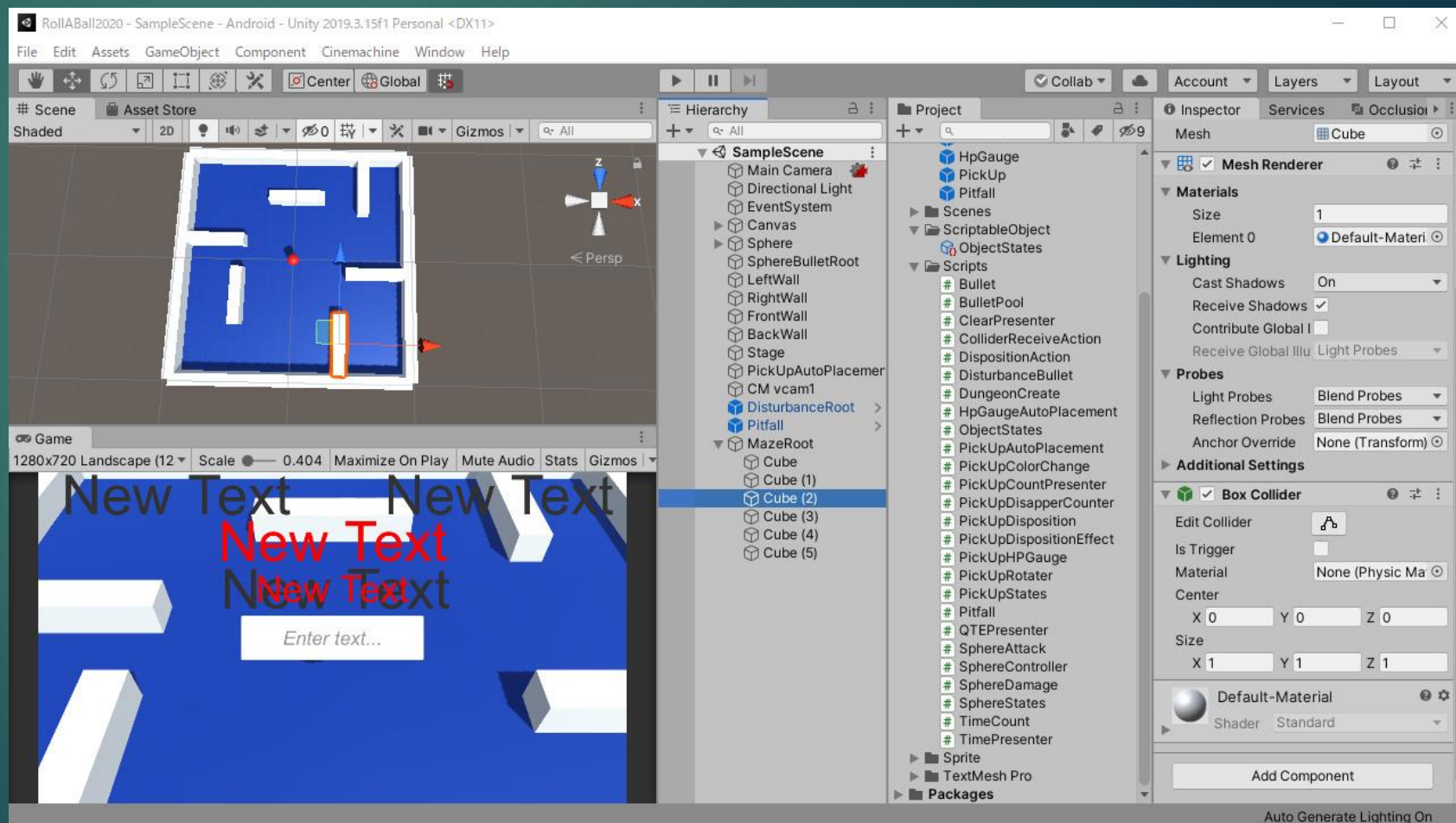


# Unity

## RollABall

52

- 書きあげられましたら、Unityで設定してみてください
- ここまで出来たら、本日の授業は終了とします
- お疲れさまでした！
- 来週に関してですが、Unityではなく、実際のゲーム業界、IT業界で使われているGitというものを学びます
- Unityを学習するうえで例えば、ミスったり、PCの情報が消えていたりすることがあると思いますが、それをタイムマシンのように変更履歴をたどって復元してくれるツールです



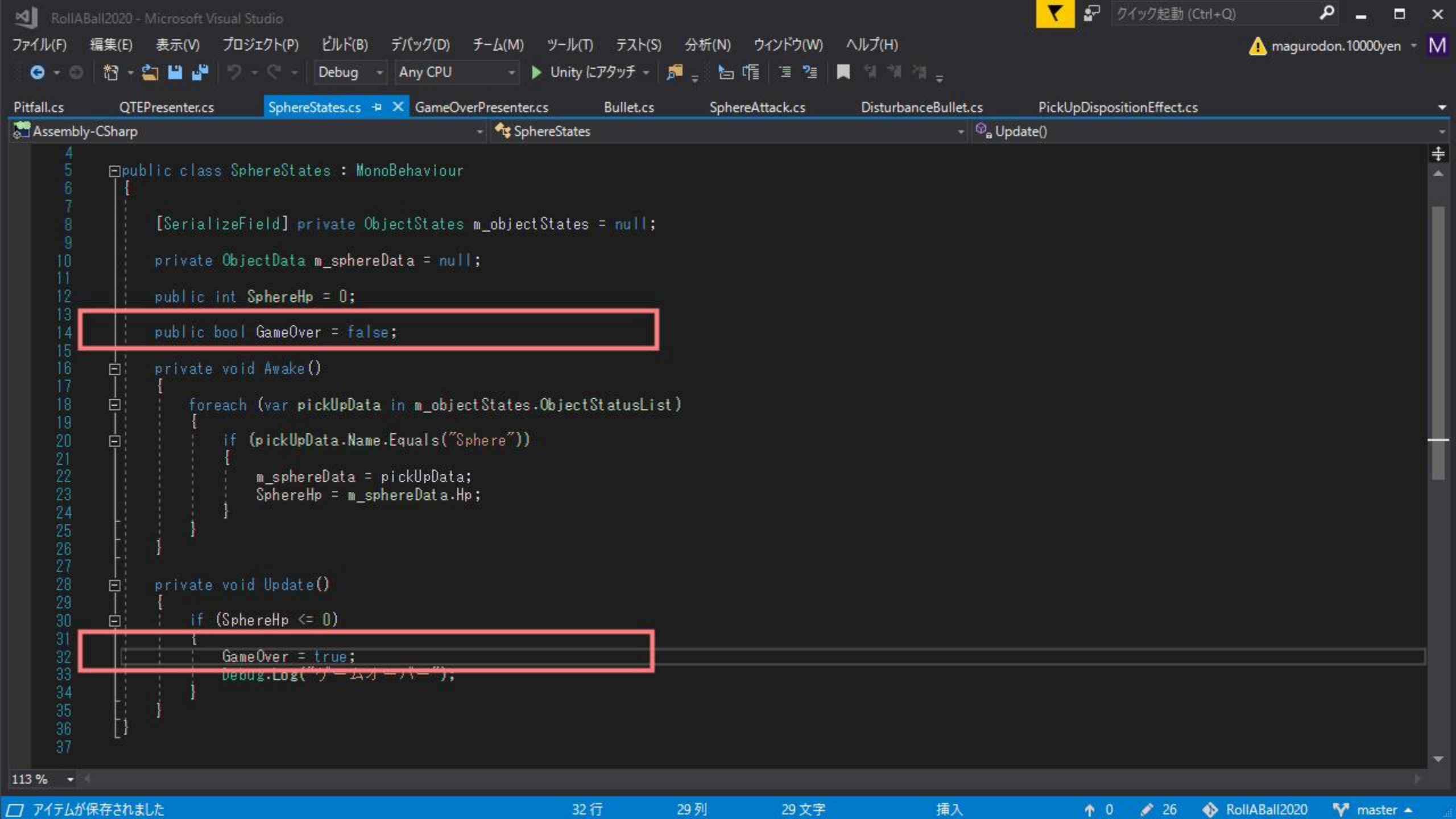
# Tips : GitHub

53

- ゲーム業界のみならず、IT業界では広く使われているツールです
- ザックリいうと変更履歴を保存しておけるので、変更したタイミングに処理や手を加えた分を巻き戻すことができます
- 創作活動やプレゼン資料の作成等にも効果的に使う事ができます
- 変更単位で巻き戻したりすることができるので、Aという機能を追加した  
はいいものの、使わなければそのAのみを変更履歴から除外することができます

```
1 using UnityEngine;
2 using UnityEngine.UI;
3
4 public class GameOverPresenter : MonoBehaviour
5 {
6     [SerializeField] private SphereStates m_sphereStates = null;
7     [SerializeField] private Text m_gameOverText = null;
8
9     private void Start()
10     {
11         m_gameOverText.text = string.Empty;
12     }
13
14     private void Update()
15     {
16         if (m_sphereStates.GameOver)
17         {
18             Time.timeScale = 0f;
19             m_gameOverText.text = "ゲームオーバー";
20         }
21     }
22 }
23
```





```
37         m_isFall = true;
38     }
39     else
40     {
41         m_isFall = false;
42     }
43
44     if (m_convergeTime > 0)
45     {
46         return;
47     }
48     m_transform.localScale = new Vector3(0, 0, 0);
49     m_convergeTime = 5f;
50     m_safeTime = 5f;
51 }
52
53 private void OnTriggerStay(Collider other)
54 {
55     if (other.gameObject.CompareTag("Player") && m_isFall)
56     {
57         other.GetComponent<SphereDamage>().AddSphereDamage(10);
58         m_stageCollider.enabled = false;
59     }
60 }
61
62
```