

2020年度 Unity講座(基礎編)



06回目

講師：幸田 将伍 (@MagurodonDev)

今回の講義の目的

2

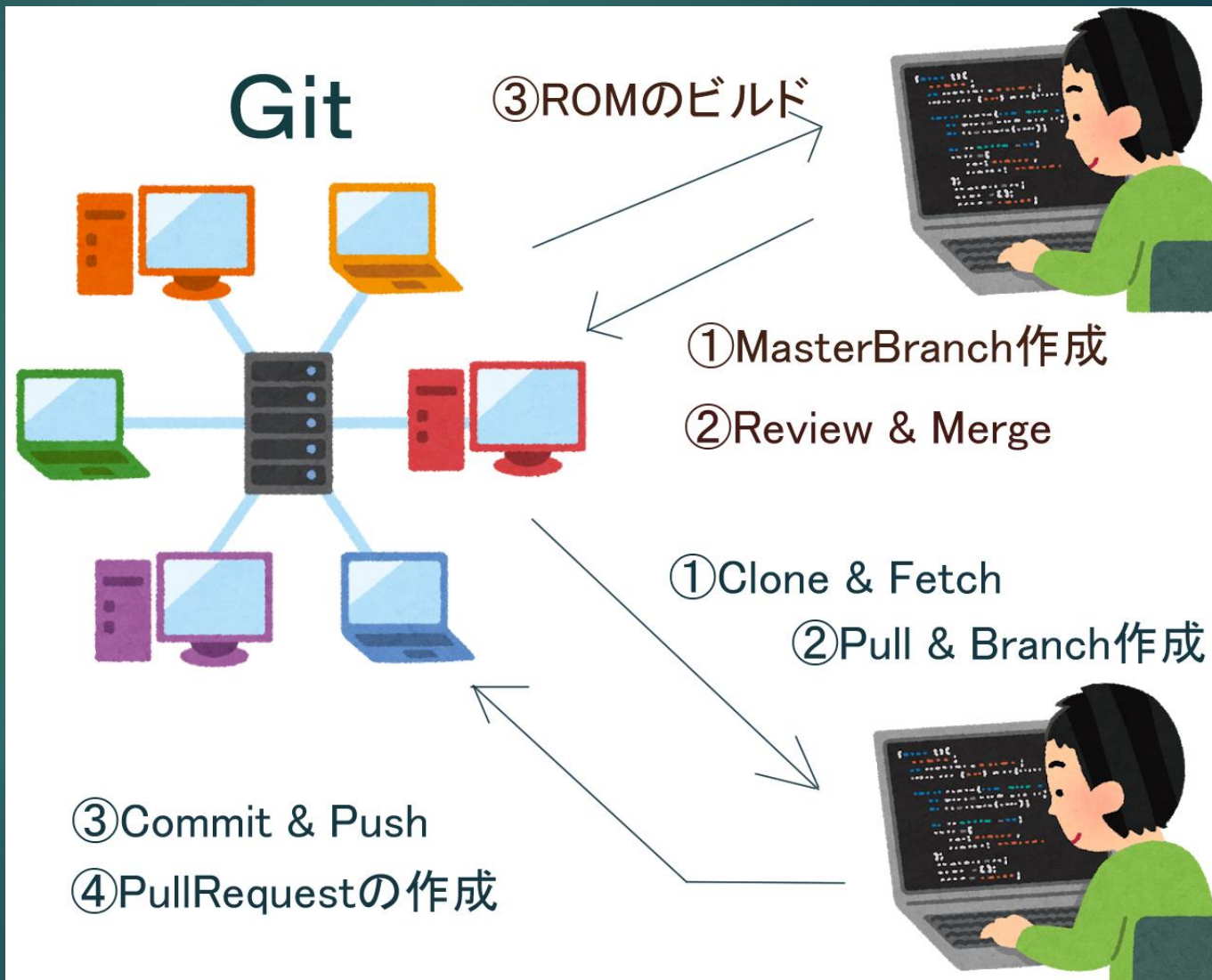
- ▶ プログラムを自分で読めるようになる
- ▶ Unityを使って自分が実現したいことをできるようになる
- ▶ 自分一人でもゲームを作成できるレベルになる
- ▶ Unityの活用事例を学び、自分の進路に役立てる
- ▶ 実際のエンジニアがどういった仕事の進め方をしているかを知る
- ▶ ゲーム会社のクライアントエンジニアとして就職できるレベルになる

一緒にレベルアップして行きましょう！

Unity

ハイアンドロー

- 前はGitHubのサービスの使い方、Gitの基礎用語等を学びました
- 今回からはハイアンドローを作っていきます
- 今回に関してはロジックの部分を作成していきます
- まずハイアンドローのルールを策定します

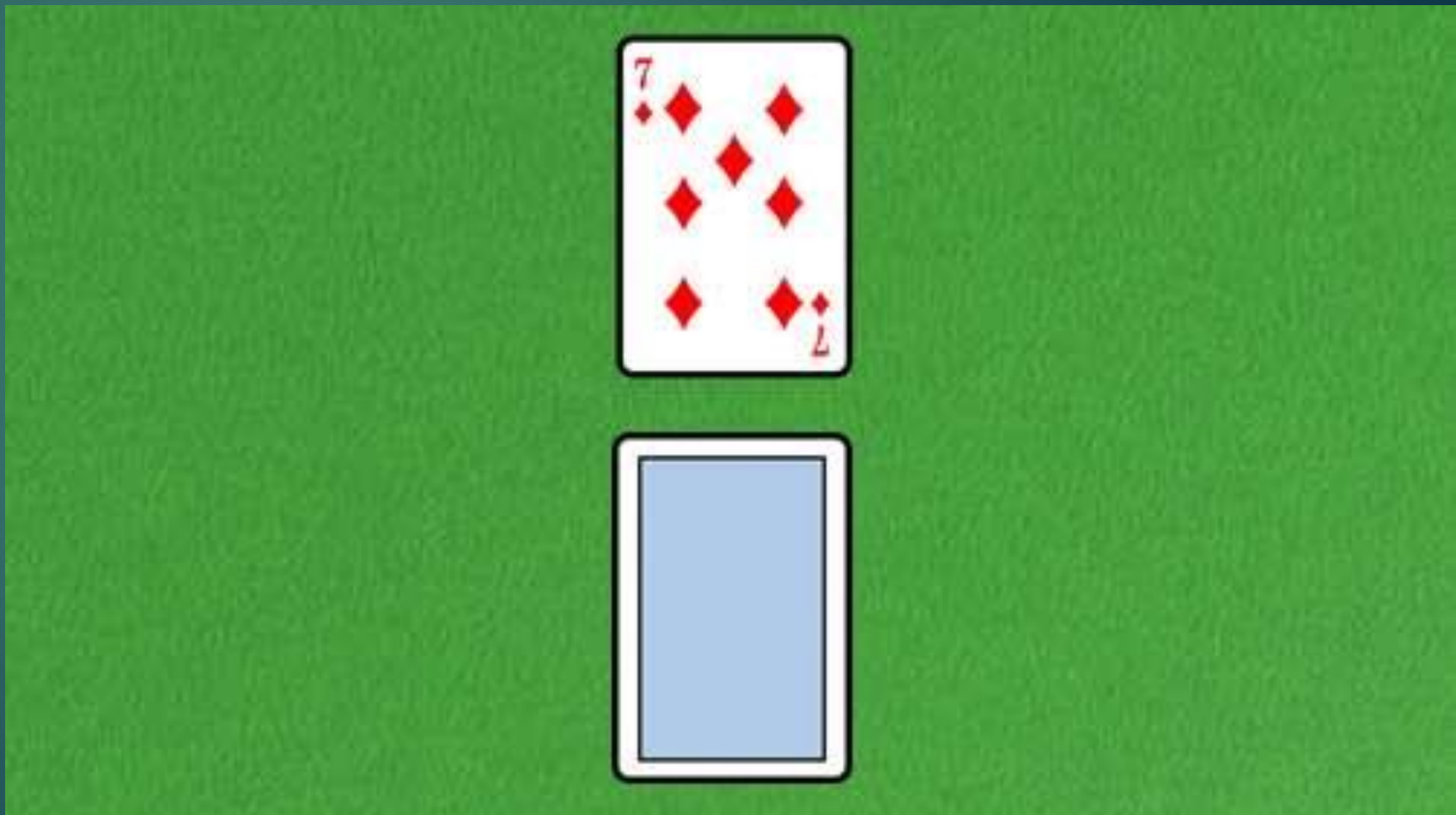


Unity

4

ハイアンドロー

- ルール(仕様)
- 使うのはジョーカーを除いたデッキ(52枚)
- プレイヤーとCPUで2つに分ける
- CPUがカードを表で出す
- プレイヤーがカードを伏せて出す
- プレイヤーは伏せられたカードがCPUの出したカードより上か下を予想する
- 当たっていればプレイヤーがカードを取得

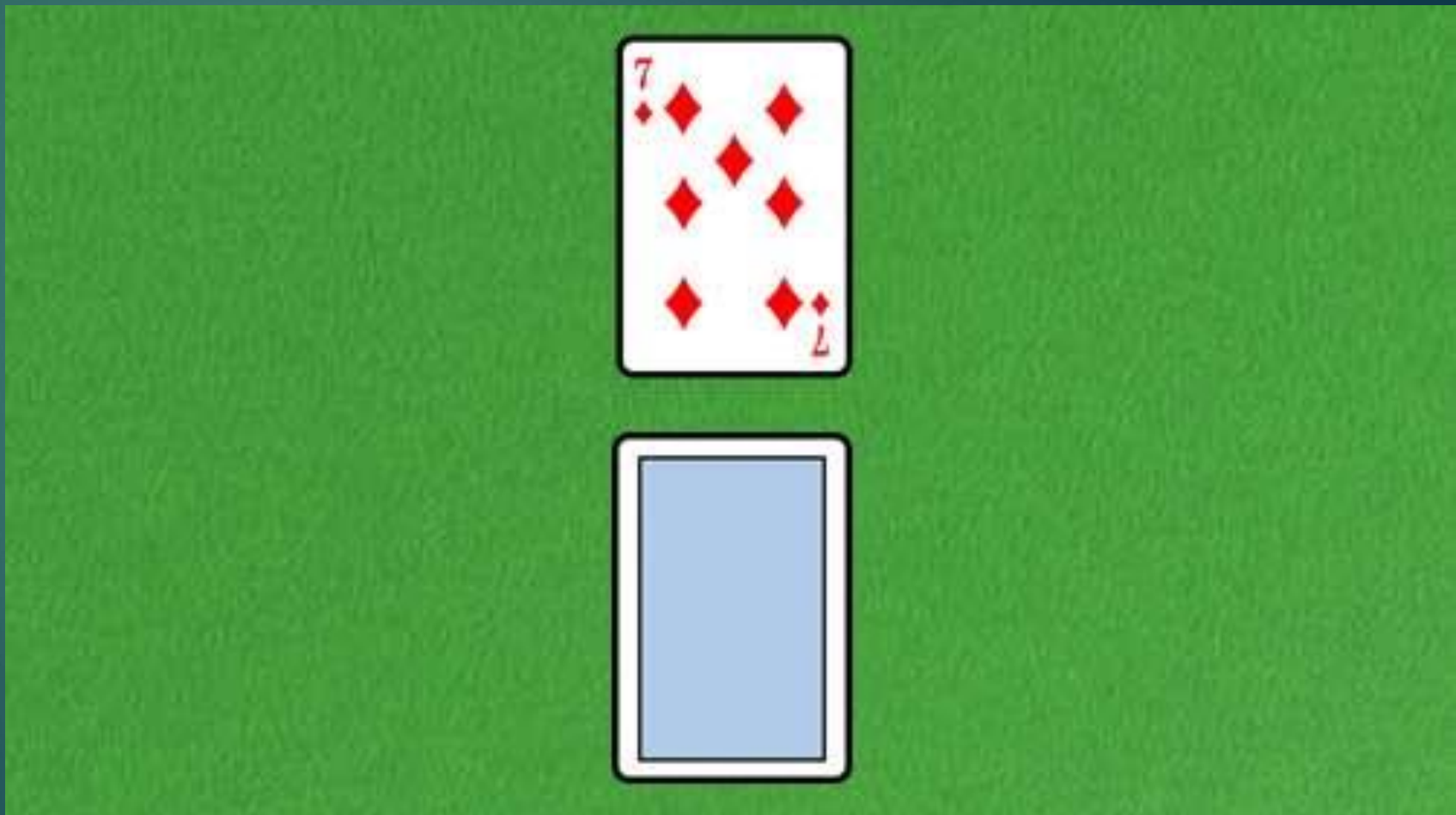


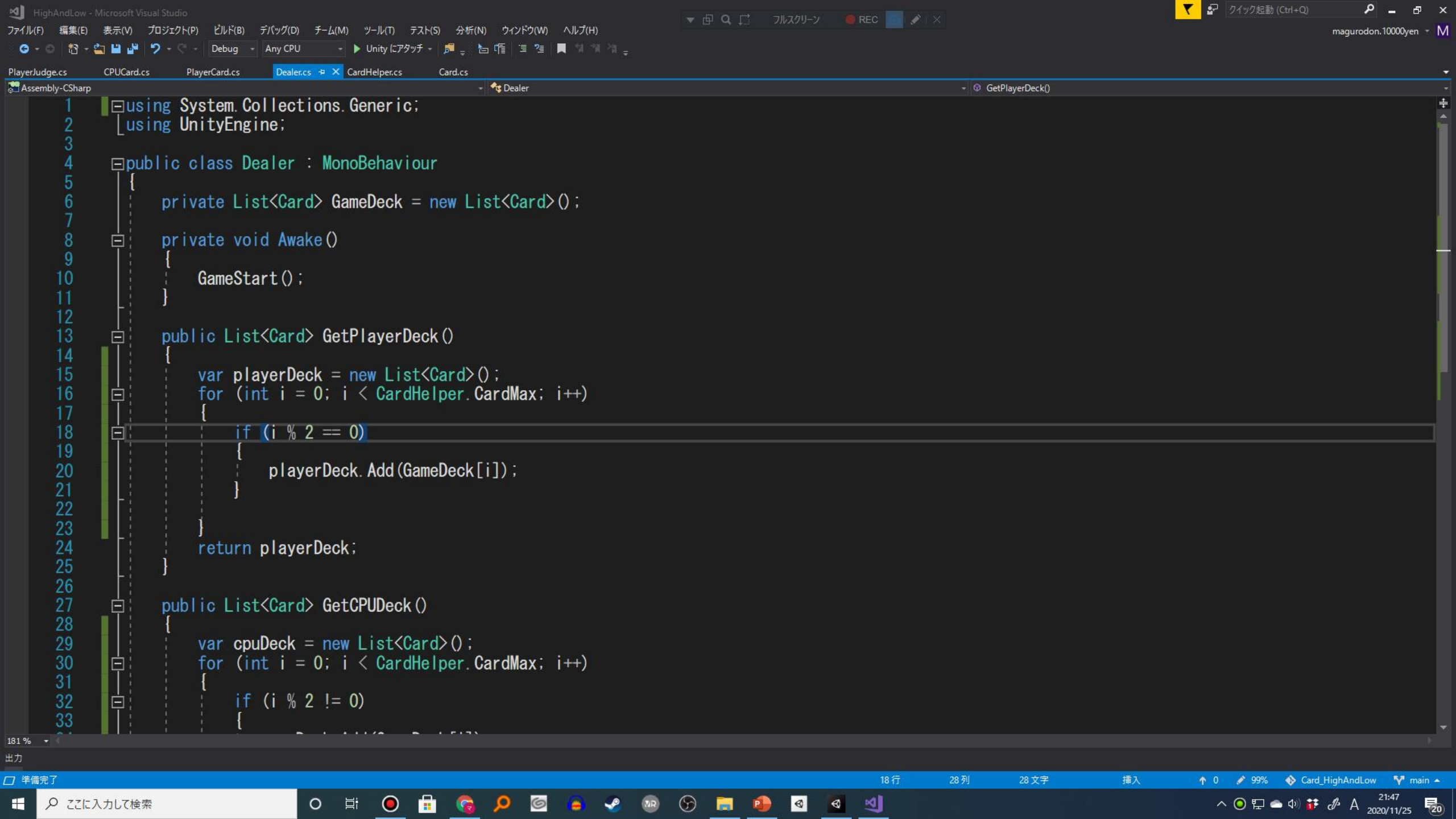
Unity

5

ハイアンドロー

- 前はデッキを作るところまではできていたと思うので、今回は
 - プレイヤーとCPUで2つに分ける
 - CPUがカードを表で出す
 - プレイヤーがカードを伏せて出す
 - プレイヤーは伏せられたカードがCPUの出したカードより上か下を予想する
 - ここまで進めようと思います
-
- では早速ですが、Dealer.csを作成しましょう
 - 役割はデッキからPlayerとCPUに1枚ずつ山札を分けることです
 - For文を使って表現してみてください





HighAndLow - 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)

Debug Any CPU Unity にアタッチ

PlayerJudge.cs CPUCard.cs PlayerCard.cs Dealer.cs CardHelper.cs Card.cs

Assembly-CSharp Dealer Awake()

25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

```
    }  
  
    public List<Card> GetCPUDeck()  
    {  
        var cpuDeck = new List<Card>();  
        for (int i = 0; i < CardHelper.CardMax; i++)  
        {  
            if (i % 2 != 0)  
            {  
                cpuDeck.Add(GameDeck[i]);  
            }  
        }  
        return cpuDeck;  
    }  
  
    public void GameStart()  
    {  
        GameDeck = Deck.GetDeck();  
        GameDeck = Deck.ShuffleDeck(GameDeck);  
    }  
}
```

181 % 出力

準備完了 9行 6列 6文字 挿入 0 99% Card_HighAndLow main

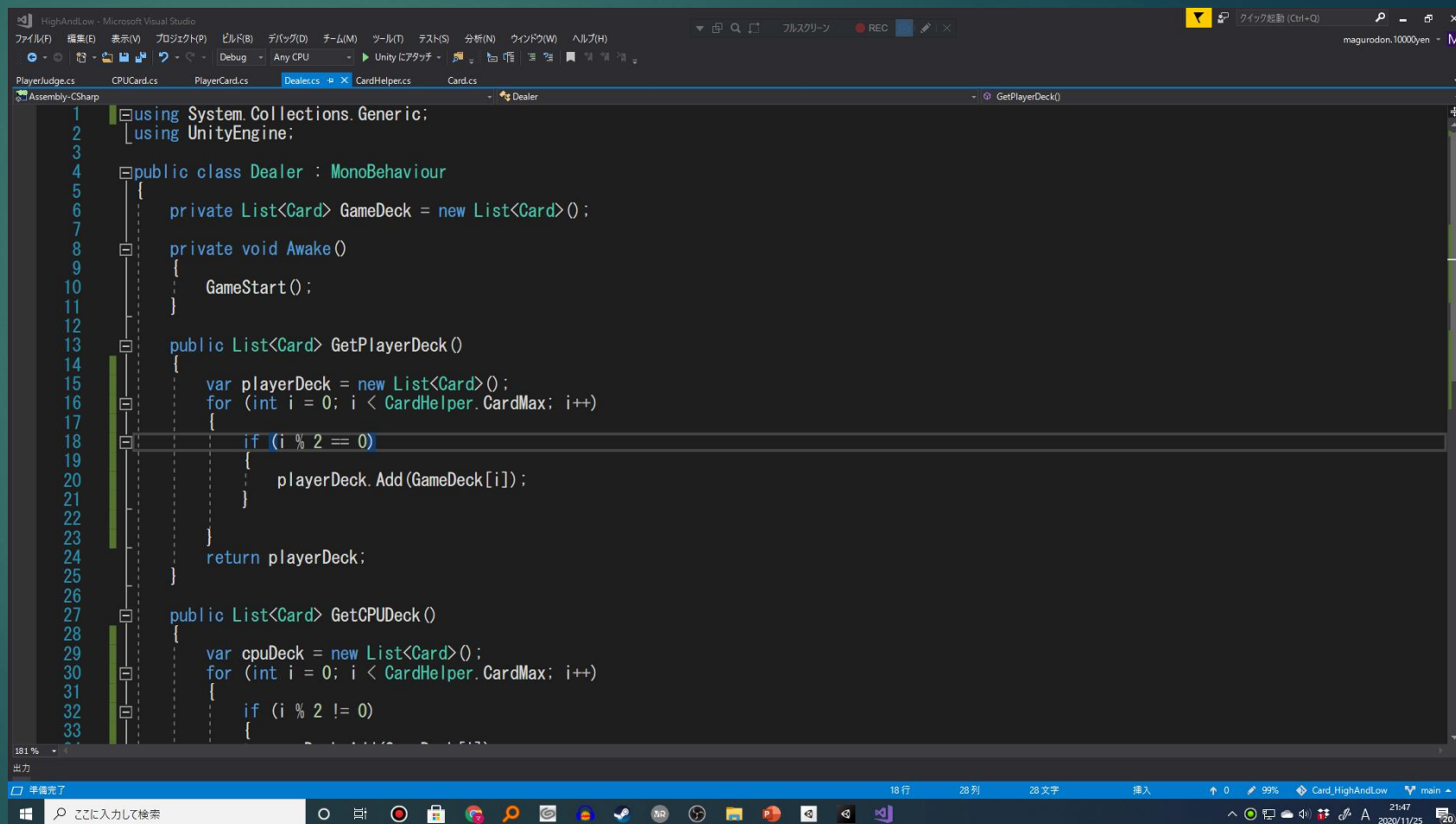
ここに入力して検索 21:47 2020/11/25

Unity

ハイアンドロー

8

- ※ = new List<hoge hoge>
- List<hoge hoge>型の変数を定義する時、必ずどこかで初期化する必要があります
- 基本的には宣言してすぐに初期化するといいいでしょう
- 初期化する方法が上記の記述方法です
- ※ List.Add(hoge hoge)
- Listに要素を追加する時に使うメソッドです
- hoge hoge型のListにhoge hogeを加えるということです



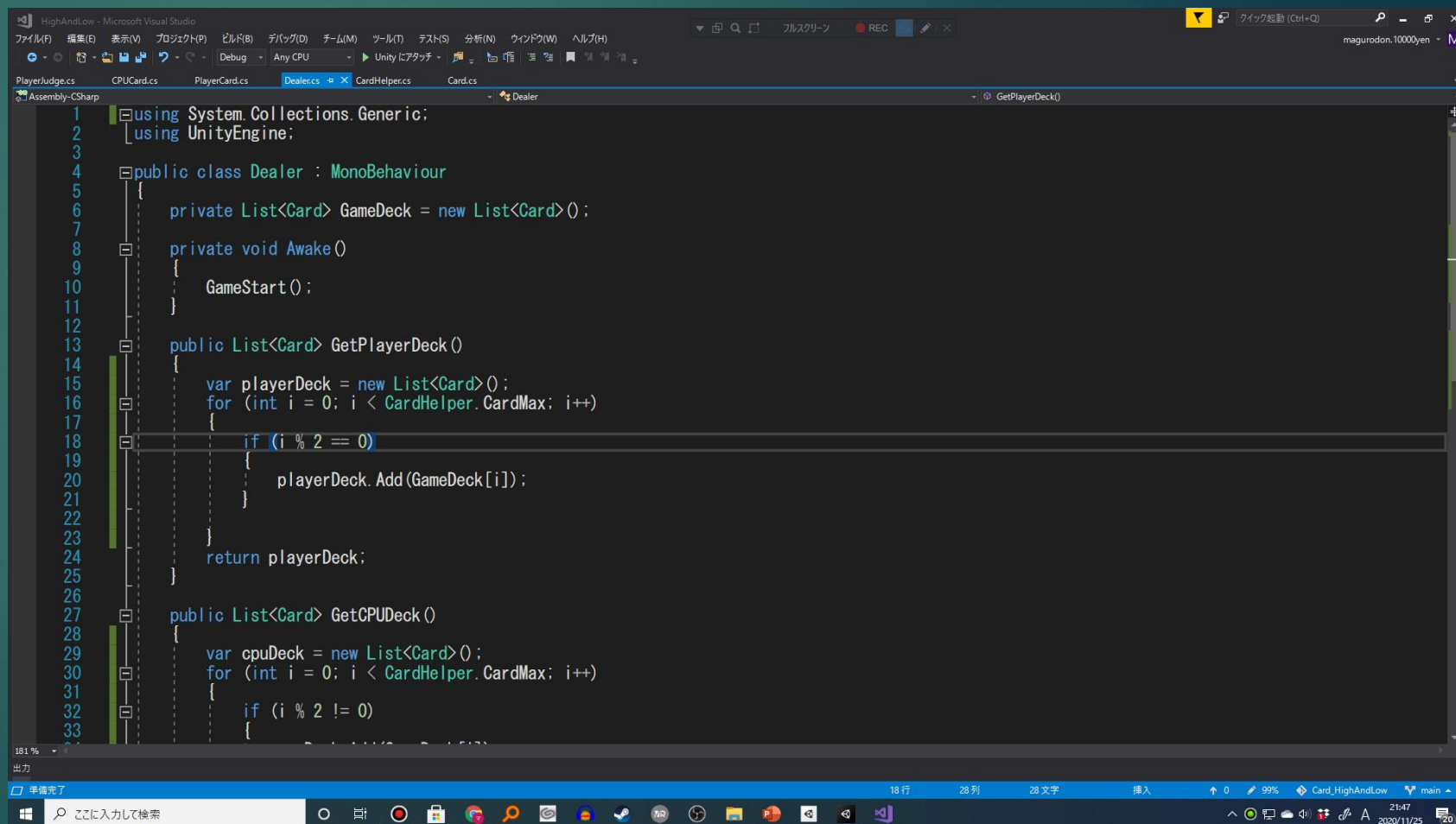
```
1 using System.Collections.Generic;
2 using UnityEngine;
3
4 public class Dealer : MonoBehaviour
5 {
6     private List<Card> GameDeck = new List<Card>();
7
8     private void Awake()
9     {
10         GameStart();
11     }
12
13     public List<Card> GetPlayerDeck()
14     {
15         var playerDeck = new List<Card>();
16         for (int i = 0; i < CardHelper.CardMax; i++)
17         {
18             if (i % 2 == 0)
19             {
20                 playerDeck.Add(GameDeck[i]);
21             }
22         }
23         return playerDeck;
24     }
25
26     public List<Card> GetCPUDeck()
27     {
28         var cpuDeck = new List<Card>();
29         for (int i = 0; i < CardHelper.CardMax; i++)
30         {
31             if (i % 2 != 0)
32             {
33                 // ...
34             }
35         }
36     }
37 }
```


Unity

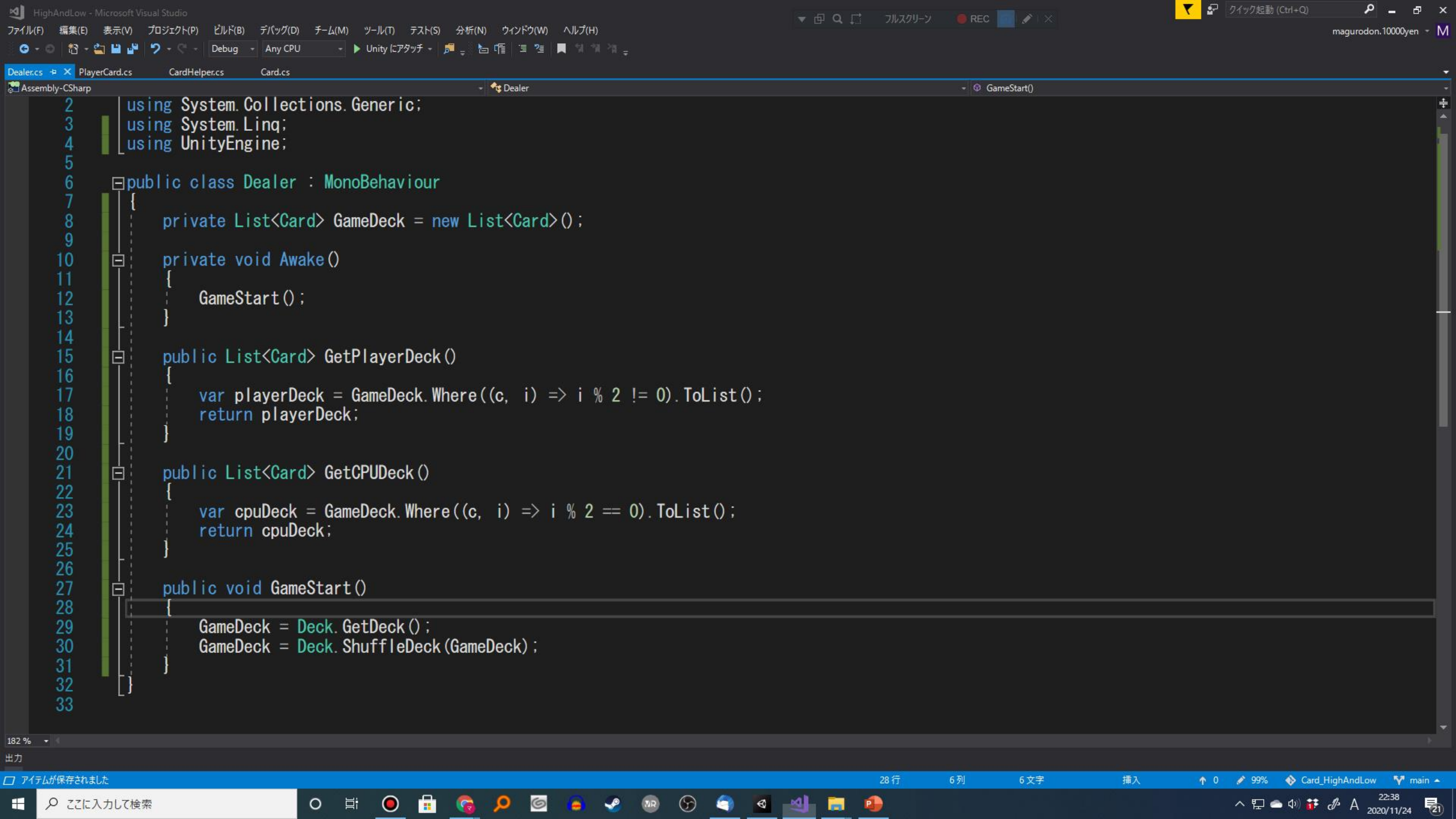
ハイアンドロー

9

- ※List[int]
- Listや配列の番数を指定して、その要素を取得する書き方です
- List[0]だったらListの0番目の要素、List[5]だったらListの5番目の要素を取得します
- ここで注意なのですが、List[-1]や、List[Listの長さより大きい数字]だった場合、エラーとなりますので注意しましょう
- 因みになのですが、linqを使うと次の実装方法のようにすっきり書くことができます



```
1 using System.Collections.Generic;
2 using UnityEngine;
3
4 public class Dealer : MonoBehaviour
5 {
6     private List<Card> GameDeck = new List<Card>();
7
8     private void Awake()
9     {
10         GameStart();
11     }
12
13     public List<Card> GetPlayerDeck()
14     {
15         var playerDeck = new List<Card>();
16         for (int i = 0; i < CardHelper.CardMax; i++)
17         {
18             if (i % 2 == 0)
19             {
20                 playerDeck.Add(GameDeck[i]);
21             }
22         }
23         return playerDeck;
24     }
25
26     public List<Card> GetCPUDeck()
27     {
28         var cpuDeck = new List<Card>();
29         for (int i = 0; i < CardHelper.CardMax; i++)
30         {
31             if (i % 2 != 0)
32             {
33                 cpuDeck.Add(GameDeck[i]);
34             }
35         }
36         return cpuDeck;
37     }
38 }
```

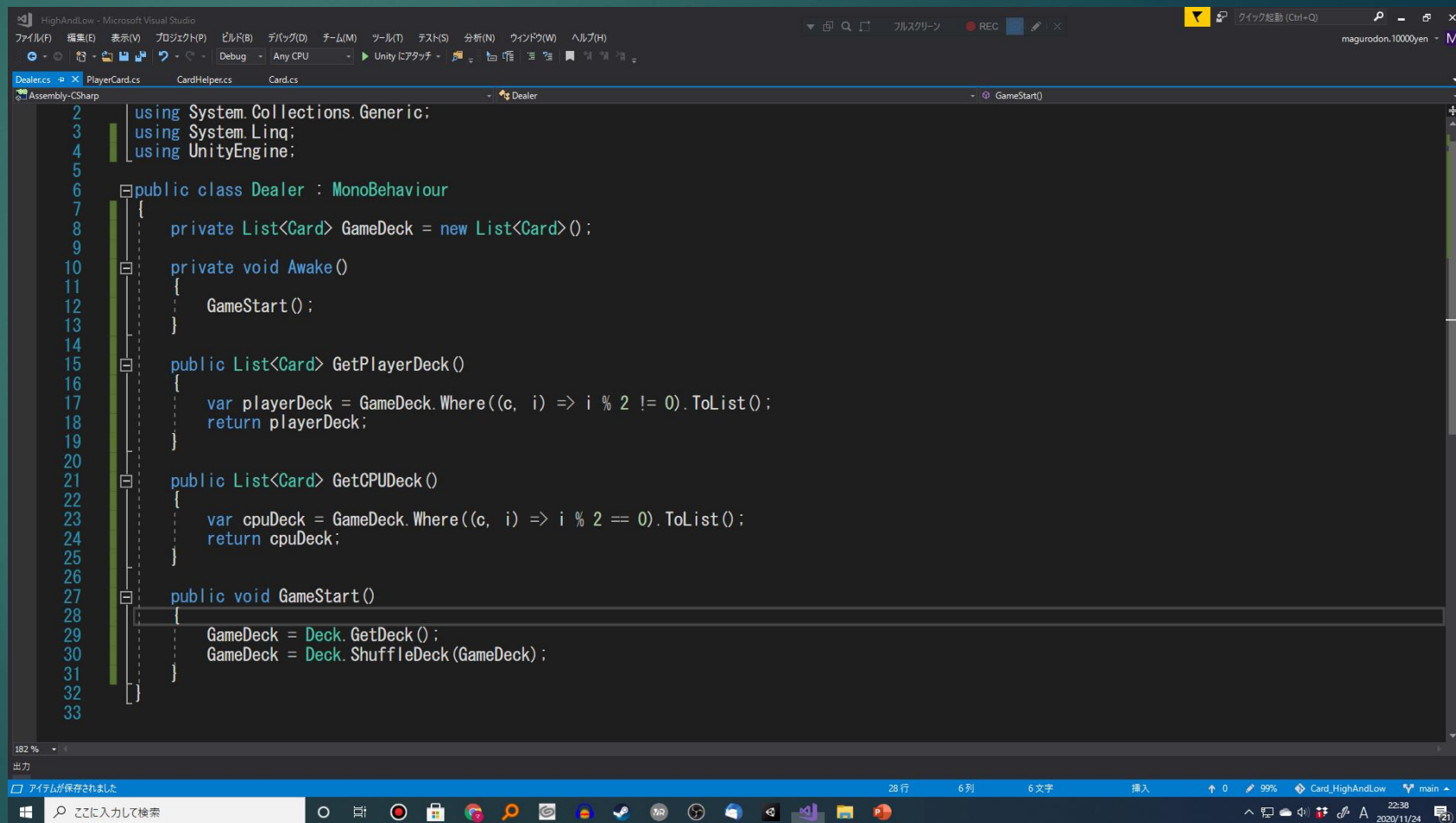


Unity

ハイアンドロー

11

- ※List.Where(hoge => hoge)
- Linqを使うとWhere句を使う事ができます
- これは英単語の示す通り、どの部分を取ってほしいのですか？ といったところです
- 今回はWhere句のWhere(Hogehoge,int)を使い、intの部分でindexを取得してます
- Indexが2で割り切れる数(余剰が0)だった場合は偶数番目
- Indexが2で割り切れない数(余剰が0ではない数)だった場合は奇数番目
- 上記の振り分けでplayerとCpuにデッキを渡しています



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

public class Dealer : MonoBehaviour
{
    private List<Card> GameDeck = new List<Card>();

    private void Awake()
    {
        GameStart();
    }

    public List<Card> GetPlayerDeck()
    {
        var playerDeck = GameDeck.Where((c, i) => i % 2 != 0).ToList();
        return playerDeck;
    }

    public List<Card> GetCPUDeck()
    {
        var cpuDeck = GameDeck.Where((c, i) => i % 2 == 0).ToList();
        return cpuDeck;
    }

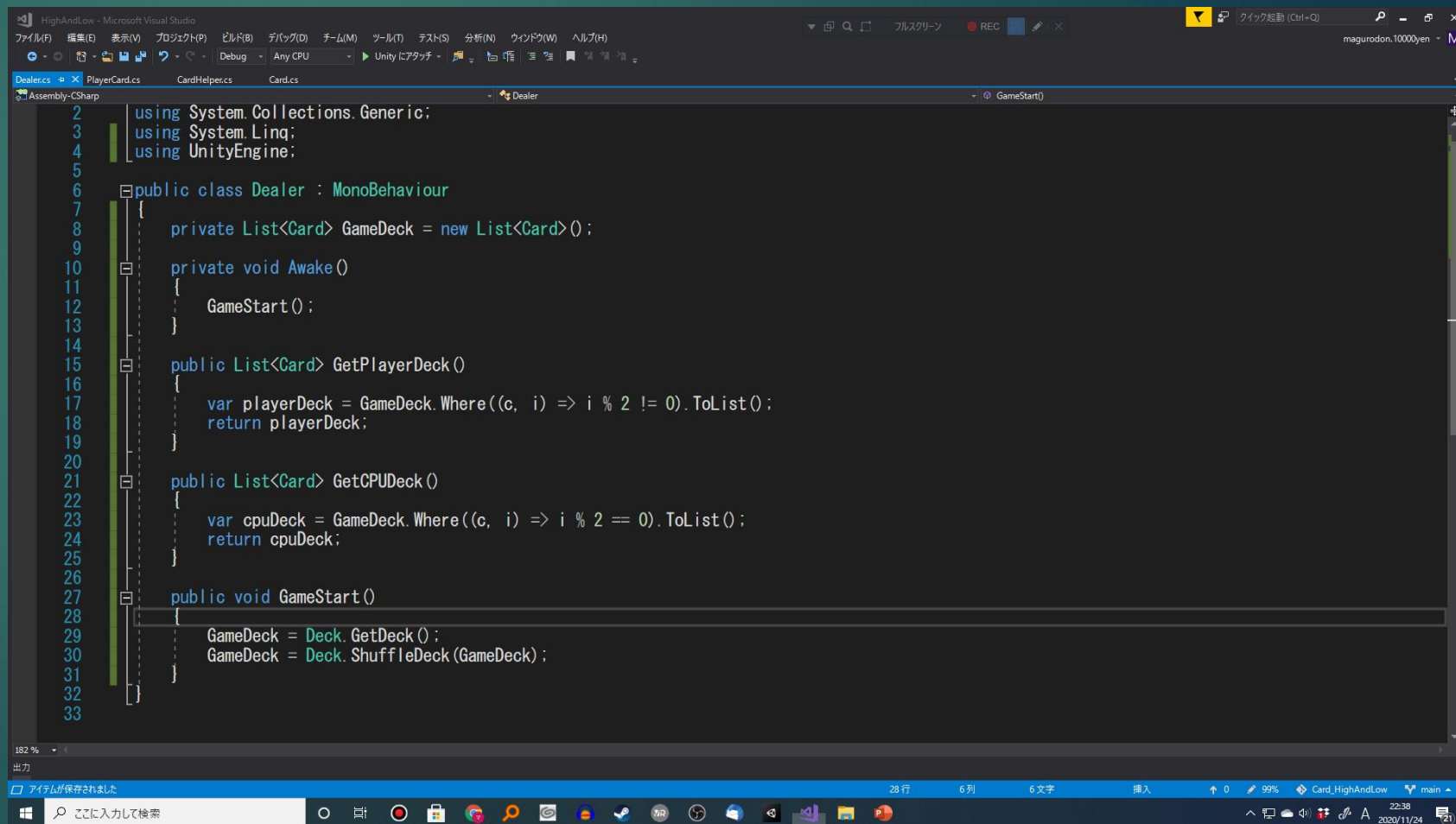
    public void GameStart()
    {
        GameDeck = Deck.GetDeck();
        GameDeck = Deck.ShuffleDeck(GameDeck);
    }
}
```

Unity

12

ハイアンドロー

- ここでListで遊んでみましょう
- 次の配り方をしたい場合、このメソッドの中身を書き換えてみましょう！
- ①for文を使用してSpadeとClubはPlayerに行くように、HartとDiaはCPUに行くようにしてください
- ②for文を使用して、0番目から25番目まではPlayerに、26番目から51番目まではCPUに行くようにしてください
- ③Playerが必ずCPUより高くなるように
- ヒント：ShuffleDeckを消して、上から26枚とれば、必ず高くなるはずです



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

public class Dealer : MonoBehaviour
{
    private List<Card> GameDeck = new List<Card>();

    private void Awake()
    {
        GameStart();
    }

    public List<Card> GetPlayerDeck()
    {
        var playerDeck = GameDeck.Where((c, i) => i % 2 != 0).ToList();
        return playerDeck;
    }

    public List<Card> GetCPUDeck()
    {
        var cpuDeck = GameDeck.Where((c, i) => i % 2 == 0).ToList();
        return cpuDeck;
    }

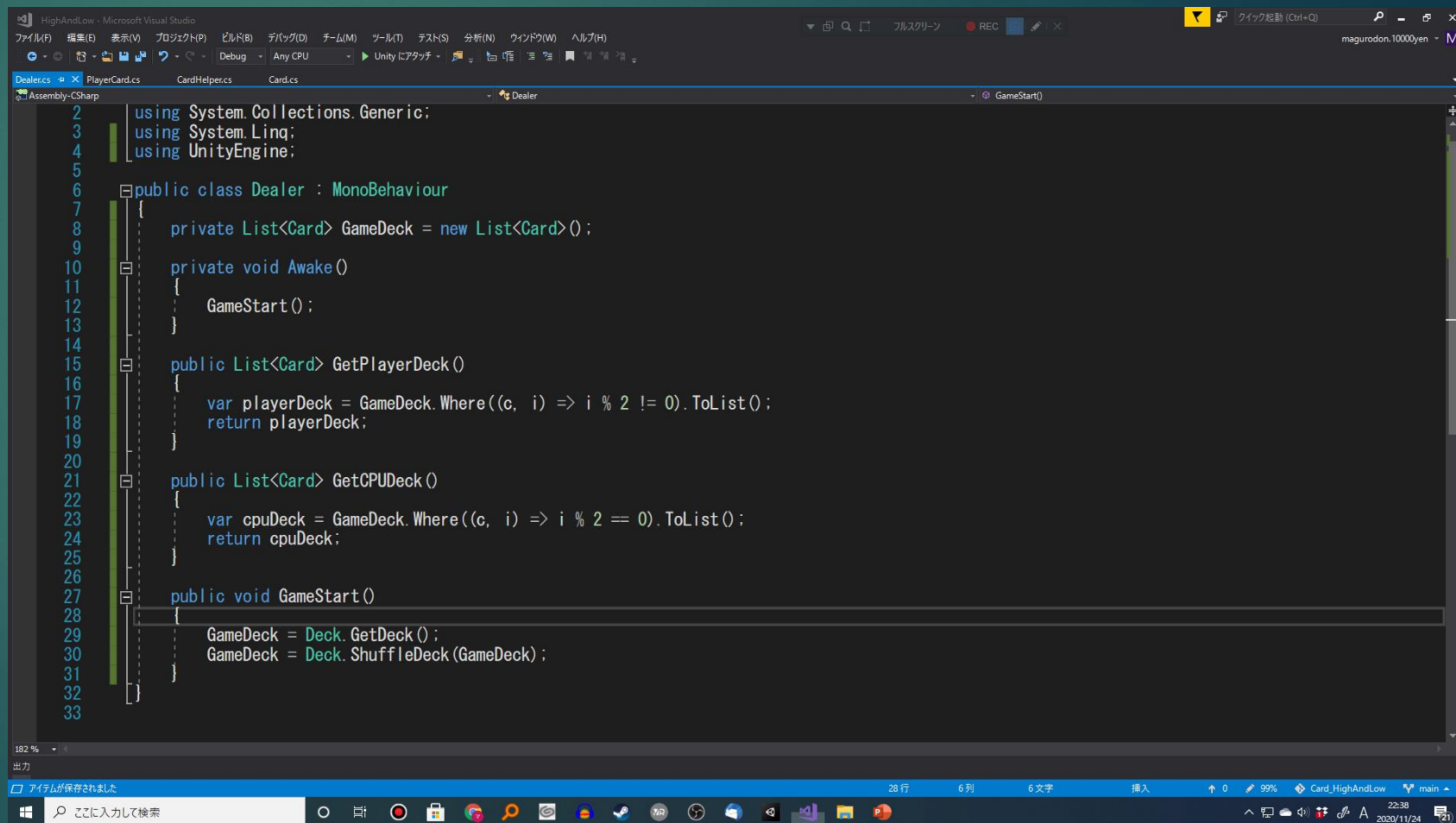
    public void GameStart()
    {
        GameDeck = Deck.GetDeck();
        GameDeck = Deck.ShuffleDeck(GameDeck);
    }
}
```


Unity

13

ハイアンドロー

- では引き続いて、CpuのカードとPlayerのカードを比べてみましょう
- PlayerCardとCPUCardというスクリプトを作成します
- Startの中でDebug.Logを使って振り分けられたデッキの一番上を出力します



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

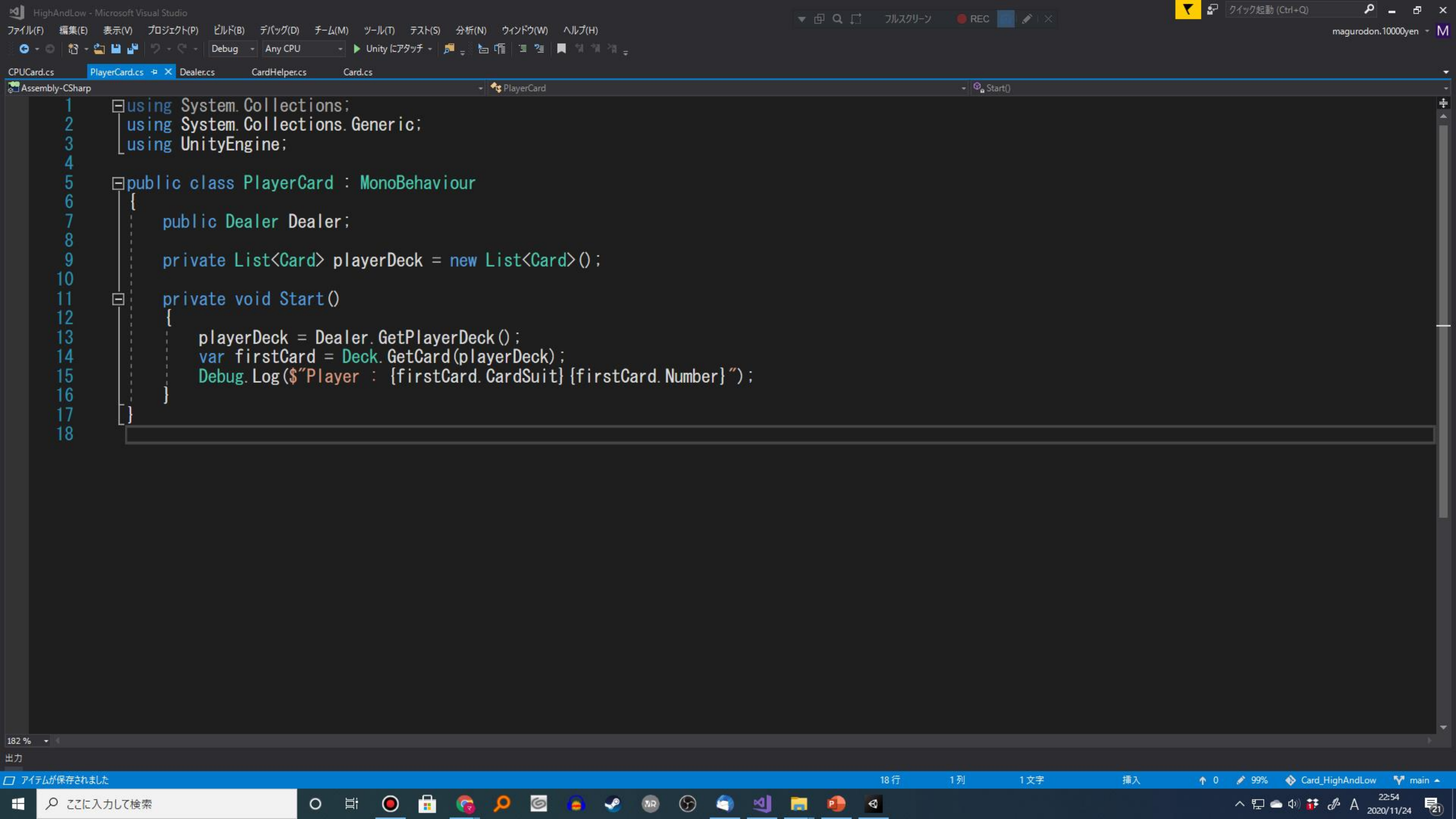
public class Dealer : MonoBehaviour
{
    private List<Card> GameDeck = new List<Card>();

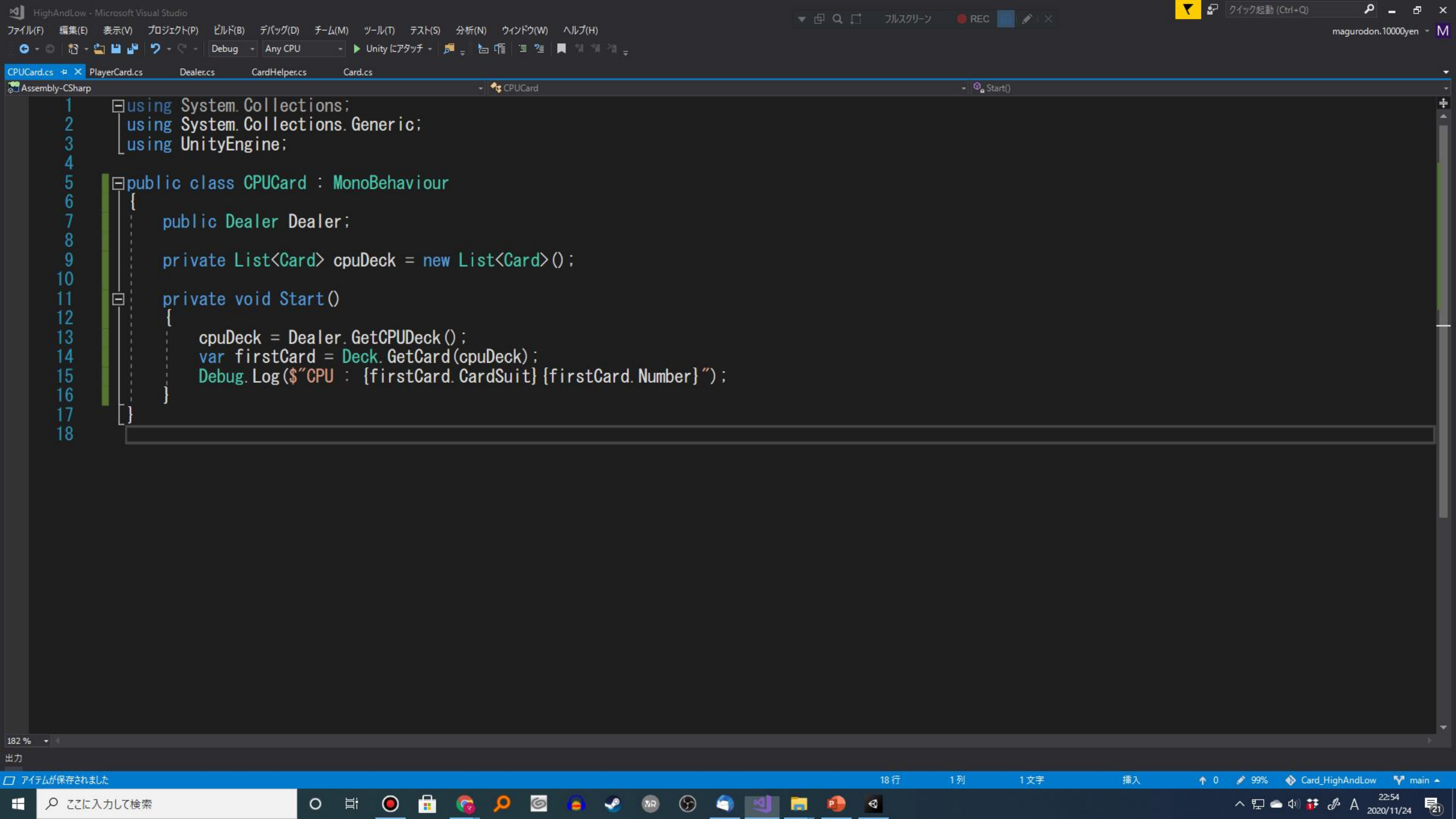
    private void Awake()
    {
        GameStart();
    }

    public List<Card> GetPlayerDeck()
    {
        var playerDeck = GameDeck.Where((c, i) => i % 2 != 0).ToList();
        return playerDeck;
    }

    public List<Card> GetCPUDeck()
    {
        var cpuDeck = GameDeck.Where((c, i) => i % 2 == 0).ToList();
        return cpuDeck;
    }

    public void GameStart()
    {
        GameDeck = Deck.GetDeck();
        GameDeck = Deck.ShuffleDeck(GameDeck);
    }
}
```

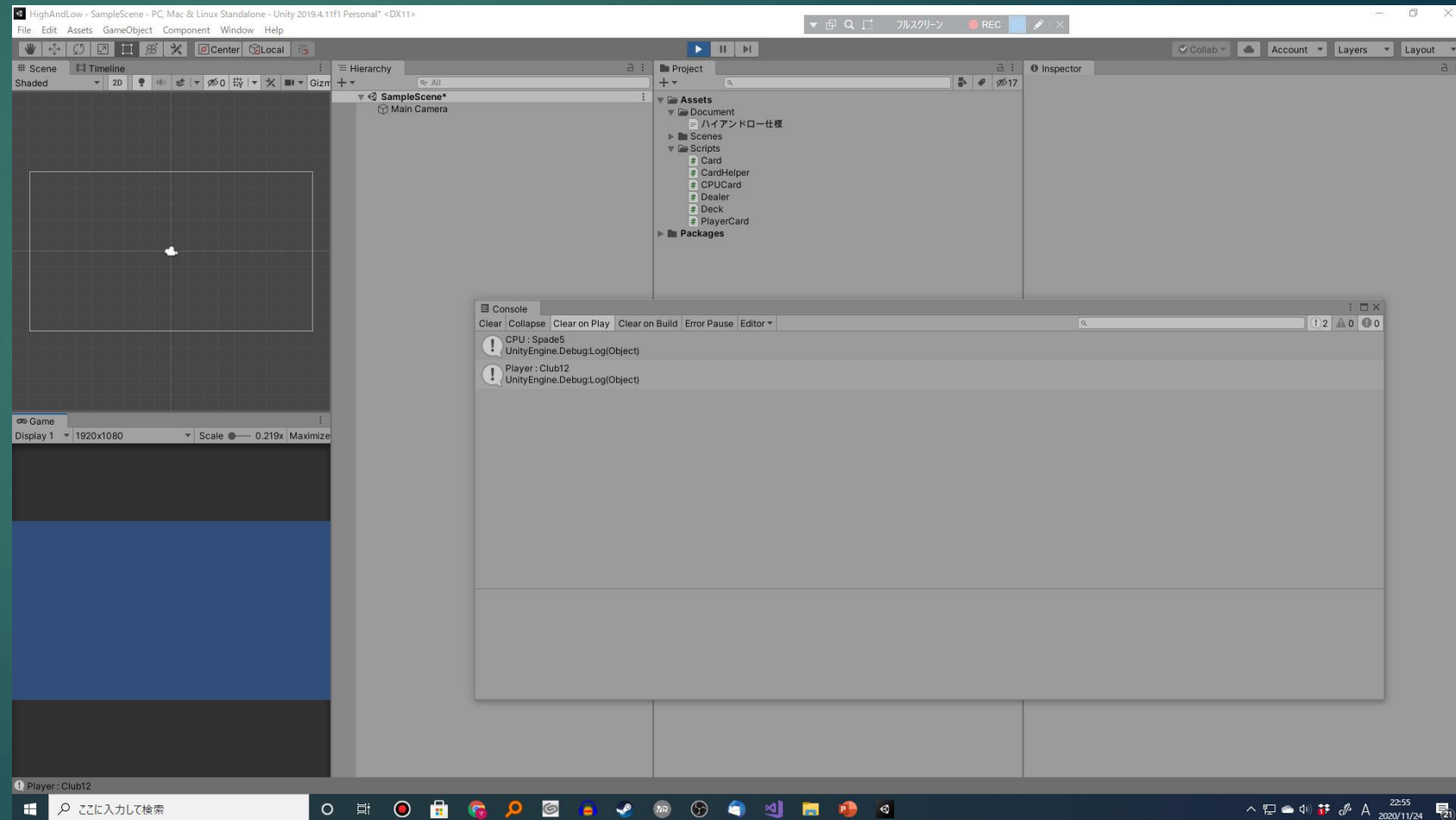


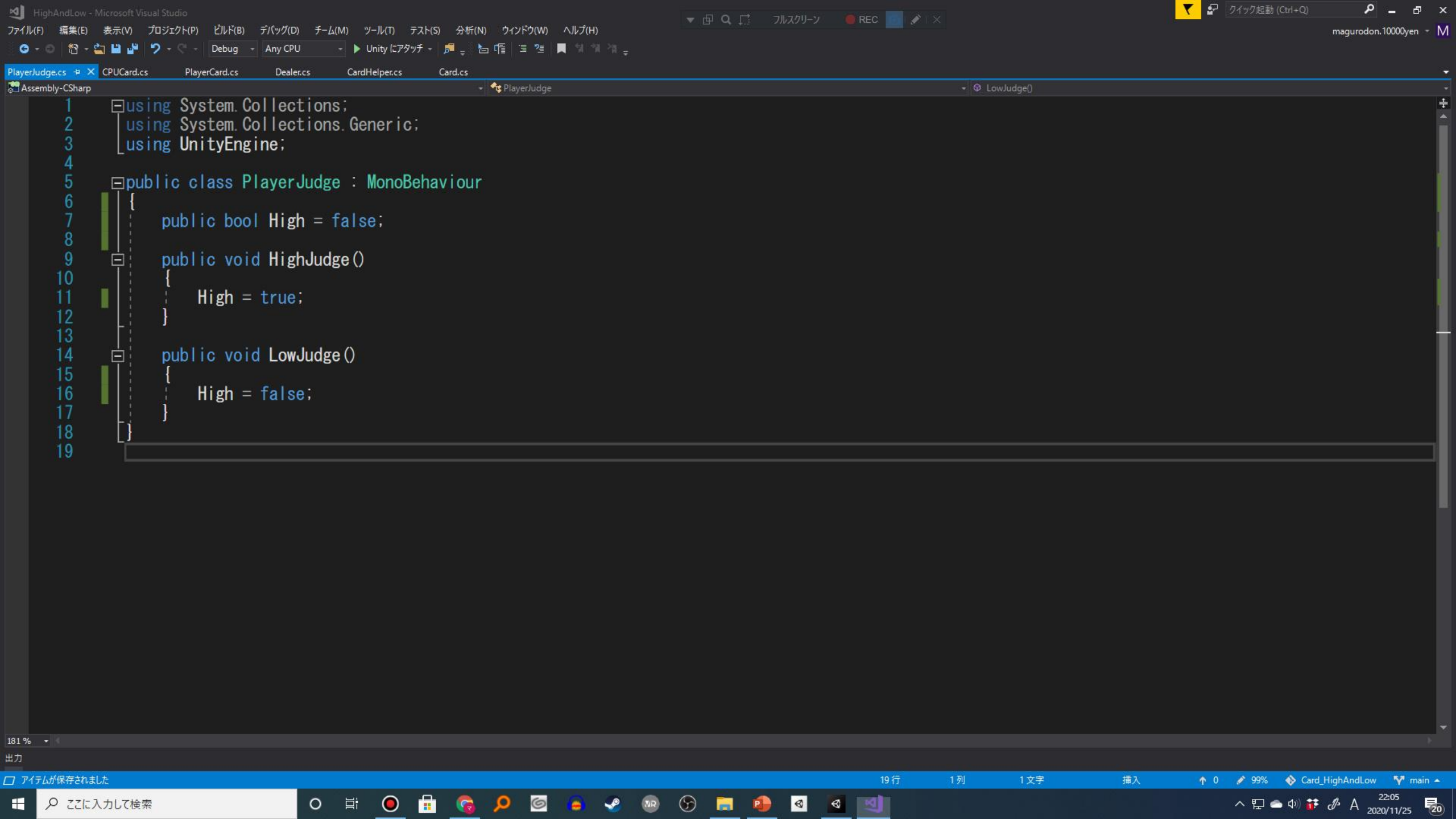
Unity

ハイアンドロー

16

- mainCameraにDealerとCPUCardをアタッチし、noneになっている場所を全て埋めて再生してください
- ConsoleにCPUとPlayerの一番上のカードを表示できたら大丈夫です
- 次は比較をしていきましょう
- Buttonを2つ配置し、二つの値を比べ、ハイかローかを判定しましょう
- PlayerJudge.csを作成します



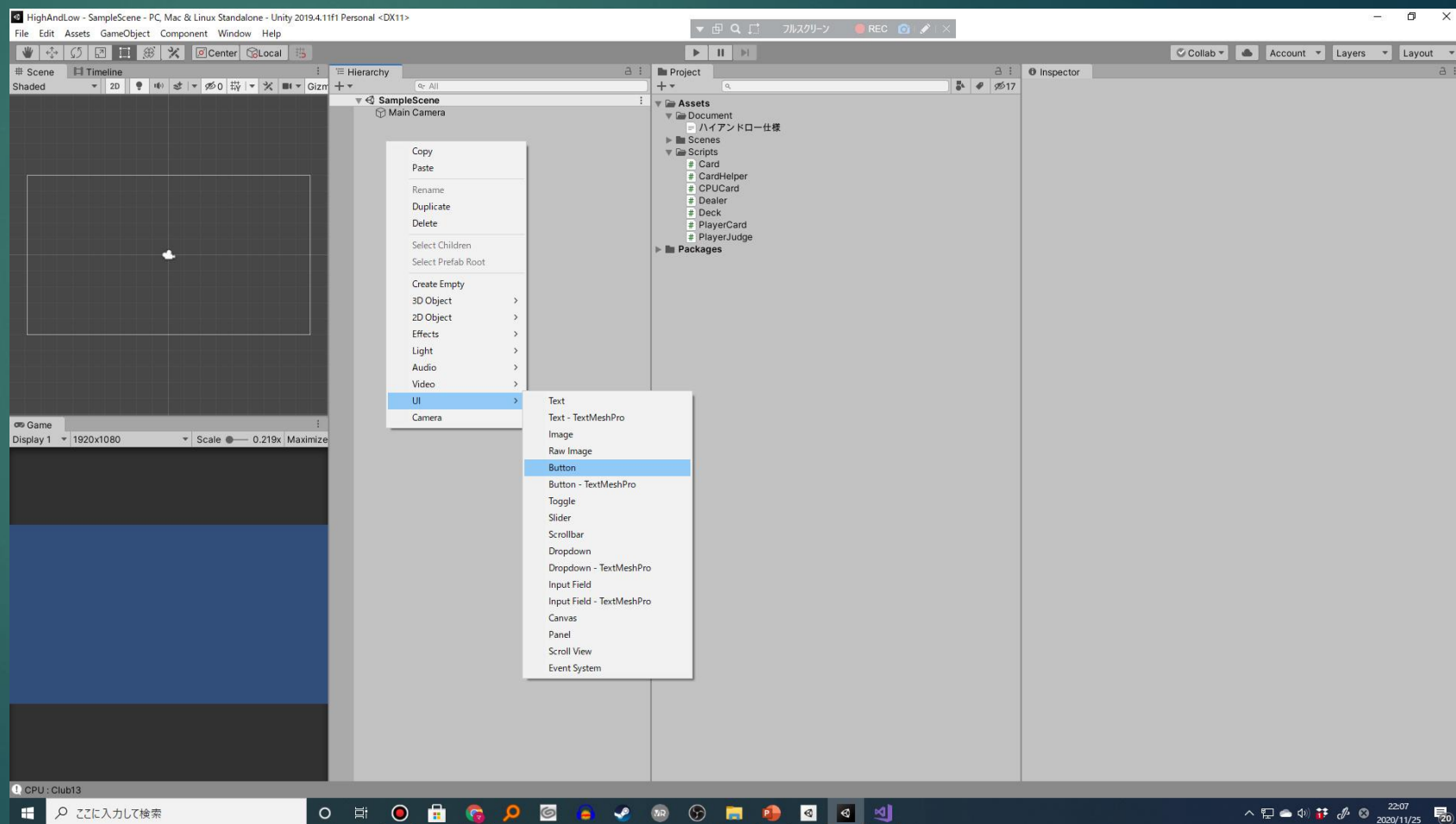


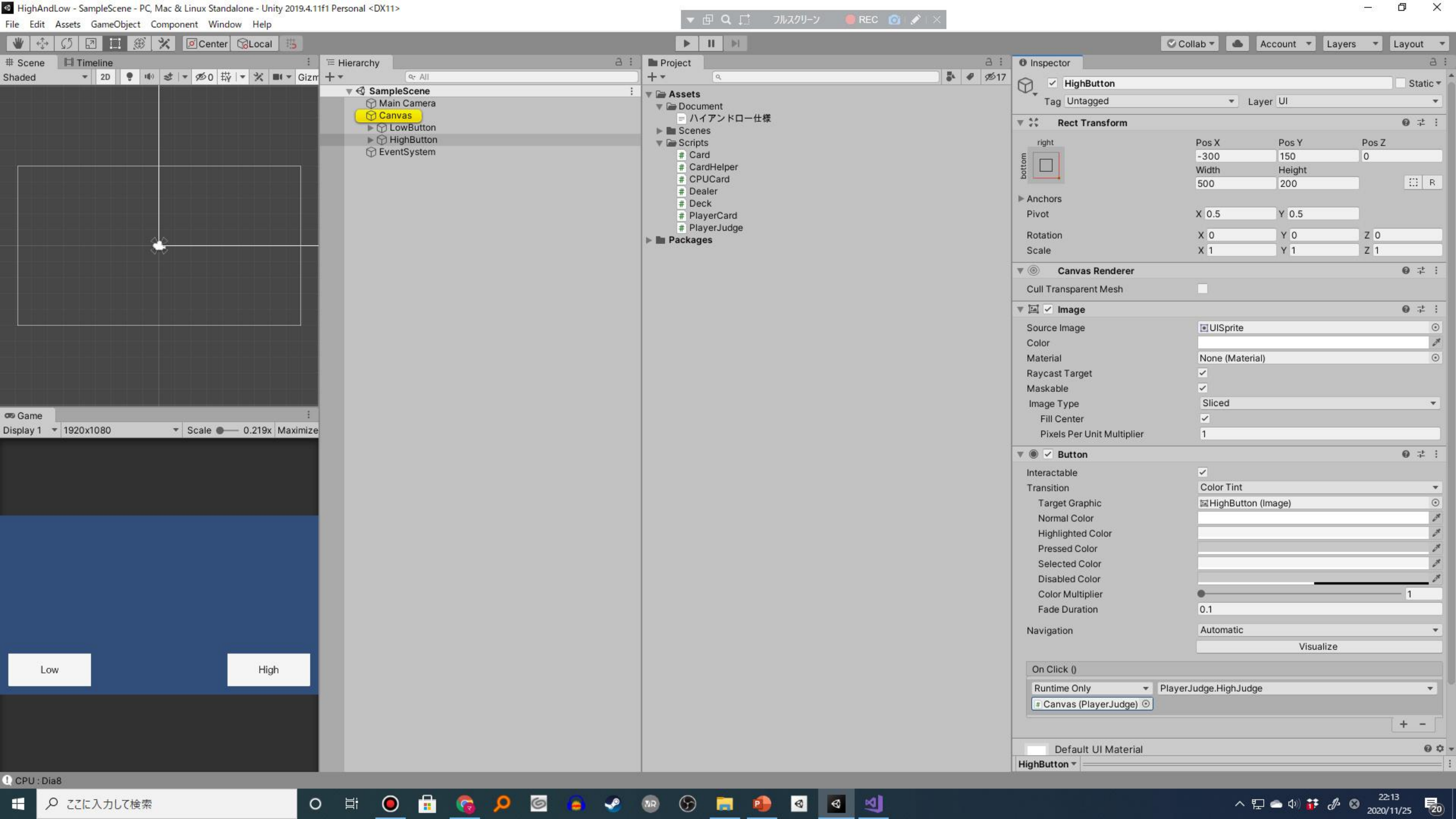
Unity

ハイアンドロー

18

- Buttonを配置してみましょう
- Hierarchy欄で右クリックし、UIからButtonを選択します
- 小さいボタンが作成されると思いますので、大きさを調整し、左側に配置します
- それをCtrl+Dで複製して、右側にも配置
- 中の文字を左側のボタンを"Low"、右側のボタンを"High"としてください
- 最後に、CanvasにPlayerJudgeをアタッチし、ボタンの設定をしてください



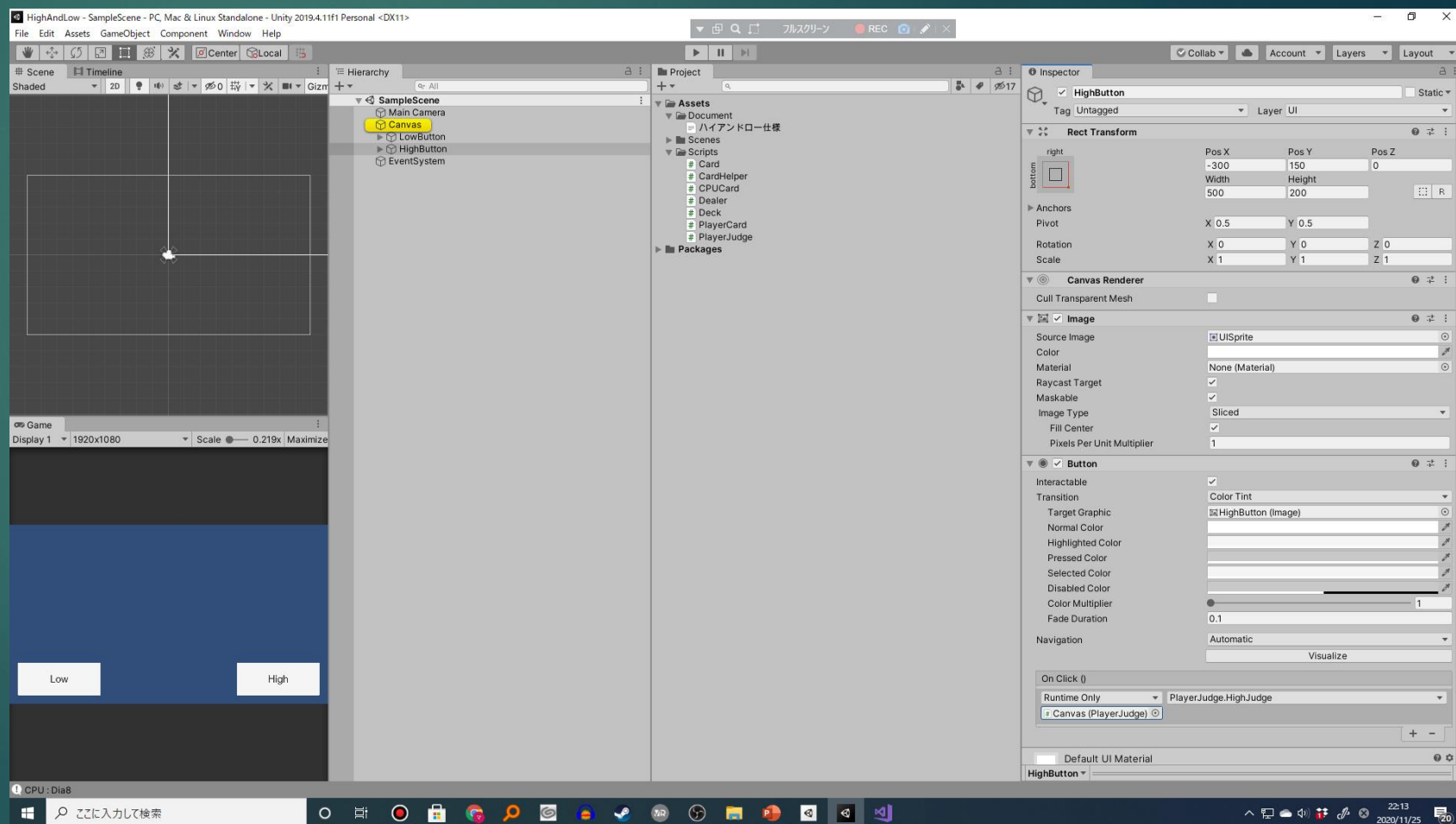


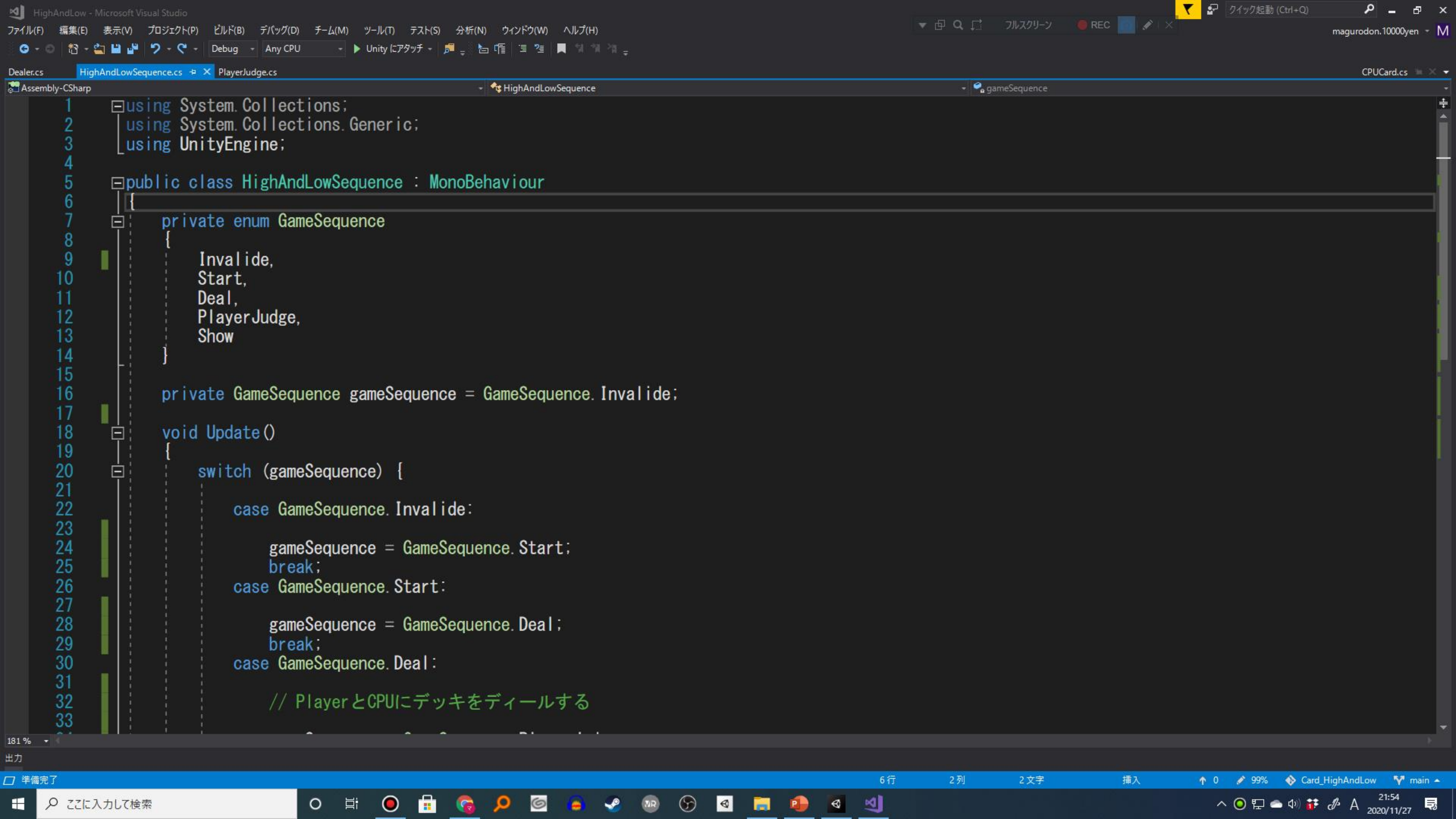
Unity

ハイアンドロー

20

- ボタンの設定が終わりましたらゲームのシーケンスを構築していきます
- HighAndLowSequence.csを作成します
- このスクリプトの役目はゲーム全体の管理をします
- 今回は簡単にenumで状態を遷移させていきましょう



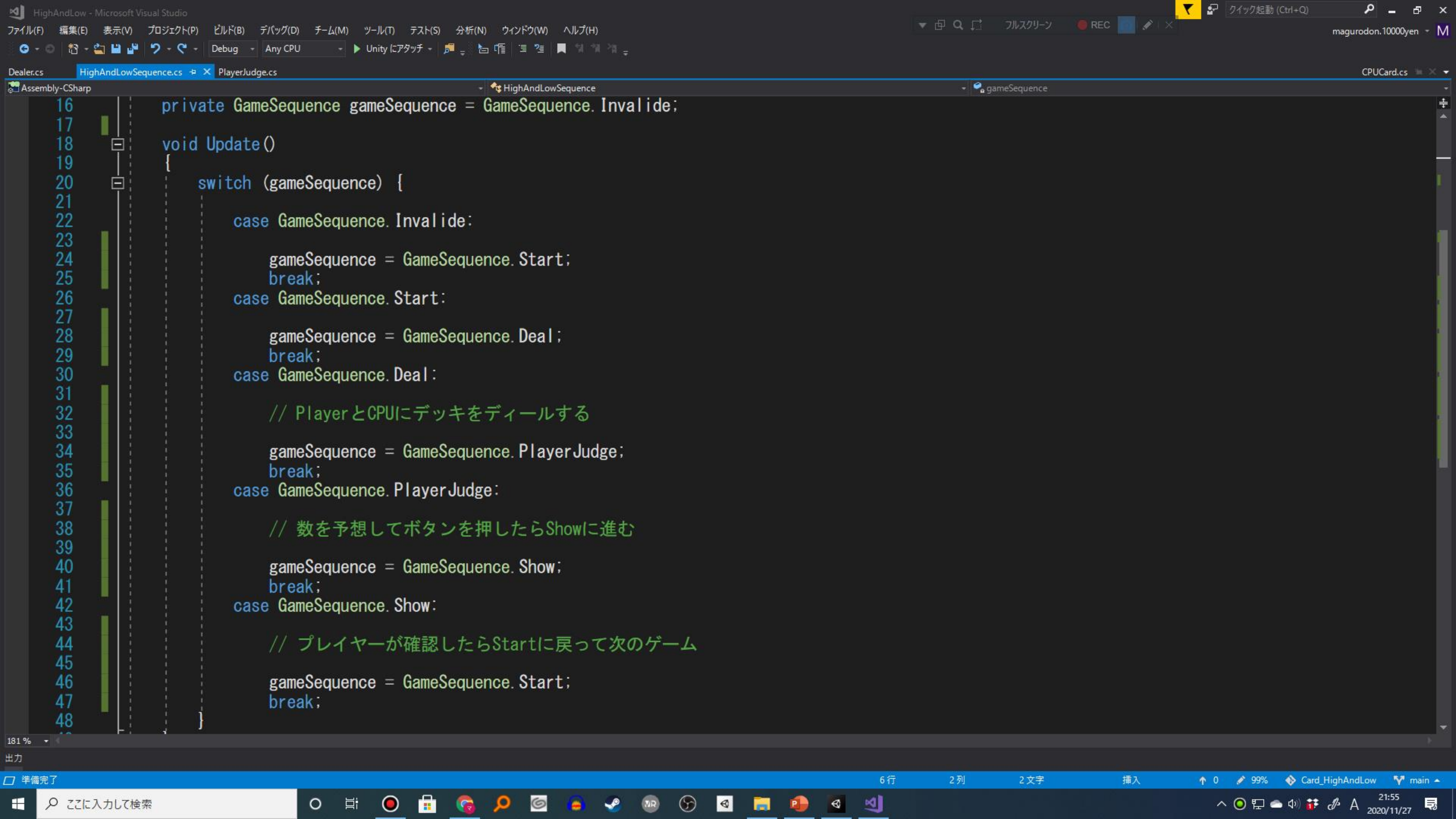


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HighAndLowSequence : MonoBehaviour
{
    private enum GameSequence
    {
        Invalid,
        Start,
        Deal,
        PlayerJudge,
        Show
    }

    private GameSequence gameSequence = GameSequence.Invalid;

    void Update()
    {
        switch (gameSequence) {
            case GameSequence.Invalid:
                gameSequence = GameSequence.Start;
                break;
            case GameSequence.Start:
                gameSequence = GameSequence.Deal;
                break;
            case GameSequence.Deal:
                // PlayerとCPUにデッキをディールする
        }
    }
}
```

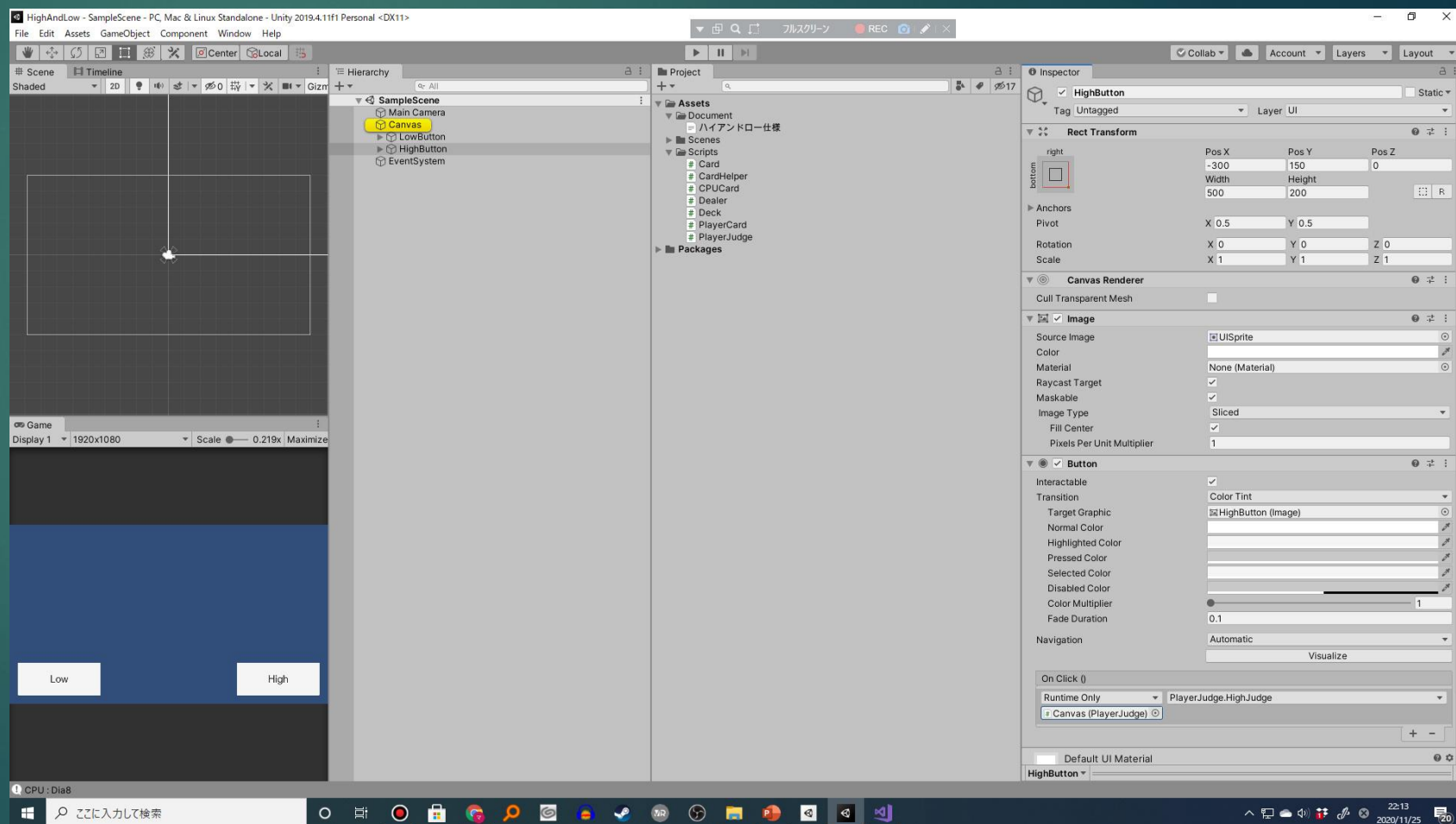


Unity

ハイアンドロー

23

- ここからは実際にどういう書き方をしてゲームを回していくかを実習していきます
- 一緒に書いていきましょう
- 次回に関しては、実際にトランプの描画をしていきます



Tips : Listや配列の利用方法

24

- リストや配列は本当によく使われます
- ひと昔前は
- [1,0,0,0,0]
- [1,1,1,1,0]
- [0,0,0,1,0]
- [0,0,0,1,2]
- これでMap等を表現していました(0が進行不能、2がゴール)
- ちなみに上は多次元配列と呼ばれるものです