Unity で文化祭用ゲームランチャーを作ろう

T.Wattz

2024 年 6 月 5 日

麻布の物理部無線班の部誌に Unity のコードエディターは VS-code みたいな事書かれてて憤慨しました。再び T.wattz です。(Mac ならわからんでもないけど Windows ならどっからどう考えても VisualStudio ですよねぇ (#ˆωˆ)

それはさておき、AtsEX はニッチで複雑怪奇な話題だったのですが、今回は C#プログラミングの定番である Unity を使ってゲームランチャーを作っていきます! (半分筆者の日記みたいなものですが)

リポジトリは github.com/cydiawaltz/GameLuncher2024 にあるのでどうぞ御覧ください。Unity は 2022.3.2f1 が必要です。また無 Asset 主義者なので基本的にアセットは登場しないと思いますのでご安心下さい。

まずはデザインを決めます。斯々然々でそこらへんに落ちてた PS2 ソフト「ナムコミュージアム アーケード Hits!」をパクることに決定しました。(以下画像)

図 1　プライドなんてない。

よし。では早速 Unity を開いてみましょう。そして、なんとなくいい感じにオブジェクトを配置します。

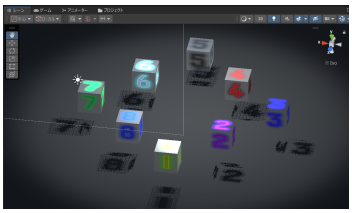

図 2　円状にオブジェクトを配置。

しました。では早速カメラを設置し、それを回すプログラムを記述しましょう。

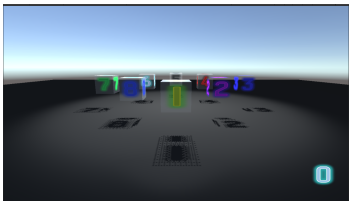

図 3　カメラを設置。

リスト 1　MoveNext.cs

```
\\using ディレクティブ、クラス名とかは省略。
    public float speed;//移動する速度
    public int allObjects;//置いているゲームの総数
    public GameObject center;//真ん中(今回は原点)にあるGameObject
    public int index;
    public int moveAngle;//回転角
    float moveSeconds;//回転する秒数(内部演算用)
    public float currentAngle;//現在の角度
    void Start()
    {
        index = 1025;
        moveAngle = 360/allObjects;//回転角の大きさを設定
        moveSeconds = moveAngle/speed;//移動時間を設定
        currentAngle = 0f;//位置を初期化
    }
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.RightArrow))
        {
            StartCoroutine(Move(-speed));
            index++;
            currentAngle = currentAngle+moveAngle;
        }
```

```
24          if(Input.GetKeyDown(KeyCode.LeftArrow))
25          {
26              StartCoroutine(Move(speed));
27
28              if(index == 0){SceneManager.LoadScene("HideScene");}//イースターエッグ
29              else{index--;}
30              currentAngle = currentAngle-moveAngle;
31          }
32      }
33      public IEnumerator Move(float speed)
34      {
35          float timer = 0f;
36          while (timer < moveSeconds)
37          {
38              transform.RotateAround(center.transform.position,new Vector3(0,1,0), speed);//これを唱えると第一引数 (Vector3 型)の
                    位置を中心に回転します。
39              //center.transform.position は普通に Vector3.zero でいいです。
40              timer ++;
41              yield return null;
42          }
43      }
```

（うわっ長っ、先が思いやられるなコレ）
このスクリプトを回転する物体（MainCamera）にアタッチし、インスペクタに諸々の情報を入力しましょう。

図 4　見にくっ。

これでオブジェクトを回転できました。へい。見づらいので紙版でご覧の方は pdf 版で見るほうがいいかもしれません。まぁ、大したことやってる写真でもないのでシカトしてもらっても構いません。
次は UI にしましょうかねぇ。ではまたクソ長コードを御覧ください。

リスト 2　UIDraw.cs

```
1   //再びusing とクラスの記述は省略
2   public MoveNext moveNext;
3   public int index;
4   public bool isDisplay;//表示・非表示
5   public GameObject imageObject;
6   public Image imageComponent;
7   public Sprite[] spriteImage;
8   public GameObject buttonObject;
9   public Image buttonComponent;
10  public Sprite[] buttonImage;
11  public GameObject info;
12  void Start ()
13  {
14      imageComponent = imageObject.
            GetComponent<Image>();
15      buttonComponent = buttonObject
            .GetComponent<Image>();
16      isDisplay = true;
17  }
18  void Update ()
19  {
20      index = moveNext.index;
21      if(isDisplay == true)
22      {
23          imageComponent.sprite =
                spriteImage[(index%(
                moveNext.allObjects
                ))];//わりかし脳筋なの
                でspriteImage[0]には
                allObjects のをいれる
24          buttonComponent.sprite =
                buttonImage[0];
25          info.SetActive(true);
26      }
27      if(isDisplay == false)
28      {
29          buttonComponent.sprite =
                buttonImage[1];
30          info.SetActive(false);
31      }
32  }
33  public void OnClick ()
34  {
35      isDisplay = !isDisplay;
36  }
```

表示/非表示とスプライトの表示をまとめてみました。
同一オブジェクトのテクスチャ画像を index の値に応じて変更する感じの処理です。
また、最後のメソッド Onclick() は表示/非表示を反転させるやつです。表示/非表示を入れ替えるボタンの OnClick() で呼び出します。

完成形はこんな感じになりま〜す。

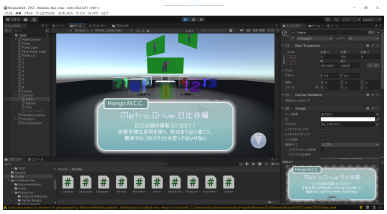

図 5　回転すると変わります。

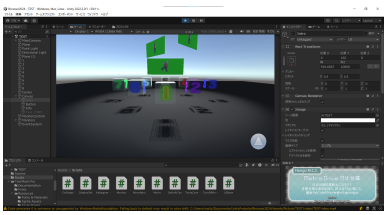

図 6　非表示にもできます。

次に解像度に応じて諸々を調整します。いらない？ウインドウの大きさいじれるし、多少はね？

リスト 3　DisplaySet.cs

```
1   public float bairitu;//倍率に応じて
        画像の大きさを変更
2   public bool isSetWidth;//縦に合わせ
        る（インスペクターで選んでね^^e2
        ^^99^^a1）
3   void Update()
4   {
5       int width = Screen.width;
6       int height = Screen.height;
7       RectTransform rectTransform =
            GetComponent<
```

```
            RectTransform>();//対象の
            オブジェクトにアタッチ
 8      float screenRatio = (float)
            width / height;
 9      if (isSetWidth)
10      {
11          rectTransform.
                SetSizeWithCurrentAnchors
                (RectTransform.Axis.
                Horizontal, width /
                bairitu);
12      }
13      else
14      {
15          rectTransform.
                SetSizeWithCurrentAnchors
                (RectTransform.Axis.
                Vertical, height /
                bairitu);
16      }
17  }
```

うん。いい感じ。isSetWidth を true にしておくと横幅に、false にすると縦幅に合うよう調整します。bairitu の値は現物合わせで調整しました。(解像度が 1920*1080 で isSetWidth を true にしてるとき、bairitu が 2 だと 1920/2=960 で横幅は 960px に調整されます。)
次に位置。このままだと解像度が大きいほど下に下がってしまって遺憾なのでそれもスクリプトでゴリ押し。
~~今思ったけど普通にピポットを下部に調整すればよかっただけジャマイカ…？~~

リスト 4　SetInfoTex.cs

```
 1  RectTransform rectTransform;
 2  public int heightBairitu;
 3  void Start()
 4  {
 5      rectTransform = this.
            GetComponent<
            RectTransform>();
 6  }
 7  void Update()
 8  {
 9      int height = Screen.height;
10      float posHeight = height /
            heightBairitu;
11      rectTransform.anchoredPosition
            = new Vector2(0,
            posHeight);
12  }
```

あんまり綺麗なコードじゃないですね... 精進します。先ほどと同じように倍率の値を設定すると解像度に関わらずいい感じの見た目になります。
最後を
new Vector2(this.transform.position.x,posHeight)
とすると x の値が毎フレームごとにどんどん増えていくんですよねぇ... なんでだろ。
UI も実装したので、メインディッシュであるアプリ起動の実装に移ります。

リスト 5　CallApps.cs

```
 1  using UnityEngine;
 2  using System.Diagnostics;
 3  using System.IO;
 4  public class CallApps :
        MonoBehaviour{
 5      public MoveNext moveNext;
 6      public int index;
 7      public string exepath;
 8      public string readmePath;
 9      string type;
10      public bool isHTML;
11      public bool isNeedQuit;
12      public int currentNum;
13      private void Update()
14      {
15          index = moveNext.index%
                moveNext.allObjects;
16          if(isHTML){ type = "html
                ";}
17          else { type = "exe"; }
18          exepath = Path.Combine(
                Application.dataPath,
                 "../Apps/GAME" +
                index + "/main." +
                type);
19          readmePath = Path.Combine(
                Application.dataPath,
                 "../Apps/GAME" +
                index + "/readme.txt
                ");
20          if (Input.GetKeyDown(
                KeyCode.Z)&&index ==
                currentNum)
21          {
22            LaunchApp(isHTML,
                  isNeedQuit);
23          }
24          if(Input.GetKeyDown(
                KeyCode.X)&&index ==
                currentNum)
25          {
26            LaunchReadMe();
27          }
28      }
29      public void LaunchApp(bool
            isHTML,bool isNeedQuit)
30      {
31          Process.Start(exepath);
32          if(isNeedQuit == true)
33          {
34            Application.Quit();
35          }
36      }
37      public void LaunchReadMe()
38      {
39        Process.Start(readmePath);
40      }
41  }
```

まあ簡単な実装ですが。一番の見どころは System.Diagnostics.Process.Start(string fileName) でしょうか。
これを呼び出すとファイルを開くことができます。便利。ちなみに ProcessStartInfo ってのを叩くと他にも引数を指定したりいろいろできるのですが、こちらについては今回あまり関係ないので割愛。(「MetroDrive 日比谷編」付属のコンポーネントインストーラも引数付きで起動してるのでサイレントインストールが使えてます。)
インスペクタから isNeedQuit（負荷の大きいゲームとかフルスクリーンのゲーム等で true)、isHTML（HTML ゲームで true）とかを設定します。起動は伝統に従って Z でアプリ起動としました。半ページ残ってるのでプレイ画面の軽量化でもやりましょうか。

リスト 6　Monitor.cs

```
1   public GameObject[] monitors;//monitors[0]には何も淹れない
2   public GameObject offMonitorsL;
3   public GameObject offMonitorsR;
4   public MoveNext moveNext;//MoveNext.cs
5   public GameObject[] beginOffMonitor;//めんどくさいので最初に描画しないモニターは手動で設定したれ
6   public int canSee;//表示されるモニターの値
7   void Start()
8   {
9       monitors[(moveNext.index%moveNext.allObjects)].SetActive(true);
10      canSee = (90 / moveNext.moveAngle);//横 90° +1で開始時に白飛びするのを防止
11      foreach(GameObject beginOff in beginOffMonitor)
12      {
13          beginOff.SetActive(false);
14      }
15      foreach (GameObject monitor in monitors)
16      {
17          monitor.GetComponent<VideoPlayer>().Pause();//全部開始時には一時停止
18      }
19      monitors[moveNext.index % moveNext.allObjects].GetComponent<VideoPlayer>().Play();
20  }
21  void Update()
22  {
23      if(Input.GetKeyDown(KeyCode.RightArrow))
24      {
25          monitors[((moveNext.index-canSee-1)%moveNext.allObjects)].SetActive(false);
26          monitors[((moveNext.index+canSee)%moveNext.allObjects)].SetActive(true);
27          foreach(GameObject monitor in monitors)//一旦全部オフにする
28          {
29              monitor.GetComponent<VideoPlayer>().Pause();
30          }
31          monitors[moveNext.index % moveNext.allObjects].GetComponent<VideoPlayer>().Play();//今表示されてるモニターだけOn にする
32      }
33      if (Input.GetKeyDown(KeyCode.LeftArrow))
34      {
35          monitors[((moveNext.index+canSee+1)%moveNext.allObjects)].SetActive(false);
36          monitors[((moveNext.index-canSee)%moveNext.allObjects)].SetActive(true);
37          foreach (GameObject monitor in monitors)
38          {
39              monitor.GetComponent<VideoPlayer>().Pause();
40          }
41          monitors[moveNext.index % moveNext.allObjects].GetComponent<VideoPlayer>().Play();
42      }
43  }
```

これで見えてるオブジェクトだけが SetActive(true) されるようになりました。満足。
他にもテクスチャ圧縮とかコライダーの削除とかで軽量化をいろいろ頑張りました。偉い。
あとはゲームです。さぁ、これが読まれているときにゲームはできているのでしょうか....

図 7　とりあえずテストは成功。