Udonと自動ドアと図書館と

UDEC 2022

水無月せきな

自己紹介

水無月せきな

2020.7 末～

アバター改変
ワールド製作
ツール開発

Before BigBang

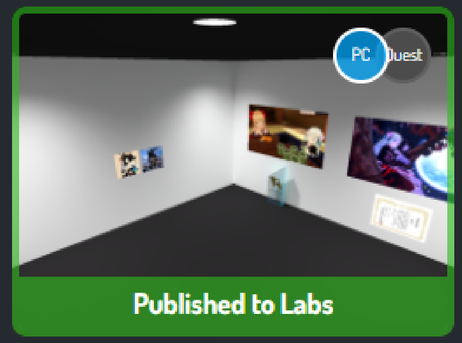
Users In-World 0
Favorites 7
Last Updated Jun 01 2022

舞鶴仮想城址公園 -Park of Maiduru Castle-

Users In-World 0
Favorites 7
Last Updated May 28 2022

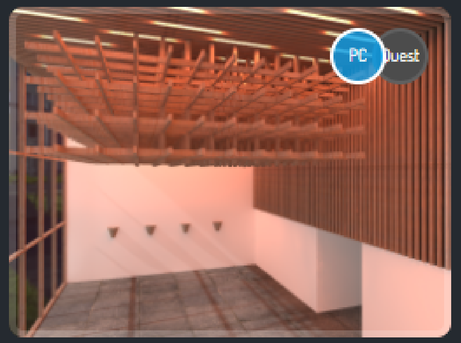
水無月図書館

Users In-World 0
Favorites 734
Last Updated May 28 2022



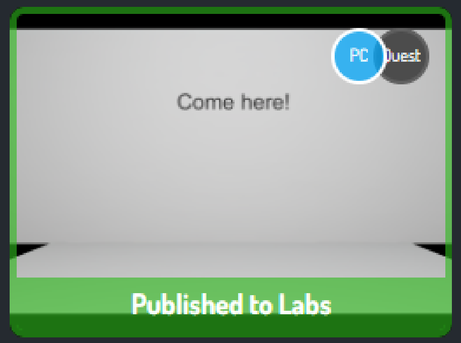
せきなの写真館

Users In-World 0
Favorites 1
Last Updated Nov 30 2021



バーチャル総合研究博物館 - CRI & Museum

Users In-World 0
Favorites 102
Last Updated Dec 14 2021



Gimmick Sample World

Users In-World 0
Favorites 1
Last Updated Nov 14 2021

葵屋敷 -Aoi Yasiki-

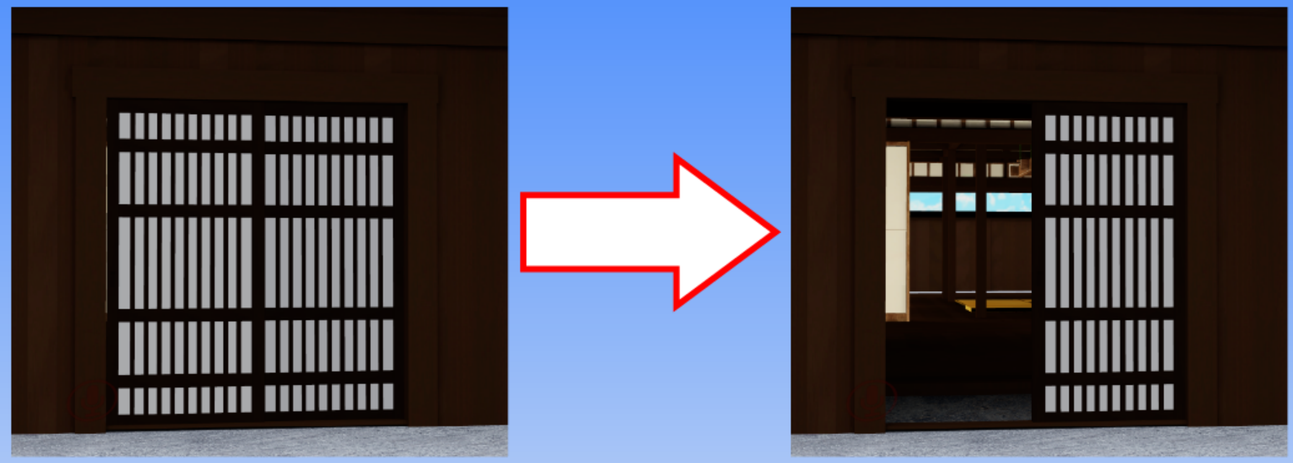
Users In-World 0
Favorites 1
Last Updated Jun 26 2022

せきなホーム -Sekina's Home-

Users In-World 0
Favorites 7
Last Updated May 29 2022

自動ドア

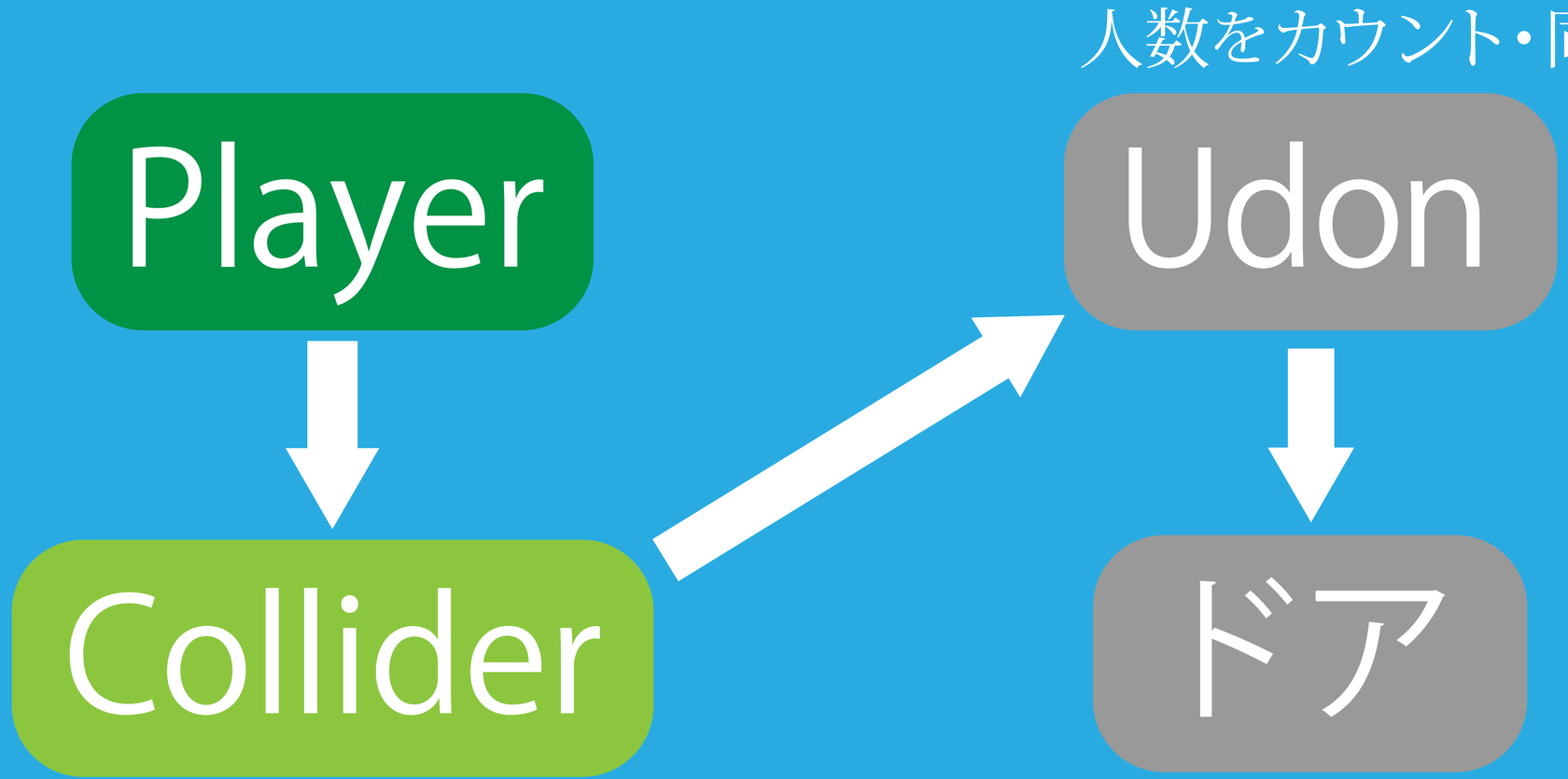
自動でドアが開閉!!



VRChat SDK3 World ギミック

※別途UdonSharp(無償)が必要です。
※ドアの３Ｄモデルは付属しません。

原理



問題①

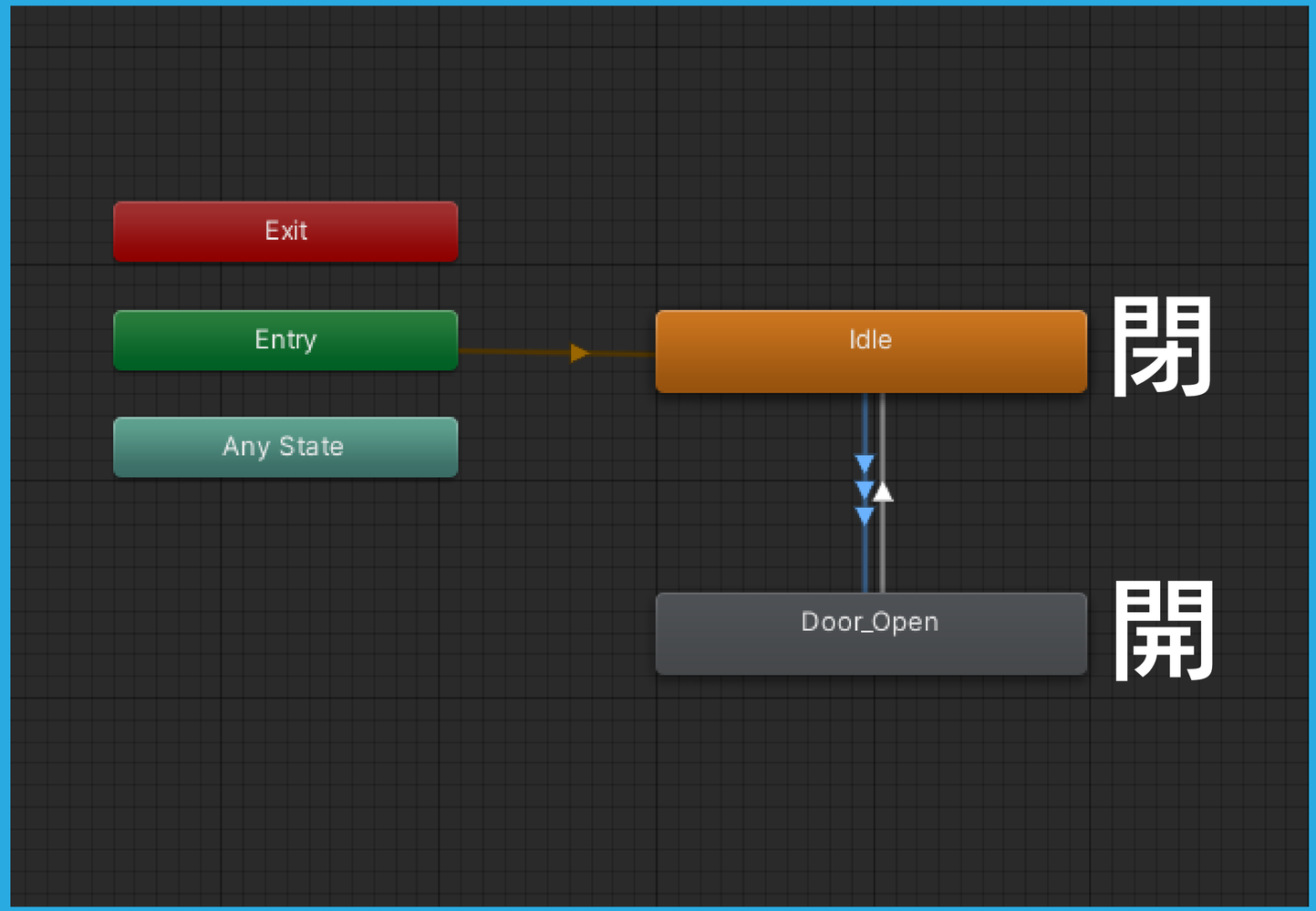
Collider内でPlayerがワールドから離れた場合、
OnPlayerTriggerExitが呼ばれない。

OnPlayerLeftでもカウントを減らす。
ただし、Collider内に居るかどうかチェック。

問題②

Updateで処理せずに、
ドアの開閉を自然な感じにしたい。

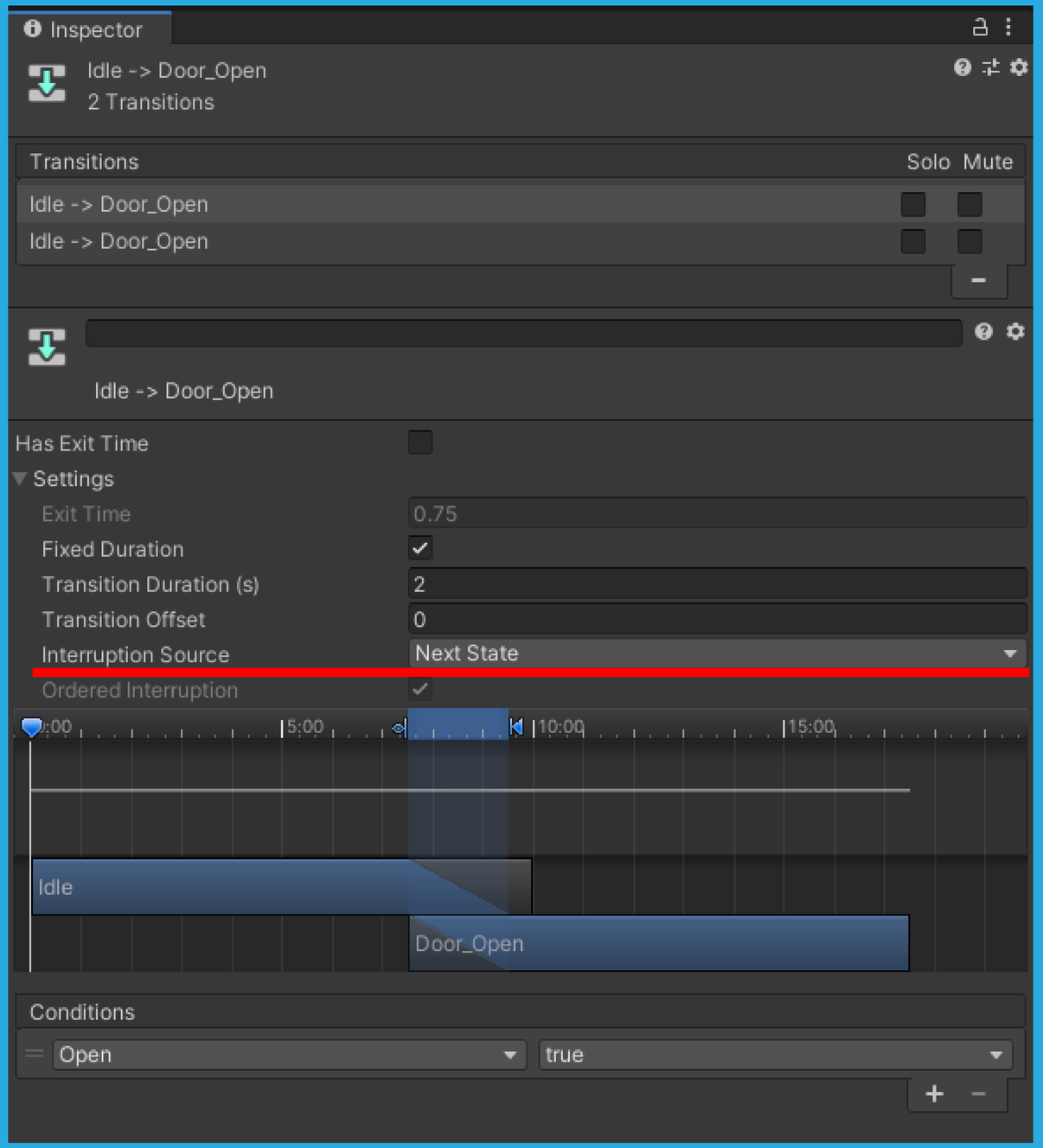
実装



Animationは状態のみ。モーションはTransitionで。

実装

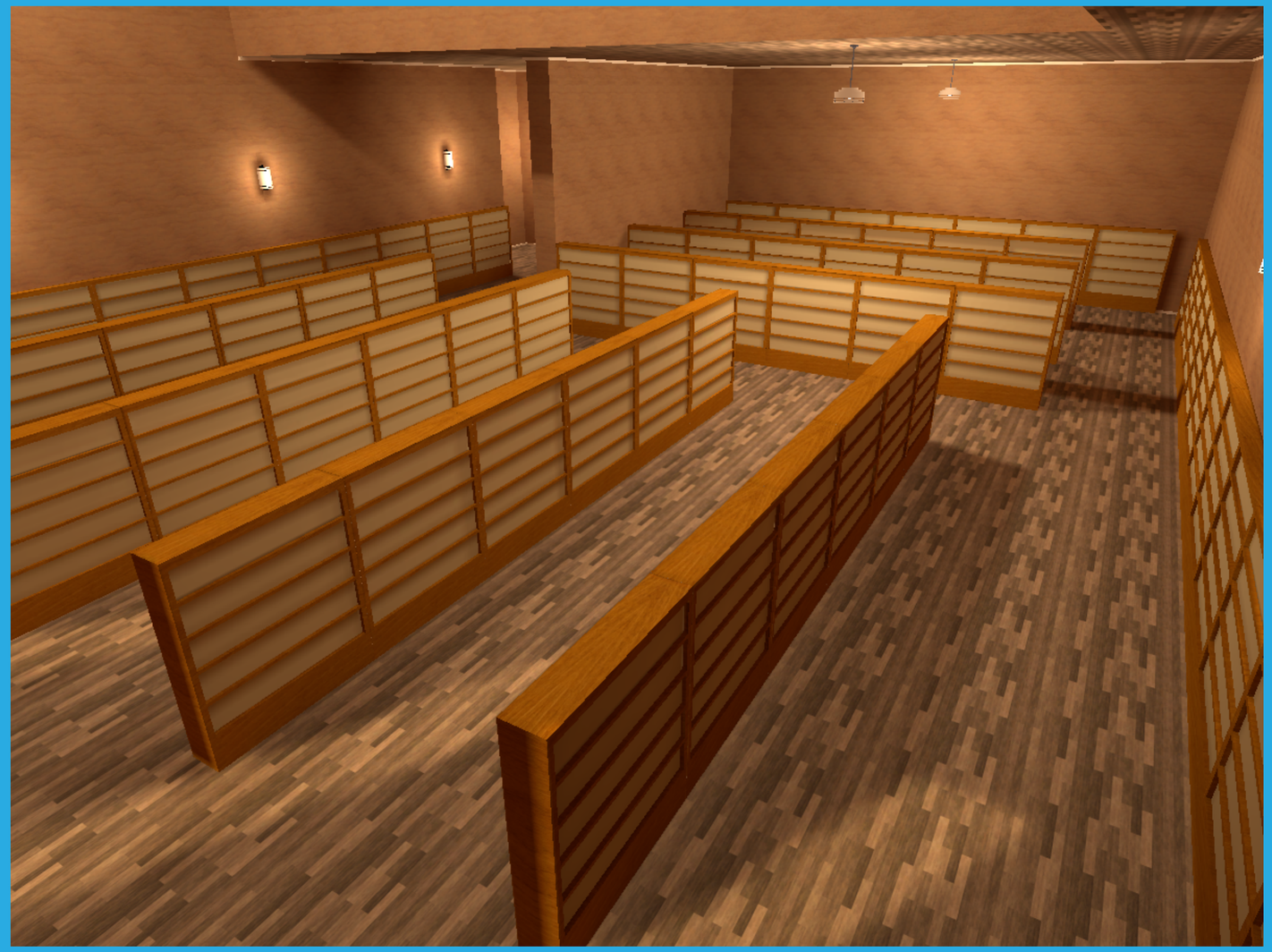
Interruption Sourceで
遷移を割り込ませる。



問題③

後から入ると、
律儀にTransitionを再生する

水無月図書館



著作権切れ＋拙作少々。
16,716冊くらい。

特徴

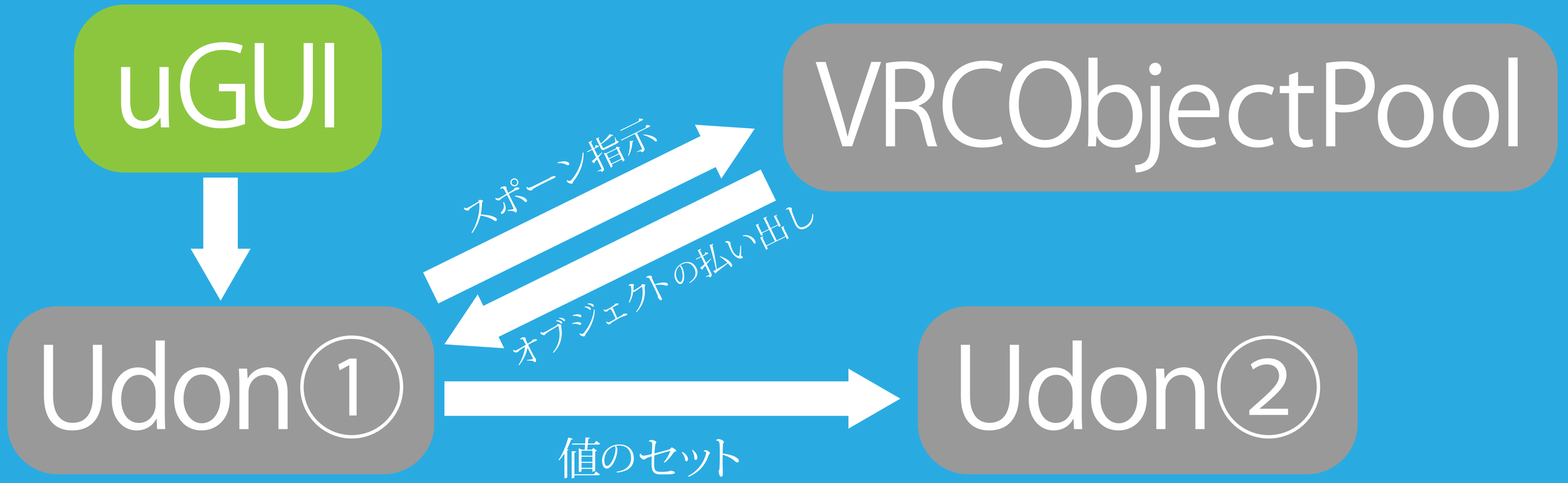
書店形式

⇒陳列された中から手に取る

構成

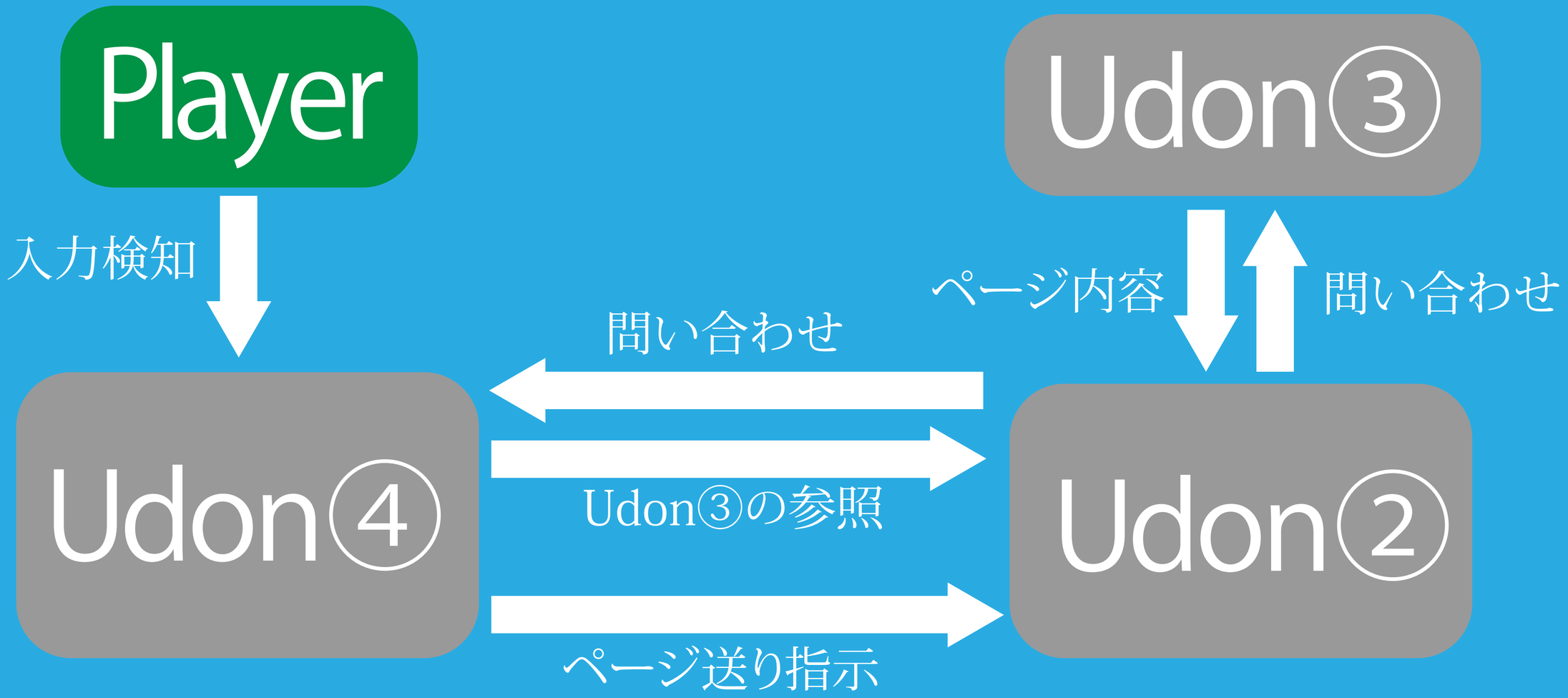
Udon①：本のスポーン位置を保持。NoVariableSync。
Udon②：本のオブジェクトに付随。番号とページ数保持。
Udon③：一定数の本の内容を多次元配列で保持。非同期。
Udon④：一定数の③を配列で保持。入力も監視。非同期。
Udon⑤：本の返却・ワールドUIの処理を担当。非同期。
VRCObjectPool：本のオブジェクトをプーリング。

全体図①

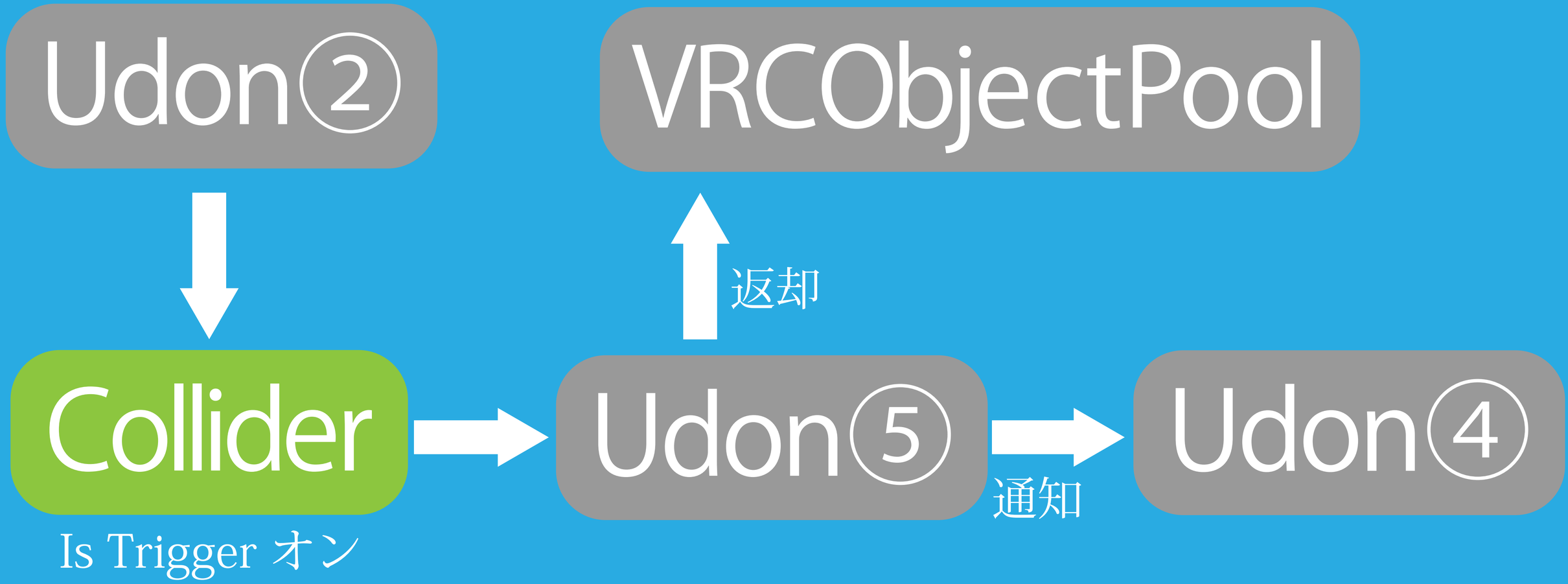


本ナンバー・ページ数：同期

全体図②


Player
入力検知
Udon③
ページ内容
問い合わせ
問い合わせ
Udon④
Udon③の参照
Udon②
ページ送り指示

全体図③


Udon②
VRCObjectPool
返却
Collider
Udon⑤
Udon④
通知
Is Trigger オン

なぜこんなにUdonが？

スイッチ式の模索 ⇒ 約1.7万個のUdon(Manual)
⇒ 入れないワールドの完成
（Noneならギリ行けたけど……）

1つのUdonに保持 ⇒ Editorの動作が激重に

Try! But I don’t know you can do it.

VRCObjectPool.TryToSpawn()

オブジェクトオーナーしかオブジェクトを取得できない。
(https://tsubokulab.fanbox.cc/posts/2672128)

TrySetVariableValue()

(https://qiita.com/hatsuca_vr/items/1c601de9c541fd09290e)

多次元配列が保存できなかった記憶……

Try系は失敗することがある。しょうがない。Tryだもの。

大量のButtonとCanvas

すべてを常時表示するとFPSが30くらい。

⇒ 非表示でスタート。近くに来た時のみ表示。

⇒ 初回はGameObject丸ごと、
２回目～はCanvasを有効/無効
(https://kazupon.org/unity-ui-optimize-tips/#Canvas-2)

膨らむデータ

TextMeshProおよびリッチテキストタグを使用。

⇒ ルビの関係もあり、容量が膨大に。

⇒ 泣く泣く縦書きを断念。それでも約3倍程度。

……圧縮できないかな？

データ圧縮

Unity Editor上 ⇒ C#のDeflateが使用可能。
VRChat(Udon) ⇒ C#のDeflateは使用不可。

DeflateのアルゴリズムはRFC1951として公開済み。
⇒ Udonでデコーダを書ければ解凍できるはず。
（できるとは言ってない。）

Variable byte Encoding

できそうなやつから。

```
List<byte> temp = new List<byte>();
while (number >= 128)
{
    byte b = (byte)(number & 127);
    temp.Add(b);
    number >>= 7;
}
number |= (1 << 7);
temp.Add((byte)number);
return temp.ToArray();
```

```
int count  = 0;
int readId = 0;
int length = 0;
while ((bytes[readId] & (1 << 7)) == 0)
{
    length |= bytes[readId] << (7 * count);
    count++;
    readId++;
}
byte b = bytes[readId];
b -= 128;
length |= b << (7 * count);
return length;
```

ハフマン符号化

(https://algoful.com/Archive/Algorithm/HaffmanEncoding)

圧縮はEditor上のみで確認。
ハフマン木は別のバイト配列として保持して解凍できた。

```
Queue<HuffmanNode> huffmanNodes = new Queue<HuffmanNode>();
huffmanNodes.Enqueue(huffmanNode);

List<byte[]> temp = new List<byte[]>();
List<byte> tempArr = new List<byte>();
int index = 0;

while (huffmanNodes.Count > 0)
{
    HuffmanNode node = huffmanNodes.Dequeue();
    if (node.IsLeaf)
    {
        tempArr.Add((byte)1);
        tempArr.AddRange(GetTinyBytes(BitConverter.GetBytes(node.Value)));
        //tempArr[0] |= 1 << 7;
    }
    else
    {
        tempArr.Add((byte)0);
        tempArr.AddRange(GetTinyBytes(BitConverter.GetBytes(huffmanNodes.Count)));
    }
    temp.Add(tempArr.ToArray());
    tempArr.Clear();

    if (node.Left != null)
    {
        huffmanNodes.Enqueue(node.Left);
    }
    if (node.Right != null)
    {
        huffmanNodes.Enqueue(node.Right);
    }
    index++;
}
return temp.ToArray();
```

文章圧縮

$$BWT \cdot MTF(2) \cdot RLE(3)$$

LZ系とかレンジコーダも検討はした。

Burrows-Wheeler Transform

(https://naoya-2.hatenadiary.org/entry/20081016/1224173077)
(https://tech.preferred.jp/ja/blog/burrows-wheeler-transform-lf-mapping/)

ソートはEditor上のみで確認。
復元では分布数え上げソートを使用。
別のソートを再帰でしようとしたら
時間が爆発した。

```
int[] countArr = new int[ushort.MaxValue + 1];
int[] sortArr = new int[target.Length];
int index = 0;
for (int i = 0; i < target.Length; i++)
{
    if (target[i] == (char)0) index = i;
    countArr[target[i]]++;
}
for (int i = 0; i < ushort.MaxValue; i++)
{
    countArr[i + 1] += countArr[i];
}
for (int i = target.Length - 1; i >= 0; i--)
{
    countArr[target[i]]--;
    sortArr[countArr[target[i]]] = i;
}
char[] result = new char[target.Length];
//string result2 = string.Empty;
for (int i = 0; i < sortArr.Length - 1; i++)
{
    index = sortArr[index];
    result[i] = target[index];
}

return new string(result);
```

Move To Front

String≒Char[]
Charは16bit正整数。

ジェネリックが使えないUdonで愚直に動かす。
⇒ 時間が掛かりすぎてUdonに止められる。
⇒ ただの入れ替えに変更。

Run Length Encoding

(https://algoful.com/Archive/Algorithm/RLE)
(http://www.nct9.ne.jp/m_hiroi/light/pyalgo29.html)

オーソドックスなタイプを実装。
3文字連続を圧縮するのが1番良さそう。

圧縮できたのか？

バイナリとして書き出した場合、ハフマン木を含めても圧縮できた。

（ような気がする。）

ビルドしてワールドに含めた場合、大して効果なし。

むしろ復元の時間で逆効果。

（多次元配列だから？）

最後に

- 借りれるものは借りよう。
- 凝り始める前に、一度落ち着こう。
- 使えるのはUdonだけじゃない。
- 心は強く、そして優しく。