



目次

私のお気に入り技術スタック	3
コードゴルフという界限	8
夏休みにすること	10
はじめての 時短 3D キャラクターモデリング	11
ライフゲームの制作と改造	15
D 言語を始めよう	19
NotebookLM で英語力 0 でも 洋書を読む入門	22
プロ基礎で作ったゲームの話	27
BitLocker って何?	31
Kinect v1 でフルトラッキングを目指す	40
随筆(笑)	45
あとがき	53

私のお気に入り技術スタック

Lime

はじめに

こんにちは Lime と申します。この記事では、実際に使ってみてめちゃくちゃ気に入った技術たちを簡単に紹介したいと思います。これらがいつでも絶対一番良いというわけではなく、適した場合において使用してみてとても良かったということを説明します。「こんな技術があるんだなー」とでも伝わればうれしいです。

1. プログラミング言語

1.1. TypeScript(殿堂入り)

JavaScript を書くぐらいなら頑張って TypeScript を書きたい。なにより変数や関数に型を指定することで、その変数・関数がどんな目的で定義されたかを自分で再確認することができるから。

```
let data = [1.24, 2.926, 3.03, 4.092, 5.17];  
  
data = data.reduce((pre, cur) => pre + cur, 0); // 16.458  
  
data = data.toFixed(2); // "16.46"
```

上は JavaScript で書けてしまうコード。いや別に JavaScript でも気を付けて書けばこんなことにはならないが、TypeScript だとそもそもエラーになる。

```

型 'number' を型 'number[]' に割り当ててはできません。 ts(2322)
let data: number[]
問題の表示 (Alt+F8)   クイックフィックス... (Ctrl+.)   Copilotを使用して修正する (Ctrl+I)
data = data.reduce((pre, cur) => pre + cur, 0); // 16.458
data = data.toFixed(2); // "16.46"

```

TypeScript で書きなおすとこんな感じになる。

```

const data: number[] = [1.24, 2.926, 3.03, 4.092, 5.17];

const sum: number = data.reduce((pre, cur) => pre + cur, 0); // 16.458

const fixed_data: string = sum.toFixed(2); // "16.46"

```

変数は使い回さず、それぞれ変えたほうがその変数の役割を明確にできる。TypeScript では加えて型を指定することで「できる操作」を限定して記述ミスを減らせる。(JSDoc? 何それ知らないなー)

2. フレームワーク

2.1. Astro

最近気に入ってよく使っているフレームワーク。静的サイトを公開するときビルド・アクセスが速いですよ、と言われている。検索するときは「astrojs」のようにすると出てくる。ページの組み立てをビルド時になるべく行い、アクセス時にクライアントで行う JS の実行をなくすことが速さの秘訣。

しかし、クライアント側でも JS を実行したい場合もそこそこある。この場合にも Astro は対応し、React など他ライブラリを用いた JS であってもちょうどいい感じにまとめてくれる。

ちなみに TypeScript の対応は完璧。

2.2. Vite

開発をより素早く、簡潔に取り掛かれるようにサポートする「次世代フロントエンドツール」。これはどちらかというビルドツールであってフレームワークではないような気もするがこちらで紹介。なぜなら Vite は先ほどの Astro 内部で利用されているから。

Vite を使うと、まず、初期設定がコマンド一つで 9 割終わる。そして、開発時にページを確認するための開発サーバー、これの起動が一瞬で完了。この爆速な開発環境で開発を終えたら、Vite でビルドする。これがとても良くて、JS のコードをいい感じにまとめたり、サイズを小さくして最適化してくれる。フロントエンド開発者にとってもやさしいツールなのである。

3. ライブラリ

3.1. HTML・JS 部分

3.1.1. React

ここ数年有名な JS ライブラリ。「コンポーネント」とか「仮想 DOM」とか「宣言的 UI」とか言われているあれ。Web ページの表示や操作に対する処理をまとめて記述しやすくすることで、その HTML 部品がどのような振る舞いをするかがわかりやすくなる。

3.1.2. nanostores

React と併せて使用する「状態管理ライブラリ」。状態管理ライブラリというのは、主にコンポーネントの状態を保存・管理しやすくするもので、複数コンポーネント間のデータ共有、データの永続化（再アクセス時までデータを残す）などを簡潔にしてくれる。

特に nanostores は中でもシンプルなライブラリで、導入してもページの読み込みが遅くなりにくく、また、難しい関数もほとんどない。

また、複数ページ間のデータ共有も簡単に記述できる。

3.2. CSS 部分 (CSS in JS)

「CSS in JS」というのは CSS を JS の中で記述できるようにするライブラリのことですね〜（そのまんま）。これを採用すると、いよいよ HTML・CSS・JS がすべて JS ファイルに記述されて、拡張子ごとのファイル区分から、機能ごとのファイル区分へと変更することができるようになる。どちらかというと、VSCode などのエディタでコードジャンプがしやすくなってうれしいというのが本音。

また、CSS in JS の中でも、CSS の生成をクライアント・サーバで実行時に行うか、ビルド時に済ませるかで分かれ、後者は「Zero-runtime CSS in JS」とか呼ばれる。Zero-runtime CSS in JS は後発で発展途上だが、そこそこよいものもあるので2つ取り上げる。

3.2.1. Linaria (WyW-in-JS)

Zero-runtime CSS in JS ライブラリの一つ。これは Zero-runtime CSS in JS 初期の頃からある歴史の長めなライブラリで、特徴は以下の通り。

- ・Zero-runtime なのでクライアント側では CSS ファイルを読み込むだけで、JS 実行の必要なし
 - ・タグ付きテンプレートリテラル記法に対応。開発者はいつもの CSS とほとんど変わらず書ける
 - ・React のコンポーネントに対応。スタイル付きのコンポーネントを作成したり、既存のコンポーネントにスタイルを付与したりできる
- 記述内容が CSS と同じなので、Stylelint といった CSS のリンター・フォーマッターが使える。

3.2.2. vanilla-extract

同じく Zero-runtime CSS in JS の一つ。割と新しめなライブラリで、以下が特徴。

- ・Zero-runtime なのでクライアント側では CSS ファイルを読み込むだけで、JS 実行の必要なし
- ・オブジェクト記法に対応。開発者は JS のオブジェクトと変わらず書ける

オブジェクト記法なので、TypeScript と併用することで、CSS プロパティの確認ができたり、各プロパティの入力補完ができたりする。

おわりに

あんまり丁寧に書けなかった。

それぞれ言いたいキーワードは含められたかなと思うので、気になった技術たちは自分で Google やら ChatGPT やら、Github Copilot など調べてみてください。

コードゴルフという界限

aotyam

コードゴルフとは、競プロの中でもできるだけ短いコードで AC しようという、タイパの権化みたいなゲームです。いろんな問題を見ていきましょう。

1. AtCoder Beginner Contest 145-A

整数 r を与える。半径 r の円の面積は単位円の面積の何倍かを出力せよ。

要するに 2 乗しろってことですね。Nibbles という言語は「*」だけ書くとなぜか入力して二乗されて出力されるのでこれで終わります。(1 byte)

```
*
```

2. AtCoder Beginner Contest 050-A

$A + B$ もしくは $A - B$ みたいな入力を与える。計算して出力せよ。bc という言語は式をそのまま解釈してなぜか出力くれるので何も書かなくても通りそうですが、空のソースコードが提出できないので空白でも置いて提出します。(1 byte)

3. AtCoder Regular Contest 033-A

正整数 N を与える。長さ N の文字列の連続部分列の個数を出力せよ。

つまり、 $N(N+1)/2$ を計算すれば良いですね。これを $N \sim 1$ の総和だと言い換えます。Nibbles で、をおくと $N \sim 1$ の配列ができて+でその総和を求めます。終わりです。(2 byte)

```
+,
```

4. AtCoder Beginner Contest 169-C

非負実数 A と B を与える。 $A \times B$ を切り捨てて出力せよ。
意外と難しい。ここで `dc` という言語が初登場です。「?」で入力して「*」で掛けて「1」で「/」割って「p」出力して終わりです。(5 byte)

```
?*1/p
```

5. AtCoder Beginner Contest 388-A

文字列 S を与える。 S の一文字目に UPC をつけて出力せよ。
><> という言語です。一文字ずつ入力できるのがこの問題との相性が抜群ですね。一文字入力して UPC を出力するだけです。見た目が面白い。CPU いおおおおおお。(11 byte)

```
"CPU"iioooo;
```

ありがとうございました。みんなもコードゴルフやろう。

夏休みにすること

きなこもち

私たち高専の夏休みは他の学校より長いです。
よって、できることがたくさんありますし、何もやらなければ時間だけを浪費し何も残らず勉強もおいて行かれます。

ですから、私は様々な思い出を作り、
勉強もおいて行かれないように「数学や C++,C#の学習」「様々な場所に訪れること」。

そして、一番大切な「実家の猫を愛でること」
この3つを目標に夏休みを過ごしたいと思います。

はじめての 時短 3D キャラクターモデリング

ななしの

こんにちは、電算部なのにプログラム関係のことを一度も書いていない気がするななしのです。今回は blender 歴 0 時間チュートリアルなしから 3D モデル作ってみました。

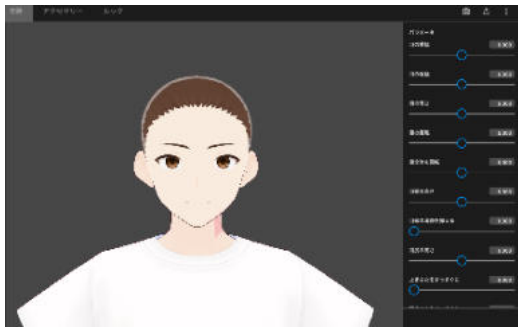
今回作ったモデルは見た目重視の改造性度外視モデルなので vroid → blender の方法で作ってみました。この方法は blender と比べると情報がないのですが意外と作りやすく初心者にお勧めかもしれません。特に vroid だけ触ったことある人なら時短になるかも…？



←初 vroid 作品

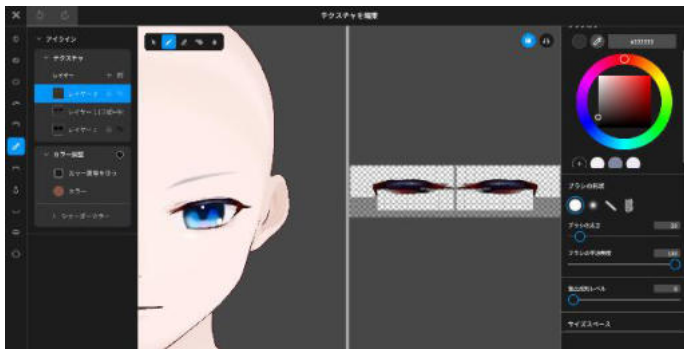
でもこの画像を見ればわかるように、vroid 単体は顔や髪は及第点でも立体的な小物や服が微妙です。なので顔や髪を作った後 blender に移してみることにしました。

まずは新規プロジェクトを作ります。初期状態は良く見る vroid 顔ですね。この要素を消すために目、肌すべてテクスチャを書き換えます。



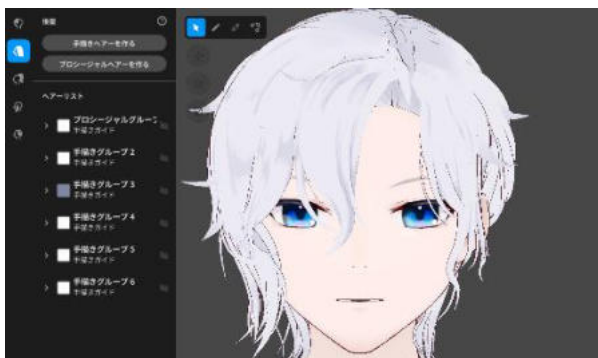
←ザ vroid

いろいろ書いてパラメータをいじりました。ここら辺はただの作業なので飛ばしていきます。



←テンプレは信用するな、
手描きが正義

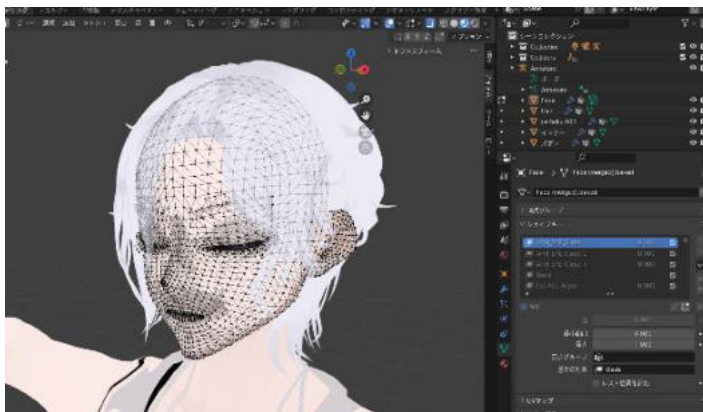
髪はシルエットを作るイメージで作って、中身はテクスチャを一本一本書くことで表現していきます。この作業量が一番多かったでしたぶん blender でやったほうが速いです。



Blender にアドオンを入れて.vrm を移して服を作っていきます。ここからは普通に 3D モデリングですけど体が先にある分やりやすいです。ついでに髪の毛のメッシュが崩壊している部分や頂点がつながっていない足などを直します。



基本的に vroid 製は表情が崩れやすいのでシェイプキーを直します。



次のステップですが、この文章があるということは UV 展開とか揺れもののボーン入れがこの部誌を書いている段階ではまだできていないということです、間に合いませんでした。

この方法の利点として、最初に作るのが服なので難しい人体を作るよりは操作やコツなどをつかみやすいですし、体のボーン、ウェイト塗りはほぼしなくても大丈夫です。下記の画像のようにボーンは最初からしっかりと設定されているので親子関係に気を付ければ大体動きます。



←設定されているボーンたち

vroid 製だとどうしても顔のメッシュがきれいにはならないのですが、見た目だけならばきれいなものができるので改造前提じゃない、映像作品を作りたい場合はこれでも十分だと思います。

作業時間的にもこのペースでいけば 50 時間程度で終わる（予定）なので初心者にしてはかなり効率がいいですし、時短というタイトルも回収できたので満足です。

今回参考にしたのは [こちら](#)

(<https://rio-vroidmmd.hatenablog.com/entry/2023/07/29/170724>)

と [こちら](#)

(<https://www.ay3s-room.com/entry/blender-vrm-edit-usefulchips#toc11>)

です。どちらも要点が抑えられていてとても分かりやすい解説になっているのでこの文を読んで気になった方はぜひ覗いてみてください。3D モデルは無限にあっていいものです、皆さん作りましょう！

ライフゲームの制作と改造

ゆーてぃー

今回の記事はアニメーション画像(GIF)を省略しています。note に GIF を含めた同一の記事があるため、そちらもご覧ください。



<https://note.com/ut0911/n/n9ad173f404c1>

○はじめに

みなさんこんにちは、ゆーてぃーです。今回は Processing を使って従来のライフゲームとルールを変えたライフゲームを作ってみました。

○ライフゲームとは

・概要

ライフゲームとは 1970 年にイギリスの数学者ジョン・ホートン・コンウェイ (John Horton Conway) が考案した数理モデルです。生命の誕生、生存、死といった流れを示しているものともいえます。(wikipedia より一部抜粋)

・ルール

ライフゲームでは格子内に白と黒のマスが存在し、白は死、黒は生を表します。世代という概念があり、世代ごとに変化が起こります。ターンが経過するようなイメージを持つとよいと思います。

マスの変化のルールは以下の通りです。ここでの隣接とは、周り 8 マス、つまり各マスを中心とした 3×3 のマスのうち自分以外のマスです。

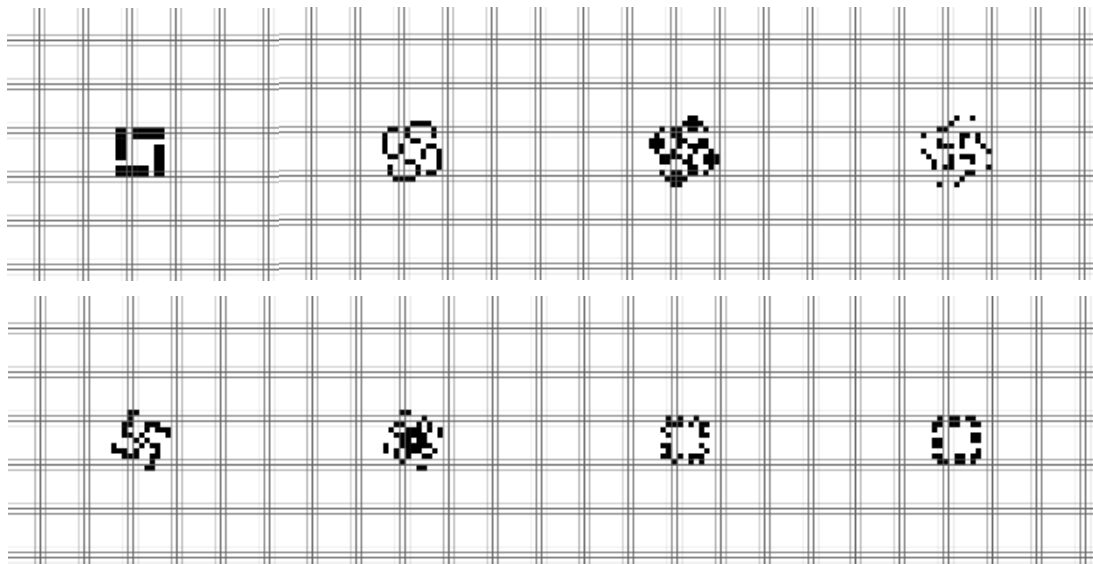
- ・ 誕生：死んでいるセルに隣接する生きたセルがちょうど3つ
→次の世代が誕生(白→黒)
- ・ 生存：生きているセルに隣接する生きたセルが2つか3つ
→次の世代でも生存(黒→黒)
- ・ 過疎：生きているセルに隣接する生きたセルが1つ以下
→過疎により死滅(黒→白)
- ・ 過密：生きているセルに隣接する生きたセルが4つ以上
→過密により死滅(黒→白)

ルールを見ると、生命の仕組みに近いと感じられます。

すべてのマスが死んでいると何も変化が起こらないため、あらかじめ初期パターンを用意してから動かします。パターンごとに動きは変わり、中には同じ動きをループするものもあります。

・ 特別なパターン

先ほど述べた、同じ動きをループするパターンの例の1つである「銀河」を紹介します。ほかにもいろいろなパターンがあるので、気になった人は調べてみてください。



8枚の画像の順にループする

・ 制作

Processing を用いて制作していきます。使用環境は以下の通りです。

- Windows11
- Processing 4.4.4

プログラムの中身を簡単に説明すると、現在の世代のマス状況に対して、ライフゲームのルールに従い次の世代にうつります。条件分岐を用いてルールを再現しました(以下の画像)。ここでの配列 **board** は現在の世代、配列 **NextGeneration** は次の世代のマス状況を表します。配列の値のうち 0 は死、1 は生です。

```
//誕生(死&生きたセルがちょうど3つ→1)
if (board[i]==0&&count==3) {
  NextGeneration[i]=1;
}

//生存(生&生きたセルが2つか3つ→1)
if (board[i]==1&&(count==2||count==3)) {
  NextGeneration[i]=1;
}
```

```
//過疎(生&生きたセルが1つ以下→0)
if (board[i]==1&&count<=1) {
  NextGeneration[i]=0;
}

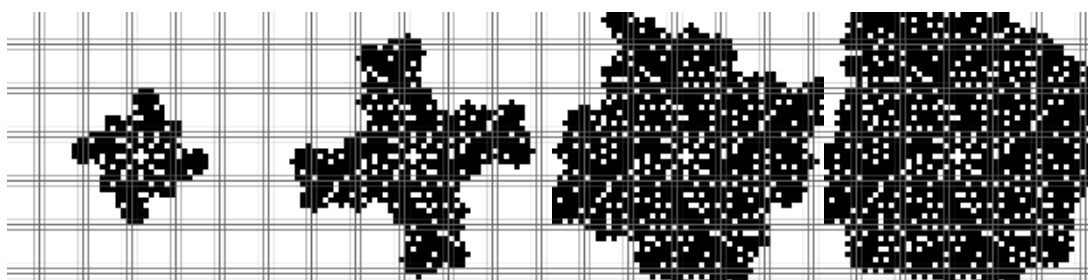
//過密(生&生きたセルが4つ以上→0)
if (board[i]==1&&count>=4) {
  NextGeneration[i]=0;
}
}
```

○改造

ライフゲームが作れたので、ここからマスの変化のルールを変えて改造します。

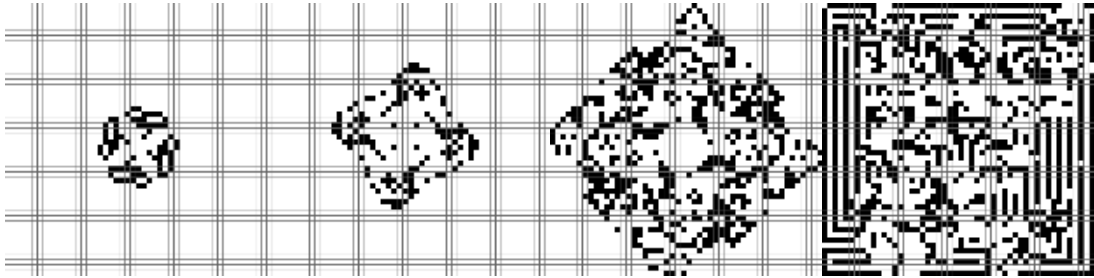
・増殖型

生きたマスがどんどん増えていくようにしたいので、過密の条件を削除しました。この状態で「銀河」の初期パターンを動かします。じわじわと生きたマスが増えていきました。隣接した生きたマスが3つの時だけ誕生するので、死んだままのマスも見られます。



・過疎で死なないパターン

今度は過疎の条件を削除しました。この状態で「銀河」の初期パターンを動かします。生きたマスが増えつつ過密になってくると死んでいきます。しばらく動かしていると規則的な形になりました。



過疎を削除したパターン

・奇数と偶数にした場合

ライフゲームのルールから大きくはずれて、周囲の生の数が奇数なら生、偶数なら死にしました。部分的に四角っぽいものが生成されたあとに全体が1つの四角になってそれが繰り返されました。



元のルールのライフゲームにはループするパターンがあるので、ルールを変えた場合もループするパターンがあるかもしれません。

○終わりに

今回は Processing でライフゲームを作り、改造しました。ネットで検索するとライフゲームで遊べるサイトもあるので遊んでみてください。

D 言語を始めよう

Nirocon

概要

D 言語コンパイラのインストールからビルド・実行方法、ライブラリのインストール方法について紹介します。素人による紹介なので間違いがあってもご容赦ください。。。

実行環境の準備

1. 公式サイトにアクセス

公式サイトに従って、D 言語のインストールを行います。

公式サイト：<https://dlang.org/download.html>

(MAC では下記のようなコマンドを実行してインストールすると楽でした。)

```
$ # DMD コンパイラのインストール
```

```
$ curl -fsS https://dlang.org/install.sh | bash -s  
dmd
```

(Windows では上のコマンドが使えなかったなので、インストーラーをダウンロードしてインストールしました。)

2. dub (パッケージマネージャ)

dub は公式のパッケージマネージャです。DMD をインストールすれば、自動で dub もついてきます。

動作確認:

```
$ dub --version
```

プロジェクトの作成・実行

プロジェクトを以下のコマンドで作成します。

```
$ # dub init <プロジェクト名>
```

```
$ dub init tetris
```

色々聞かれますが、とりあえず全部 enter で (遊ぶ分には) OK です。

ディレクトリ構造は次のようになります。

```
tetris
├── source
│   └── app.d
└── dub.json
```

ディレクトリを移動します。

```
$ cd tetris
```

以下のコマンドでビルド・実行します。

```
$ dub build # ビルド
```

```
$ dub run # 実行
```

ライブラリのインストール

ライブラリのインストールは以下のコマンドで行います。ここでは、テトリスで使った raylib-d のインストールを例として説明します。

以下のコマンドで raylib-d を設定ファイル

(dub.json, dub.selections.json)に追加します。

```
$ # dub add <ライブラリ名>
```

```
$ dub add raylib-d
```

```
$ dub upgrade
```

raylib-d をインストールします。

```
$ # dub run <ライブラリ名>:install
```

```
$ dub run raylib-d:install
```

まとめ

ここでは基本的な使い方についてまとめました。構文の説明は省略していますが、他言語のいいとこ取りのような構文が多かったので触りやすかったです。まだいろいろ調べきれてないところが多いのでもっと試していきたいです。

参考資料

D 言語の公式サイト：<https://dlang.org/>

D 言語の公式ドキュメント <https://dlang.org/dmd-doc/>

D 言語環境構築 2022 年版：

<https://qiita.com/outlandkarasu@github/items/7bb095d786b849378084>

令和の D 言語基礎文法最速マスター：

<https://qiita.com/lempiji/items/4c62edb2f37cfe7fe998>

NotebookLM で英語力 0 でも 洋書を読む入門

tamura_rasa

はじめに

NotebookLM という AI ツールをご存じですか？ NotebookLM は無料で誰でも使える Google が開発している AI ノート整理サポートのツールなのですが、そんなツールでとっても便利な使い方を見つけたのでここで共有していきます。なお、自分は電子メディア工学科なのでテクニカルな話は一切保証できません。

洋書が読みたいけど読めない

私が趣味で電子回路の勉強をしようと思ったとき、学校の図書館でいろいろ本を探していたのですが、そのときいかにも良書らしい本を見つけました。灰色のかっこいいデザインのカバー、1000 ページ以上の巨大なサイズ、ケンブリッジ大学出版、なんといってもその本は、洋書でした。自分が読める洋書なんて、Graded Readers だけです。でも他の本もなにか気に食わないのです。そんな形で、無謀ですが、一発狙って借りてみました。少し気分が高揚したまま、その気持ちのままで電子辞書片手にわからない単語があったら調べ、それを繰り返した結果、1 ページあたり 30 分かかりました。そんな形で前書きをなんとか乗り越えたものの、その後の”LEGAL NOTICE”で心がおられ、結局断念しました。今こうして振り返ってみて、何が足りなかったのかというと、もちろん基本的な語彙も足りないのですが、なんといっても一番の敗因は、専門分野の語は電子辞書には載っていないし、検索してもよくわからない。そのような中で読み続けるのは、単純に時間がかかる以上にキツかったのです。

NotebookLM との出会い

上に書いた理由によって、単純な Google 翻訳や DeepL でカメラで翻訳するのも除外です。本来ならばここであきらめるはずだったのですが、今は昔にはなかったたくさんの選択肢が探せばあります。そこで自分がみつけた一つの手段が NotebookLM です。もともと NotebookLM はテスト勉強の Anki デッキ作成のために使用していたのですが、よくよく考えたら洋書の翻訳にも使えそうだったので使ってみました。

やり方

通常の使い方の通りに NotebookLM にソースをいれます。ここでソースのデータを取得する方法ですが、力業で本をスマホのカメラで PDF データとしてスキャンをします。事前の準備はこれで終了です。あとは適当なプロンプトを入力することで、洋書を日本語に翻訳できます。例えば、このようなプロンプト

この資料のページ 8-9 を日本の電子工学の場で使用されているような単語で全文改変などはせずに日本語訳してください。途切れているところは途切れたような翻訳にしてください。数式は LaTeX の形式で出力し、図や表などはそれについての説明を入れるだけにしてください。

これを NotebookLM にいれると、

メモを追加

音声解説

マインドマップ

この資料のページ8-9を日本の電子工学の場で使用されているような単語で全文改変などはせずに日本語訳してください。途切れているところは途切れたような翻訳にしてください。数式はLaTeXの形式で出力し、図や表などはそれについての説明を入れるだけにしてください。

ご依頼いただいたページ8-9の日本語訳を以下に示します。日本の電子工学の分野で一般的に使用される用語を使用し、内容を全文改変せず、途切れている箇所はそのままの形で翻訳しています。数式はLaTeX形式で、図や表はそれについての説明を加えています。

10.3 電圧分圧器 ①

私たちは今、最も普及している電子回路の構成要素として、電圧分圧器について説明しましょう ①。

ここで6つの電圧分圧器を示します ①。非常に単純に言えば、電圧分圧器とは、特定の電圧入力を与える

入力を開始します...

1 個のソース

できた！

なんと洋書が読めちゃいました。ここでちょっと不自然な翻訳を見つけたとしても、それは NotebookLM に聞けば、

ロジックアナライザーってなに？日本でもそう呼ばれるの？

ロジックアナライザーについてのご質問にお答えします。

- ・ロジックアナライザーとは、デジタル回路のトラブルシューティング（問題解決）のために使用される特殊な目的の測定器です ①。
- ・本書の目次では、第11章「マイクロプロセッサ」の「11.14 エミュレーター、開発システム、ロジックアナライザー、および評価ボード」の項目に記載されています ②。これは、マイクロプロセッサのようなデジタルシステム開発において、エミュレーターや他の開発ツールと共に用いられることが示唆されています。
- ・電圧や電流を測定する多くの機器が存在する中で、オシロスコープが最も多機能であり、回路内の複数の点の電圧を時間に対して「見る」ことができます ①。マルチメーターは電圧、電流、抵抗を測定するのに適していますが、電圧の変化を追うような用途にはオシロスコープに及ばない、とされています ① ③。

日本での名称についてですが、ご依頼で「日本の電子工学の場で使用されているような単語で」という指示がありましたので、前回の翻訳でも使用したとおり、日本では**「ロジックアナライザー」**というカタカナ表記が一般的に使われています。これは英語の"Logic Analyzer"をそのまま音訳したものです。

あら便利。

これであなたが英語を読めなくとも、洋書が読めちゃう。とはいっても、あくまでこれは数ページを一度のプロンプトで都度ページを指定して翻訳するものなので、一度に PDF 全体を翻訳して、日本語訳版 PDF を出力するなんてことはできません。これは大きな課題です。いわゆる AI エージェントを使えばそういうこともできるのかもしれませんが、よくわかりません。とはいっても、NotebookLM の無料版の制限は一日 50 プロンプトなので、一日 100 ページ以上翻訳できるとすると、問題ないように思えます。とはいっても、出力を待つ時間はもどかしいですが。

実は、それに加えて、それ以上に理系の専門書を読むにあたってのデメリットがあります。それは、数式が LaTeX のソースコードのまま表示されることです！

```
$$ V_{Th} = V_{open, circuit} $$ $R_{Th} = \frac{V_{Th}}{I_{short, circuit}}$
```

この方法を分圧回路に適用してみましょう ①。分圧回路は抵抗と単一の電圧源からなる2端子回路網であるため、テブナン等価回路が存在するはず ①。開放電圧は

```
$$ V = V_{in} \frac{R_2}{R_1 + R_2} $$
```

うーん、これだとともに読めません、まだ $\frac{\{\}}{\{\}}$ ぐらいならわかっても、もはや微積分などが入ってくると意味不明です。これを改善したい、なにがしたいのかというと、これをちゃんと LaTeX 表記にしたい。そう思いとりあえずググって見たらそんな拡張機能を発見！とりあえず入れて見たら。。。うごかない！

こうなったらようやく困ったものなので、とりあえず ChatGPT に聞いてみました。そうしたら、KaTeX なるものを使えばウェブブラウザで LaTeX 表記に変換できるようで、とりあえず拡張機能を頼んで作ってもらいました。そうしたら。。。

$$V_{Th} = V_{open,circuit}$$

$$R_{Th} = \frac{V_{Th}}{I_{short,circuit}}$$

この方法を分圧回路に適用してみましょう ①。分圧回路は抵抗と単一の電圧源からなる2端子回路網であるため、テブナン等価回路が存在するはず ①。開放電圧は

$$V = V_{in} \frac{R_2}{R_1 + R_2}$$

です ①。

このとおり！

数式がちゃんと表示されていると安心感が違いますね。ChatGPT に感謝。

おわりに

こんな感じで、現代では英語力がなくとも洋書が読めます。洋書とはいっても、英語に限らず、ドイツ語でも中国語でも NotebookLM に対応している言語ならなんでも読めます。例えばインターネットアーカイブで著作権切れの 19 世紀の本などを翻訳することも可能であるし、UnityLearn の英語のページも日本語訳することも可能ですので、いろんな機会が広がります。みなさんもこの方法で楽しい洋書ライフを送ってください！

プロ基礎で作ったゲームの話

S

・自己紹介

Sです。今日は高専に入るまでプログラミングに触れたことがなかった私がプログラミング基礎の授業で作ったゲームについて紹介していきたいと思います。

拙い文章ですが、勘弁してもらえると嬉しいです。

・使用した言語

使ったプログラミング言語は Python です。

ライブラリは Pygame を使用しました。

・ゲームの概要

ボール（自分）を矢印キーで動かして、敵をよけながらステージを突破していくゲームです。



敵に当たる、または画面上端以外から画面外に出るとライフが一つ減ります。

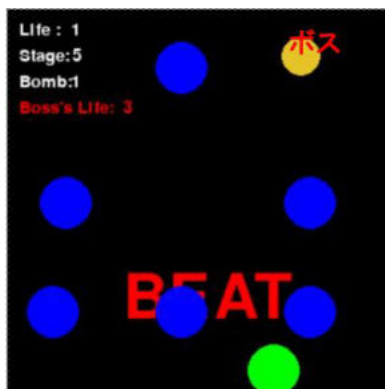
画面上端に到達すると次のステージが始まり、敵の数が増えます。



最終的には敵の数が六体になります。



このステージをクリアするとボスステージが始まります。
ボスはしばらく敵をよけていると現れます。



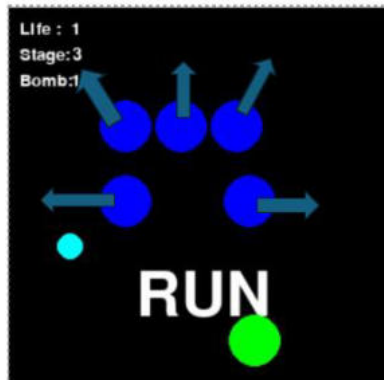
ボスに触れることでボスを倒すことができ、三回倒すとゲームクリアです。



また、水色のボールを取ることで、自分のライフが一つ増えます。



また、スペースキーを押すと敵を画面端に飛ばすことができます。



・これからの事(改良案)

このゲームをよりよくするために以下の三つの改良案を試していきたいと思います。

1. ステージ数を増やす、またステージごとに異なる特徴を追加してみたい。
2. タイムアタック的な要素を追加して、友達同士で競えるようにしたい。

3. 敵の攻撃方法を多様にすることでよりゲームに緊張感を持たせたい。

・まとめ

私は、これからもプログラミングを学び続けていろいろなものを作って行きたいです。

最後まで読んでいただきありがとうございました。

BitLocker って何？

渡辺

渡辺です。最近、私のパソコンで突然 BitLocker という機能が動作しました。いろいろ調べてみると、これは Windows に標準搭載されているディスク暗号化機能で、気を付けて利用しなければならない機能だとわかりました。今回は、私が体験した出来事を含めて BitLocker についてまとめてみます。

○そもそも BitLocker とは？

BitLocker は、一言でいうと「パソコンのストレージを暗号化し、盗難や紛失時のデータ流出を防ぐ機能」です。

BitLocker の仕組み

- ・PC の中には TPM という小さなセキュリティチップがあります。
- ・TPM は「このストレージはこの PC のものだ」と記憶していて、起動時に 2 つが一致しているかを確認します。
- ・一致していれば自動でストレージの暗号化を解除します。
- ・ストレージを別の PC に接続するなどして、TPM とストレージが一致しない場合には、暗号化を解除できず回復キーが必要になります。

○回復キーとは？

BitLocker を有効化すると、48 桁のランダムな数字で構成された回復キーが発行されます。この回復キーは、BitLocker で暗号化されたストレージのロックを自動的に解除できない場合に必要になり、入力することでストレージへのアクセスを回復することができます。

回復キーが求められる場面の例

- ・ログイン時に PIN コードなどでの認証に規定回数以上失敗したとき
- ・ストレージを別の PC に接続したとき
- ・ストレージを取り換えた時
- ・何らかの不具合(過去には WindowsUpdate のファイルに問題があったことで発生したことも)

○メリット・デメリットと必要性

メリット

- ・紛失や盗難、処分の際に、個人情報や機密情報の流出を防ぐことができる
- ・標準搭載されていて新たにソフトウェアをインストールする必要がない
- ・外付け HDD や SSD、USB の暗号化にも対応できる

デメリット

- ・回復キーの管理が面倒
- ・まれに誤作動が起きて回復キーの入力が求められる可能性がある
- ・回復キーを紛失すると復元が難しくなる
- ・PC やストレージが故障した際、データの復旧が難しくなる

必要性

- ・企業や学校、機密情報を扱う人にとっては有用な機能。
- ・あまり外へ持ち出さない、機密性が低い場合などに利用するメリットは小さい

上にあげたようなデメリットもあるため、利用するなら回復キーをしっかり管理し、必要ないと感じたら無効化してしまうのもあります。

○体験談

出来事

私の場合次のようなことが起こりました。

- ・ PC をスリープをし、防水性のタブレットケースに入れて放置。
- ・ 翌日パソコンがケース内で高温になっていた。
- ・ 開くと BitLocker という青い画面が表示され、回復キーの入力が求められた。
- ・ マイクロソフトアカウントにアクセスすることで回復キーを見つけることができ、ロックを解除できた。



なぜ回復キーを求められたのか

ChatGPT とやり取りをした結果、次のようなものが原因の可能性があるということになりました。

-
1. PC をスリープ状態で通気性の悪いケース内に保管し、熱がこもる
 2. TPM が高温状態によって一時的に正常に動作しなくなる
 3. TPM が使えないため暗号化の自動解除ができず、信頼できる状態での起動ではないと判断し、セキュリティ上の理由から回復キーの入力を要求
-

あくまでも私自身の場合で、スリープ状態が必ず危険なわけではありません。また、パソコンの不具合や、アップデート不足が原因だった可能性もあります。

ChatGPT によると BitLocker は誤作動だった可能性が高いようです。出た時は本当に焦ったので、これからはすぐに回復キーを見つけられるよう、メモに書いたり印刷したりして対策しました。

○BitLocker の管理をどうすればよいか

BitLocker は利用するなら回復キーをしっかり管理する必要があります。また、特に機密情報などを扱ってなくて、回復キーの管理が面倒なら BitLocker を無効化してしまうのも選択肢の一つです。

ここからは、

- ・現在 BitLocker が有効化されているかの確認方法
- ・BitLocker の回復キーの確認・保管方法
- ・BitLocker の無効化方法

についてまとめます。正直ほかの web サイトを見たほうがわかりやすいかもしれないので、あくまで参考程度にしてください。

○BitLocker が有効化されているかの確認

1. タスクバーにあるエクスプローラーを開く

2. 左下にある PC をクリックする
3. 表示されるドライブに鍵のマークがついていれば有効化されている



○BitLocker の回復キーの確認・保管方法

回復キーの確認方法は、

- ・ パソコン本体で確認する方法
- ・ マイクロソフトアカウントで確認する方法

の 2 つがあります。

すでに回復キーを求められてパソコンが起動できない状態なら、マイクロソフトアカウントで確認するしかありません。

パソコン本体で確認する方法

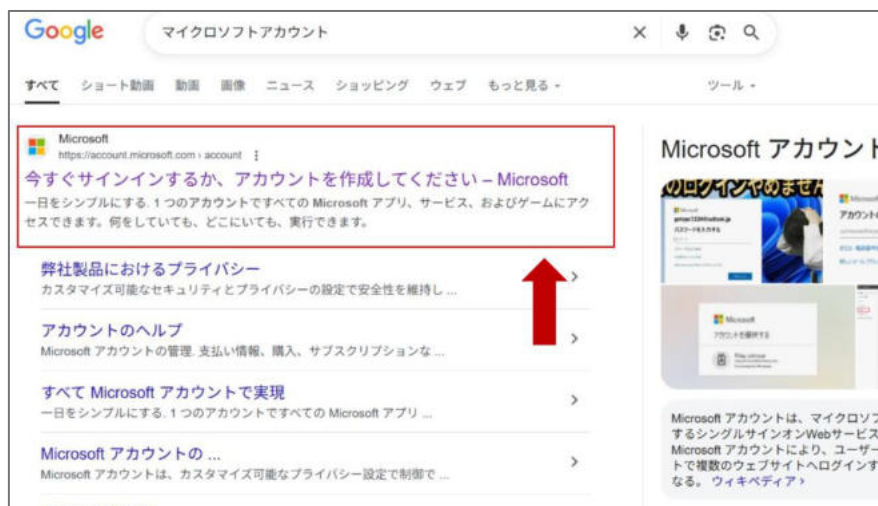
1. タスクバーの検索欄に「コントロールパネル」と入力し、開く
2. 「システムとセキュリティ」をクリック
3. 「デバイスの暗号化」をクリック

4. 「回復キーのバックアップ」をクリック
5. バックアップ方法を選択(「ファイルに保存する」を選んだ場合は外付け USB に保存する必要があります)



マイクロソフトアカウントで確認する方法

1. ブラウザで「マイクロソフトアカウント」と検索し、マイクロソフトアカウントのメールアドレスを使ってサインインする



2. デバイス欄の「詳細の表示」をクリック
→ BitLocker データ保護の「回復キーの管理」をクリック



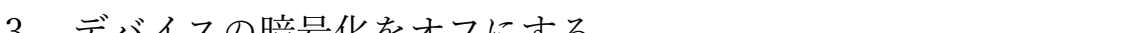
3. デバイス名とキーID、回復キーが表示される



人によってはいくつか表示されることがあります。回復キーを求められたときには、画面に表示されるキーID に対応した回復キーを入れる必要があります。

○BitLocker の無効化方法

1. スタートから設定を開く
2. 「プライバシーとセキュリティ」から「デバイスの暗号化」をクリック



プライバシーとセキュリティ > デバイスの暗号化

小しでも参考になれば幸いで十、おかしなところばかりなので、こ

BitLocker の回復キーを突然求められる時の原因や確認方法を紹介：

https://licensecounter.jp/win_migration/blog/detail032/

BitLocker の設定方法・活用方法とメリット・デメリット：

https://www.buffalo.jp/recovery/contents/detail63_bitlocker.html

Kinect v1 でフルトラッキングを目指す

新井 智也

1. はじめに

VR はゴーグルを装着するだけで別世界へと行ける革新的な技術ですが、ゴーグルとコントローラーだけでは頭部と手の動きしか反映されず、没入感には限界があります。そこで登場するのが、フルトラッキングです。フルトラッキングとは、全身の動きや位置を取得して、VR 内の自分のキャラクターに反映させるものです。それを実現させるためのデバイスには多くの種類がありますが、ネットで調べると一般的な構成では 3 万円以上はかかってしまうようです。高すぎます。本格的にフルトラッキングをしたいならば仕方のない出費ですが、私のような「なんとなくフルトラッキングってやつ、やってみたいなあ」という人にとっては高すぎるように感じます。そこで、今回は某ジャンク屋で購入した動作未確認で 1000 円未満であった Kinect v1(Xbox 360 Kinect)を用いてフルトラッキングを目指します。



Kinect v1(Wikipedia より引用)

2. 構成

使用したデバイスおよびソフトウェアは以下の通りです。

- ・ Kinect v1(Xbox 360 Kinect)(以下 Kinect と呼ぶ)
- ・ Kinect を PC に接続するためのアダプター
- ・ Meta Quest 2(64GB)(以下 Quest 2 と呼ぶ)
- ・ デスクトップ PC(CPU: Ryzen 5 3500 , GPU: GTX 1650 Super, Windows 11)

- Kinect for Windows SDK v1.8
- Amethyst (Kinectなどでトラッキングした結果を SteamVR など
で利用するためのソフト)
- SteamVR
- Meta Quest Link
- VRChat

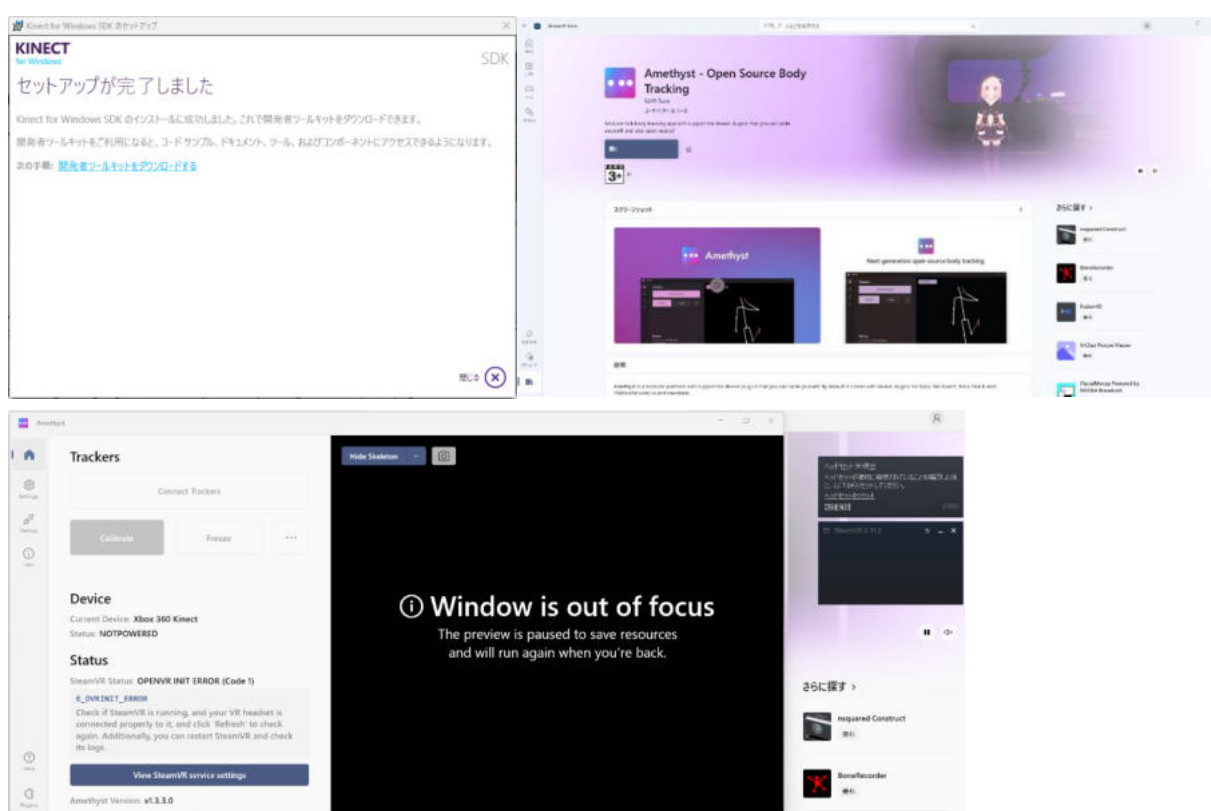
3. 準備

まず、Kinect を Windows で機能させるために Kinect for Windows SDK v1.8 を導入します。(1 枚目)

次に、Amethyst を導入します。Microsoft Store でダウンロードできました。(2 枚目)

Amethyst を起動し、初期設定を進めます。

すると SteamVR が起動して 3 枚目のようになるので、Meta Quest Link を用いて Quest 2 を SteamVR に接続します。(3 枚目)



左上、右上、下の順に 1,2,3 枚目

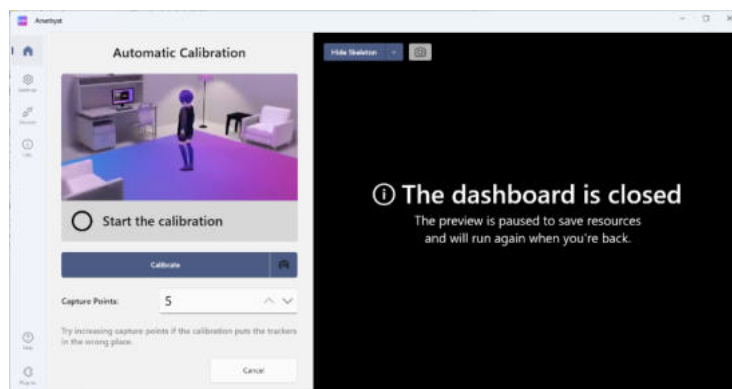
これで Amethyst、SteamVR の設定を調整するだけで Kinect をフルトラッキング用デバイスとして使えるようになりました。

4. 実際に使ってみる

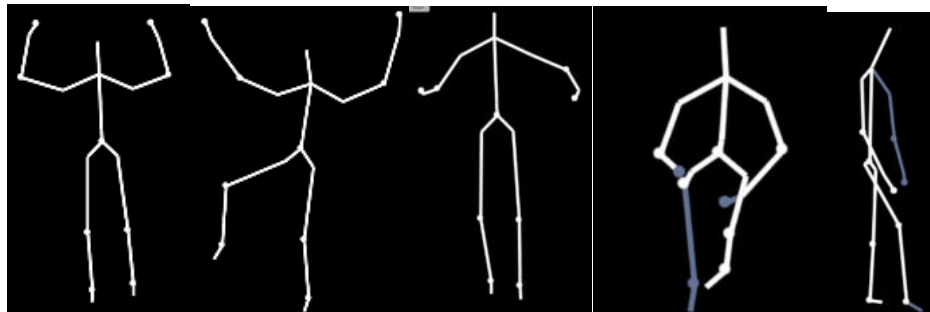
4-1. Amethyst 上で動かす。

まずは Amethyst 上で動かしてみます。

Amethyst を立ち上げた後は毎回キャリブレーション(補正)が必要です。



完了後、Kinect と接続して表示した結果が以下です。



左から 1,2 枚目は立っている状態で、3 枚目はジャンプ、4 枚目が体育座り、5 枚目が横向きで立っている状態です。1,2,3 枚目は完全ではないですが、しっかりと捉えられています。しかし、4,5 枚目のように腕や足が重なった複雑な状態では精度が低下し、正面を向いて立っているときが最も安定して認識される結果となりました。

4-2. VRChat で使ってみる

次に実際に VRChat 内で使ってみます。以下がそのときの写真です。



(アバターは VRC 合法チート研究会様の「チセ」ちゃんを使っています)

左から、1,2 枚目が正面を向いて手足を挙げて立っている状態、3 枚目が足踏みしている状態、4 枚目がジャンプ着地時、5 枚目が体育座りをしている状態です。基本的には Amethyst 上で動かしたときと同じような結果でしたが、ジャンプ着地時には、上半身が過度に沈み込み、不自然な姿勢になる現象が発生しました。VRChat 内の設定や、Amethyst 側の設定、Kinect の位置でもう少し精度を上げられる可能性はありますが、少なくとも座り込む、体を横に向ける、激しい動きをするときにはあまり向かないことがわかりました。

4-EX. Quest2 の機能のハンドトラッキングと併用する

Quest 2 にはゴーグルのカメラを利用したハンドトラッキング機能があります。これと Kinect から得た全身のトラッキングデータを併用すれば、さらに詳細な自分の姿をゲーム内に出現させることができるはずです。これには調べた限りですと、Quest 2 版 VRChat に Amethyst から OSC という通信規格を用いて全身のデータを送信する方法と、Steam Link(Steam が起動している PC に無線で接続するためのソフトウェア)を用いてハンドトラッキングのデータとともに Quest 2 を PC につなぐ方法がありますが、私の環境では OSC がうまくいかず、Steam Link での接続もうまくいかなかったため、今後の課題とさせていただきます。

5. 最後に

Kinect でフルトラッキングを実現しようとする、正面を向いて立つ必要があるなど、多少の制約がかかってしまうものの、フルトラッキング自体はできることがわかりました。一般的なものを購入すれば、3 万円以上はかかるような機能ですから、一般的な構成の 30 分の 1 ほどの価格でここまで実現できたのは、フルトラッキングの体験としてはとてもコストパフォーマンスに優れた方法といえると思います。

これからも、すぐに楽な方法に飛びつくのではなく、ほかに方法がないのか、もっと安く済ませる方法がないのかなど、じっくり考えたうえで行動できるようにしていきたいです。

随筆(笑)

arrow/JJ1URG

初めにの前に

あ、こんにちは。これ最初締め切り日に書いてたんですけどやる気なくして提出が一週間後になりました。途中で人格変わってます。部長、本当にすみませんでした。

初めに

私は今窮地に立たされている。そう、この部誌の締め切りは本日なのである。もっと言うと、現在時刻は 22 時 22 分。あと 1 時間半強で本日が終わる。さあどうしよう。

現在地点：スタバ

今私はスターバックスにいる。しかしこれを執筆しているパソコンはなんと Macbook ではない。Windows 機である。しかもマウスコンピュータ製。イキリ Mac が出来ない。仕方がないので今私が飲んでいる飲み物を紹介しよう。

ダークモカフラペチーノである。

サイズは贅沢にも Venti。なんと驚異の 590mL である。どうやら Venti はイタリア語で 20 を意味し、20 オンス(約 590mL)から来ているそうだ。オプションとしてチョコレートシロップ追加(+55 円)、ショット追加(+55 円)、ダークチョコレートパウダー多め(+0 円)、チョコレートチップ多め(+0 円)にしている。

基本的にはチョコレートの味が前面に出ている。ダークという名前の通り、砂糖の割合は一般的なミルクチョコレートよりもかなり少なめだと思われるが、それでも甘い。そこにエスプレッソショットの濃い苦みとマッチしているのである。コーヒーとチョコレートのコンビなんておいしさに決まっている。おいしい。10 分で飲み干してしまった。

おいしそうだと思ったそこでないあなた。795 円を払えば無料で飲めるぞ。特においしそうと思わなかったそのあなた。正しい。こんなものに約 4.84×10^1 分(985 円/時として計算。2025 年 8 月 4 日現在の群馬県における最低賃金)のお金を費やすな。お金を貯めろ。ところで私は分割払いが控えているゾ☆(((

よし、スタバで文字数結構稼いだぞ。

さすがに…

電算部の部誌なのに、さすがに IT 要素が無さすぎである。最近 iPad を購入したので、今まで Apple 製品に一つも触れてこなかった私の驚いたポイントを紹介しよう。

さては分割払いこれかと察したその君。正しい。

Android 使いの私が iPad に驚いた点

iPad は iPadOS を搭載している。これは、macOS からフォークされた Unix 系の OS である。それに対して、Android は Linux ベースのオープンソースの OS である。(だから何だ)

まず驚いたのは、3 ボタンナビゲーションが無いことである。画面下にあるバーを上をスワイプしてホームに戻ったり、タスク一覧を表示したりするしかない。いや、Android にもその設定はある。慣れの問題である。しかしながら、ボタンではなくジェスチャーにより操作することを余儀なくされ、私の中では実際に手汗などの状況でうまくいかないことがよく起きる。タスク一覧を表示したいのにホームに戻ってしまうなど、意図しない動作がしばしば起きる。3 ボタンナビゲーションの選択肢は正直欲しい。

次に、ファイルシステムと画像が切り離されていることである。ファイルはファイルで、画像は画像で異なるソフトウェアから独立にアクセスしなければならない、ファイルシステムから画像を閲覧したり、反対に画像からファイルシステムとしての画像にアクセスしたりすることができない。いやあるかもしれない。少なくとも私には分からない。ファイルシステムすらも、(内部的には OS レベルのファイル

システムの一部であるのだろうが)サンドボックス化していることには驚いた。Android においては、OS レベルのファイルシステムをそのまま活用する。Google photo による画像閲覧も、photo が勝手に画像を見つけ出し、画像が格納されているフォルダを並列に表示してくれる。

ただ、このサンドボックス化されたファイルシステムは正直使いやすい。余計なことを考えなくてよいからだ。とにかく PDF を階層別に突っ込んで、閲覧する。便利。

多分もっとあるが、もう書くのがめんどくさいので次の話題に行く。

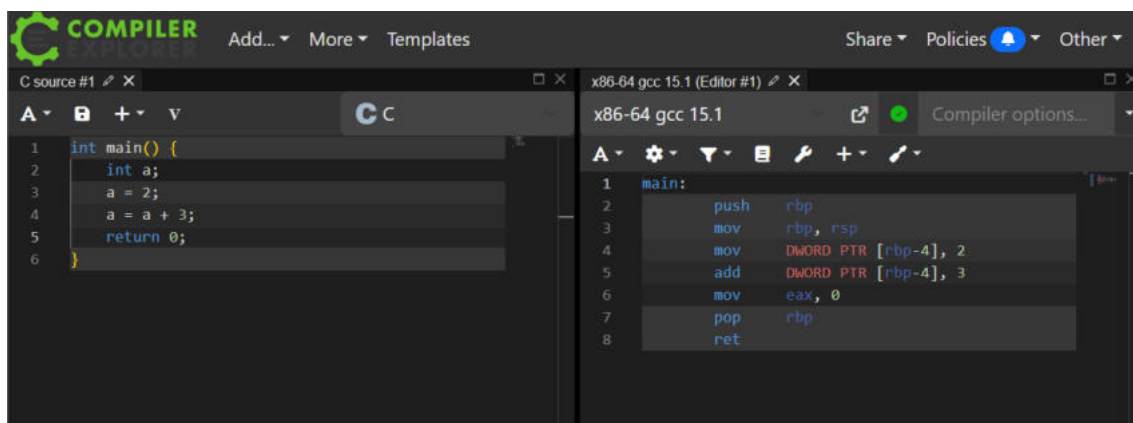
C 言語ってなにしてるんだろう

人格が変わったので敬語で書きます。(!?)

C 言語を書いたことがある人は、なぜに C 言語はこんなに自由度がないのかとパソコンを殴りたくなった経験はありませんか。文字列型がなく char 型の配列で表現？変数を初期化しないと変な数字が入ってる？？配列の範囲外にアクセスできる？？？

そんな疑問を解消するための一助として、C ソースのコンパイル後はどうなっているのかを見てみたいと思います。

[Compiler Explorer](<https://godbolt.org/>)で実際に見ていきましょう。このサイトは、あらゆるプログラミング言語のコンパイラ結果を表示してくれるサイトです。すごい。これを使って、C 言語をアセンブリにしてみましょう。使用するコンパイラは「x86-64 gcc 15.1」です。



The screenshot shows the Compiler Explorer interface. On the left, the C source code is displayed in a dark-themed editor:

```
1 int main() {  
2     int a;  
3     a = 2;  
4     a = a + 3;  
5     return 0;  
6 }
```

On the right, the assembly output for x86-64 gcc 15.1 is shown:

```
1 main:  
2     push    rbp  
3     mov     rbp, rsp  
4     mov     DWORD PTR [rbp-4], 2  
5     add     DWORD PTR [rbp-4], 3  
6     mov     eax, 0  
7     pop     rbp  
8     ret
```

サイトにアクセスすると、以上のように左側にソース、右側にコンパイラ結果が表示されます。しかも、コンパイラ後の文はコンパイル前のソースのどの部分に相当するかが色分けされます。

さて、左側の C ソースを詳しく見ていきましょう。

```
1  int main() {  
2      int a;  
3      a = 2;  
4      a = a + 3;  
5      return 0;  
6  }
```

Int 型の変数 a を宣言し、2 を代入、3 を加えて再代入しています。

printf で出力等なにもしていない無意味コードですが、これはあくまでコンパイル結果を見るためのテストコードですので、悪しからず。

さあ、アセンブリではどうなっているでしょう。

```
1  main:  
2      push    rbp  
3      mov     rbp, rsp  
4      mov     DWORD PTR [rbp-4], 2  
5      add     DWORD PTR [rbp-4], 3  
6      mov     eax, 0  
7      pop     rbp  
8      ret
```

うーん、ナニコレ

なんか、関数名だった main は左上にあって、何やら後にコロンのついていますね。しかも、そのあとに続く文たちはインデントされていますねえ。

まず、main:は、ラベルと呼ばれるものになっているようですね。ラベルとは、アセンブリソース内での目印です。分岐処理などで、分岐後の行先をラベルを用いて指定することが出来たりします。

さて、次からが本番です。まず、アセンブリ言語は命令と、命令に必要な引数のようなもの(引数とは違いますがここでは便宜上引数と

呼びます)による一行を一つの処理とし、それを並べたものになっています。そして、誤解を恐れずに言えば、この命令こそ CPU が直接実行できる命令となるわけです。

プログラムは、様々なサブルーチンの集合により成り立っているといっても過言ではありません。そんなサブルーチンを支えているのが、スタックです。

スタックとは、後入れ先出しのデータ構造のことをいいます。詳しくはググってください。そして、スタックを実現するために必要なものは、スタックの一番上の場所だけです。スタックに値を push したり、pop したら、スタックの一番上の場所を更新するだけで済みます。

アセンブリのような低レイヤで使われるスタックも、メインメモリ上で実装されています。そして、このスタックの一番上の場所であるアドレスを保持しているのが、スタックポインタと呼ばれるレジスタです。ここで、このスタックはメインメモリのアドレスが小さくなる方向に成長します。←ここ大事

アセンブリコード中の 2、3、及び 7 行目がスタック関連の操作となります。

まず、2 行目における push 命令が、スタックを積む命令です。まず、x64 におけるスタックポインタを格納しているレジスタは、rsp です。push 命令は、この rsp に格納されているアドレスを“デクリメント”し、その後 push 命令の引数、ここでは rbp に格納されている値をデクリメント後の rsp が示す場所に格納します。rbp とはベースポインタと呼ばれるレジスタで、現在作業中の基準となるアドレスです。そう、サブルーチンはこのベースポインタを基準として、スタック上でいろいろな計算をします。2 行目で積んだ rbp の値とはすなわち、このサブルーチンである main を呼び出したより上位存在のベースポインタです。

3 行目は、mov 命令となっています。引数には rbp と rsp が渡されています。これは、rbp に rsp の値がコピーされることを意味しています。そう、スタックポインタの値をベースポインタにコピーし、作業の基準としています。

飛ばして 7 行目では、今までの作業が終わった後、スタックから値を呼び戻して `rbp` に格納し、`rsp` を”インクリメント”します。このとき呼び戻した値は、より上位のサブルーチンのベースポインタであり、8 行目でプログラムがより上位のサブルーチンへ戻るわけです。さて、次からがいよいよ本懐となる部分です。といっても 3 行しかありませんが。

4 行目は、`mov` 命令という文になっています。これは、値をコピーする命令です。この `mov` 命令の引数としては、`DWORD PTR [rbp-4]` と、2 が与えられています。つまりですね、`DWORD PTR [rbp-4]` に 2 をコピーするという文です。そして、これに対応する C ソースは 3 行目の `a = 2` みたいです。ということは、つまり、`DWORD PTR [rbp-4]` が `a` に相当するものだということなんダナ (・▼・)

じゃあ何よこの `DWORD PTR [rbp-4]` ? って感じですね。さて、まず `[rbp-4]` について説明したいと思います。なんで右から説明するんだと。まあ、いいじゃないですか。

さて、これはベースポインタ `rbp` から 4 を引いた値をアドレスとして、メモリ上のそのアドレスに存在する値を読み出すことを表しています。つまり、ベースポインタが指し示すアドレスの 4 バイト前のアドレスが、変数 `a` の実体なのです。そして、なぜアドレスを 4 バイトもどしたのかといえば、変数 `a` の大きさが 4 バイトだからなのです。このアドレスを 4 バイトとして解釈するために、`DWORD PTR` という文字列がついています。この `DWORD` が、アドレスの指し示す先について、4 バイトを 1 つの塊として解釈せよということを表しています。ちょうど `int` が 4 バイトであることに一致しますね。

5 行目は、いよいよ `add` 命令となっています。これは、1 つ目の引数と 2 つ目の引数を加算し、1 つ目の引数の場所に格納せよということを表しています。

あとは何もしてないのでこれで終わりですが、6 行目はレジスタ `eax` に 0 を格納していますね。これはどうやら `return 0` に対応するようです。サブルーチン終了直前に `eax` に格納されている値がこのサブルーチンの戻り値となるわけですね。

前述の通り、アセンブリ言語は CPU が直接実行できる命令の集合です。それを、本当に CPU に直接流せるようにしたものが機械語です。アセンブリ言語はあくまでテキストファイルですからね。アセンブリソースにおける単語一つ一つをバイナリに置き換え、機械語に直す操作がアセンブルとなるわけです。

終わりに

急に C 言語とアセンブリについて語り始めてしまいましたが、多分理解できる人は少ないと思います。補足説明すべき事項はたくさんありますし、なによりこれ随筆ですからね。筆者自身もほとんど理解できていませんし。しかし、これを機に C 言語が如何にアセンブリ言語を書きやすくしただけのものなのか、他の言語が如何に高級なのか勉強を進めるきっかけになったら面白いです。(?)

ところで、これを読めているということはですね、部長が寛大な慈愛の心を以ってこの記事に掲載してくれたということです(現在締め切り日から一週間後)。C 言語が如何に機械語であるか、を語ろうとして一週間が過ぎました。すいませんでした。

まさかここまで読んでいる人がいるとは思えないが、ここまで読んだ物好きなそこのあなた。何してるんですか、勉強してください！！！！

参考文献

Compiler Explorer

<<https://godbolt.org/>>

(2025 年 8 月 12 日参照)

@tobira-code. *x86_64 プロセッサのスタックを理解する*. Qiita. 2016

<<https://qiita.com/tobira-code/items/75d3034aed8bb9828981>>

(2025 年 8 月 13 日参照)

@kaito_tateyama. *アセンブリに触れてみよう*. Qiita. 2019

<https://qiita.com/kaito_tateyama/items/89272098f4b286b64115>

(2025 年 8 月 13 日参照)

@edo_m18. アセンブラに手を出してみる. Qiita. 2018

<https://qiita.com/edo_m18/items/83c63cd69f119d0b9831>

(2025 年 8 月 13 日参照)

2-4. メモリーへの転送. アセンブラ入門

<https://programmer.main.jp/assembler/2_4.html>

(2025 年 8 月 13 日参照)

あとがき

部長のゆうていーです。この度は D 言語 2025 年度版を手にとっていただきありがとうございました。

今回もたくさんの記事が集まりました。前から読んでもよし、あえて後ろから読んでもよし。目次を見て自分の気になる記事から読むのもよし。同じ記事を何回読んでもよし。

プログラミングには時間計算量という考え方があります。時間計算量とは処理にかかる時間をプログラムからある程度見積ったものです。この D 言語をあなたが読む時間計算量はどれくらいになるのでしょうか？記事数を N としたら時間計算量は $O(N)$ (時間計算量はこのように表します)と考えられますが、同じ記事を何回も読んだり読む順番を変えたりすれば、時間計算量は $O(\infty)$ になるかもしれません。本来、時間計算量は少ないほうが良いですが、この D 言語を読む時間計算量はいくら多くても構いません。無限の時間、無限の読み方で楽しんでもらえると嬉しいです。

改めまして、ここまで読んでいただき本当にありがとうございました。ぜひ次回の D 言語をお楽しみに！

D 言語 2025 年版

2025 年 8 月 21 日 初版

著者 Lime, aotyam, きなこもち, ななしの,
 ゆーてぃー, Nirocon, tamura_rasa, S,
 渡辺, 新井 智也, arrow/JJ1URG

校閲 Lime, ゆーてぃー, Nirocon, 渡辺

表紙 ゆーてぃー

編集発行 群馬高専電算部

X(Twitter) @GNCT_densan

誤字脱字等ありましたら、X(@GNCT_densan)までご連絡ください。

©2025 National Institute of Technology, Gunma College Densan Club.

群馬高専電算部