目次

第 1 章

Brainf*ck であそぼう！

78 回生　locker

1.1　はじめに

こんにちは！ パソコン部の部長を務めております、locker と申します。普段はどちらかというと数学をやっているのですが、なぜでしょうか、部長になりました。なんたらはなんたらより奇というやつですね（本当？）。近頃ついに高 2 になってしまい老いを実感しているところです。

今回の部誌では、パソコンに詳しくない人でも楽しんでいただける話題を選んだつもりです。では行きましょう！

1.2　Brainf*ck って何？

Brainf*ck はプログラミング言語の 1 つで、名前の後半が卑語であることからチョメチョメ表記をされていることが多いです。そして Brainf*ck の大きな特徴は、

「　8 種類の文字だけで書くことができる　」

ということです。そんなことある??　具体的には、Brainf*ck のプログラムは、<>+−.,[] の 8 文字で構成されます。それぞれの文字は次のような意味を持ちます。

- <> ：矢印を 1 つ左・右にずらす。
- +− ：矢印があるマスに書かれた数を 1 だけ増やす・減らす。
- . ：矢印があるマスに書かれた数に対応する文字を表示する。
- , ：矢印があるマスに、入力された文字を書き込む。
- [：矢印があるマスに書かれた数が 0 なら，対応する] まで読み飛ばす。
-] ：矢印があるマスに書かれた数が 0 でないなら，対応する [まで戻って読み返す。

ここで、「矢印」と「マス」について説明しておきましょう。Brainf*ck のプログラムを動かすときには、（ほぼ）無限に一列に並んでいる “0” と書かれたマス目と、矢印 1 つを用意します（図 1.1）。

これを元に上で書いたような操作を行っていきます。では具体例を見ていきましょう！

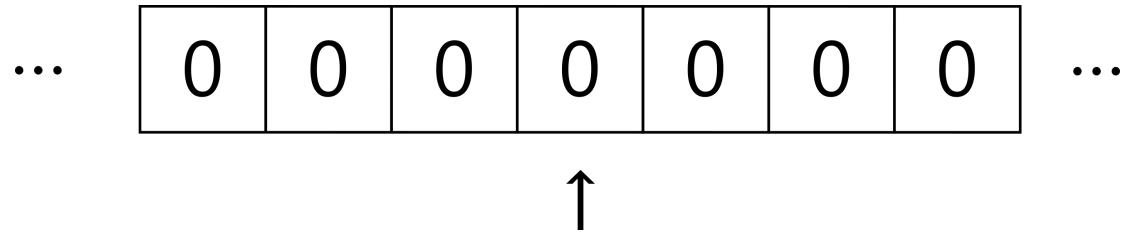


図 1.1　マス目と矢印

1.3　ゲームみたいなもの！

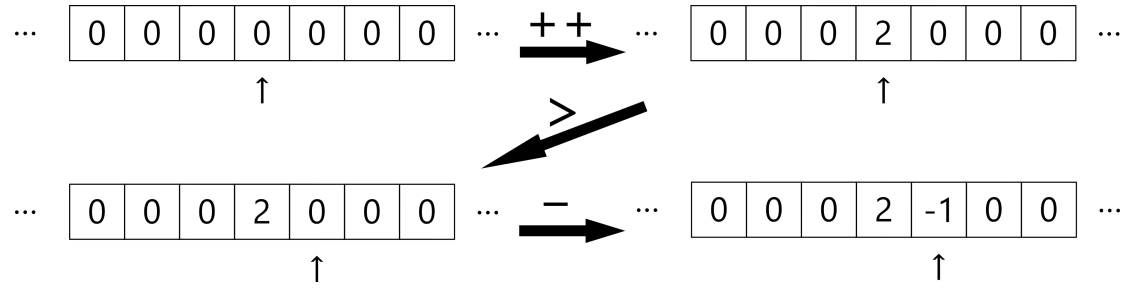


図 1.2　++>- を実行する

最初に、++>- という Brainf*ck のプログラムを実行してみましょう。すると、図 1.2 のように操作されます。この図を順番に追ってみてください。「矢印をずらす」、「書かれた数を変える」ということが理解できたでしょうか？

しかし、このままではマス目に書かれた数をいじることしかできませんね。もっと何かを表示したり、プログラミングらしいことがしたいです。そんな時に使うのが , と . です。これも例を見てみましょう！

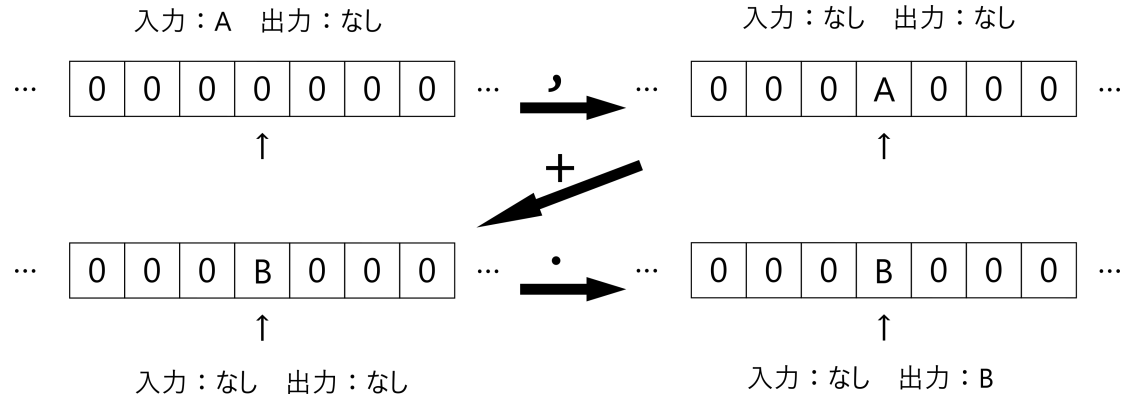


図 1.3　,+. を実行する

図 1.3 ではプログラム ,+. の過程を追っています。図の通り、これは「入力された文字の次の文字[*1]を出力するプログラム」になっています。

どうでしょうか？ マス目の中の矢印を操作していく感じは、ゲームに近いのではないでしょうか。次節では [と] を使ったもっと複雑なプログラムを見ていきましょう！

[*1] 正確には、マス目には ASCII コードという対応表によって文字を数字に変換したものが書き込まれています。ASCII コードで A は 65 に対応し、65+1 の 66 は B に対応するため、B が出力されるのです。

1.4　行ったり来たり

例えば，大文字の A を出力するプログラムを書いてみるとしましょう。前頁の脚注にも書きましたが，Brainf*ck で文字を出力するときは，ASCII コードという対応表によって対応する数を入れる必要があります。ASCII コードは Google で検索すればいくらでも出てくるので，ここでは割愛しますが，A は 65 に対応します。つまり，次のようなプログラムを書けば A と表示することができます。

```
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++
+++++.
```

つまり + を 65 個繰り返すことで，マス目に書かれた数を 65 にし，それを出力させたということです。確かにこれは正しい挙動をしますが、あまりにも冗長です。しかし、今知っている +-<>,. だけではこうとしか書きようがありませんね。そんな時に使うのが [] です。働きは最初のページで説明しましたが，具体例を見てみましょう！

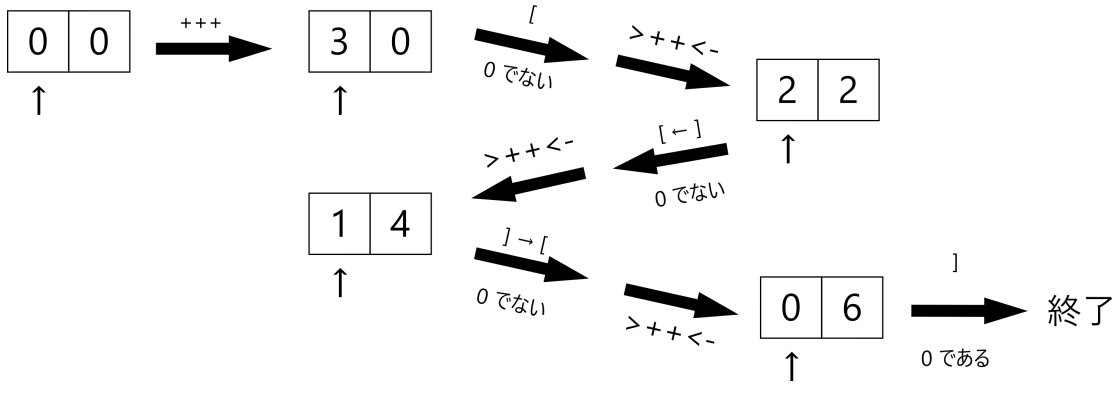


図 1.4　+++[>++<-] を実行する

上の図 1.4 で挙動を追ってみてください（結果的に 2 マスしか出てこないためそれ以外のマスを省略しています)。左のマスの 3 が 1 ずつ減っていくことで，3 回の繰り返しが生まれていることが分かるでしょうか？

[] を使うときはこのように、ループ構造を生むために使うことが多いです。では，これを使って再び A を出力するプログラムを書いてみましょう。次のような実装ができますね。

```
++++++++[>+++++++++<-]>+.
```

前半部分は図 1.4 とほぼ同じで，これによって $8 \times 8 = 64$ を 2 マス目に書き込んでいます。ループが終わった時点で矢印は 1 マス目ですから，>+ で 2 マス目を 65 にし，出力するという流れです。

1.5　おわりに

いかがでしたでしょうか。タイトル通り「あそんで」頂けたなら幸いです。Brainf*ck はあまり知られていない言語だと思いますが、考えるのが楽しい言語なので、ぜひ暇があれば皆さんもプログラムを作ってみてください！ ここまでお読みいただき、お疲れさまでした。ありがとうございました！

第 2 章

OpenStreetMap から人口を推測する

78 回生　stranger_86952

2.1　はじめに

「地図いじってなんかやりたい」って思ってたので、今回は地図上の情報からある程度可能そうな、人口予測に挑戦してみます。インド都市部などのググるだけでは古い統計しか得られない地域や、そもそも調べるだけでは人口がわからないエリア (灘校から半径 1km の範囲、など) もあるので、精度さえ出せればある程度需要もある技術だと思っています。

2.2　OpenStreetMap について

OpenStreetMap Wiki によると

> OpenStreetMap は、フリー、かつ編集可能な世界地図であり、数多くのボランティアによってゼロから作られ、オープンコンテンツのライセンスのもとで提供されています。

らしいです。地図版の Wikipedia みたいなかんじですね。今回は OverpassAPI を使って OpenStreetMap の情報を使います。
OverpassAPI では OpenStreetMap のほとんど全ての情報を入手できて、Manual にも載っていますが、空港や高速道路、地名などはもちろん、ATM の位置や氷河の割れ目など、本当に多岐にわたる情報が得られます。API は Overpass turbo で簡単に試すことができ、例えば以下のクエリで「灘校から半径 1km 以内にある駅の数」を得ることができます。

```
[out:json];
node(around:1000,34.71955, 135.26818)["railway"="station"];
out count;
```

このクエリだと住吉 (JR, 阪神, 六甲ライナー)、魚崎 (阪神, 六甲ライナー) が該当するので 5 と返ってきます。

2.3　どうやって人口を予測するのか

先に述べた通り、膨大な種類のデータを得ることができるため、建築物の分布やインフラの整備具合をAPI で取得し、線形回帰やランダムフォレストといった機械学習の手法を使って人口を予測するモデルを作ります。もちろんこれらの条件が同じでも人口は同じになるわけではないので、国や地域ごとにパラメーターを調整しないといけません。
例えば人口あたりの病院や駅の数は国ごとに異なるため、同じ病院、駅の数でも先進国と発展途上国なら人口がかけ離れている可能性があります。
そもそも OSM から得られる情報を元に直接導けるのはせいぜい世帯数だと考えられるため、一世帯あたりの人数が必要ですが、これは国・地域ごとに明らかに異なります。それらのパラメーターを調整するために、既存の人口データも用います。
次の章から、実際にこれらの要素を用いて人口予測を行い、どれくらいの精度が出るのか調べてみたいと思います。

2.4　日本国内で人口予測を試みる

まずは人口統計が充実していて、地方でも比較的インフラが整備されている日本国内で試します。OSM から学校やコンビニなどの数を取得した後 (いわゆる説明変数)、以下のようなデータセットを大量に用意し、人口を求めるモデルを Python で作ります。ちなみに、学習時に人口データが必要なので総務省が公開している統計をダウンロードして良い感じに使える形に処理しないといけないのですが、"～市～区"や"～郡～町"などの例外処理がちょっとだけ面倒でした。なお OverpassAPI の使用上、同一名称の市町村を区別することが難しいため、今回は同名の市町村および北方地域の自治体を除外しています。
※実際のデータは CSV ですが、収まりきらないためここでは JSON に変換しています。

ソースコード 2.1　Example JSON

```
{
 & "city": "札幌市",
 & "level": 7,
 & "population": 1959512,
 & "households": 1096729,
 & "school": 410,
 & "college": 59,
 & "hospital": 184,
 & "doctors": 472,
 & "clinic": 63,
 & "dentist": 485,
 & "restaurant": 1085,
 & "fast_food": 274,
 & "supermarket": 292,
 & "convenience": 960,
 & "park": 2458,
 & "platform": 3709,
 & "station": 75
}
```

このデータのうち city は市区町村名、population と households が予測する値、school より下が予測に用いる値です。
Python の sklearn ライブラリで簡単に試せる手法として、線形回帰、勾配ブースティング木、パーセプトロン、などさまざまなものがありますが、どれが最適なのかわからないのでとりあえず回帰と木を試し

てみます。少なくともニューラルネットワークは適切じゃなさそうだったので排除しました。コードはほとんど変わらないので、線形回帰の場合だけ以下に載せます。

ソースコード 2.2　Linear.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

drop_columns = ['city', 'level', 'population', 'households']

train_data = pd.read_csv('../data/learn.csv')
X_train = train_data.drop(columns=drop_columns)
Y_train = train_data[drop_columns[2:4]]

test_data = pd.read_csv('../data/test-input.csv')
X_test = test_data.drop(columns=drop_columns)

Model = LinearRegression()

Model.fit(X_train, Y_train)
Predictions = Model.predict(X_test)

test_data['population'] = Predictions[:, 0].astype(int)
test_data['households'] = Predictions[:, 1].astype(int)
test_data.to_csv('../data/test-output.csv', index=False)
for i, row in test_data.iterrows():
    print(row.iloc[0],row.iloc[2],row.iloc[3])
```

ランダムに 10 市区町村をピックアップし、残りの市区町村からランダムに選ばれた 10,100,1000 個のデータを学習させ、ピックアップしたものを予測させた結果がそれぞれ以下の通りになります。
※ここでは世帯数の予測については結果を省略しています。

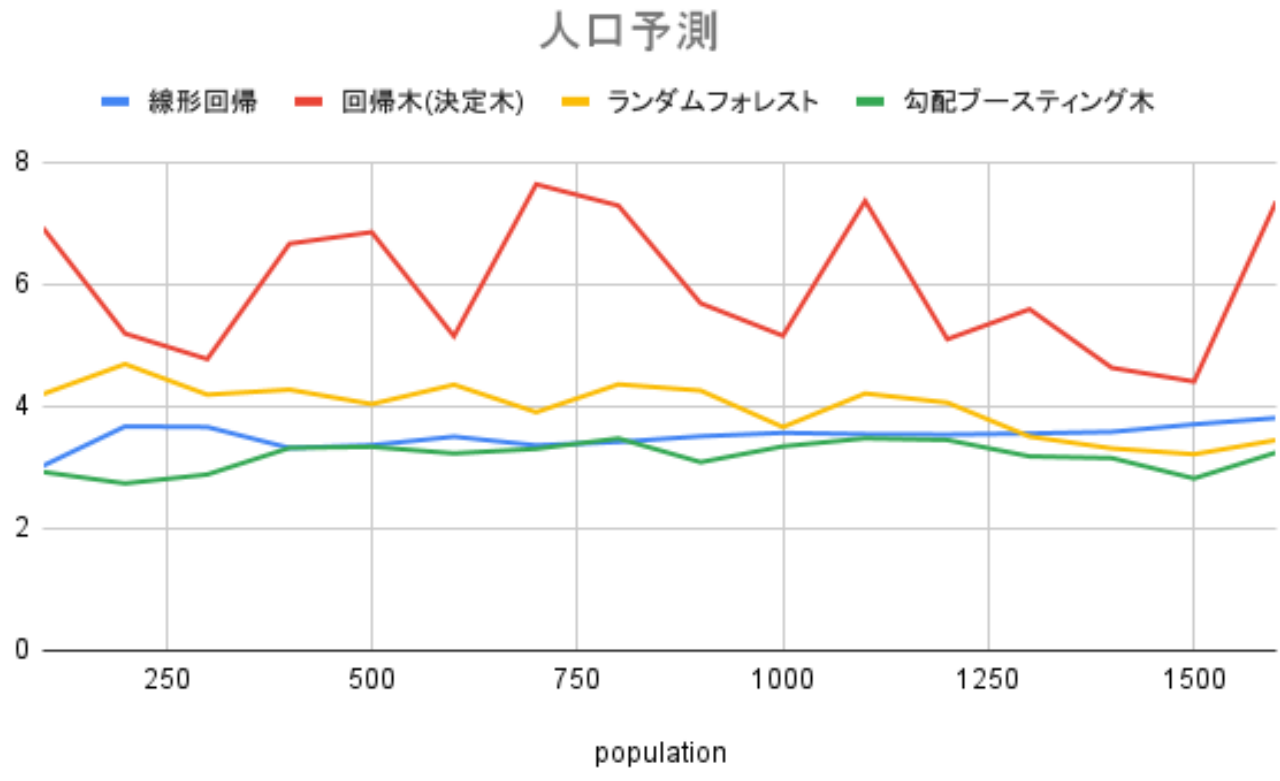
線形回帰	実際の人口	10	100	1000
玉東町	5241	1113	-1640	-3147
門真市	117937	-20453	95077	97586
山陽小野田市	60209	65287	56634	66631
宮代町	33514	26152	25023	24197
留萌市	19234	35599	21203	18606
大津市	344552	142351	216066	237161
砥部町	20510	14172	17633	9370
栄村	1642	7272	916	942
美波町	6071	17680	17290	17541
みなべ町	11988	10799	17628	17655

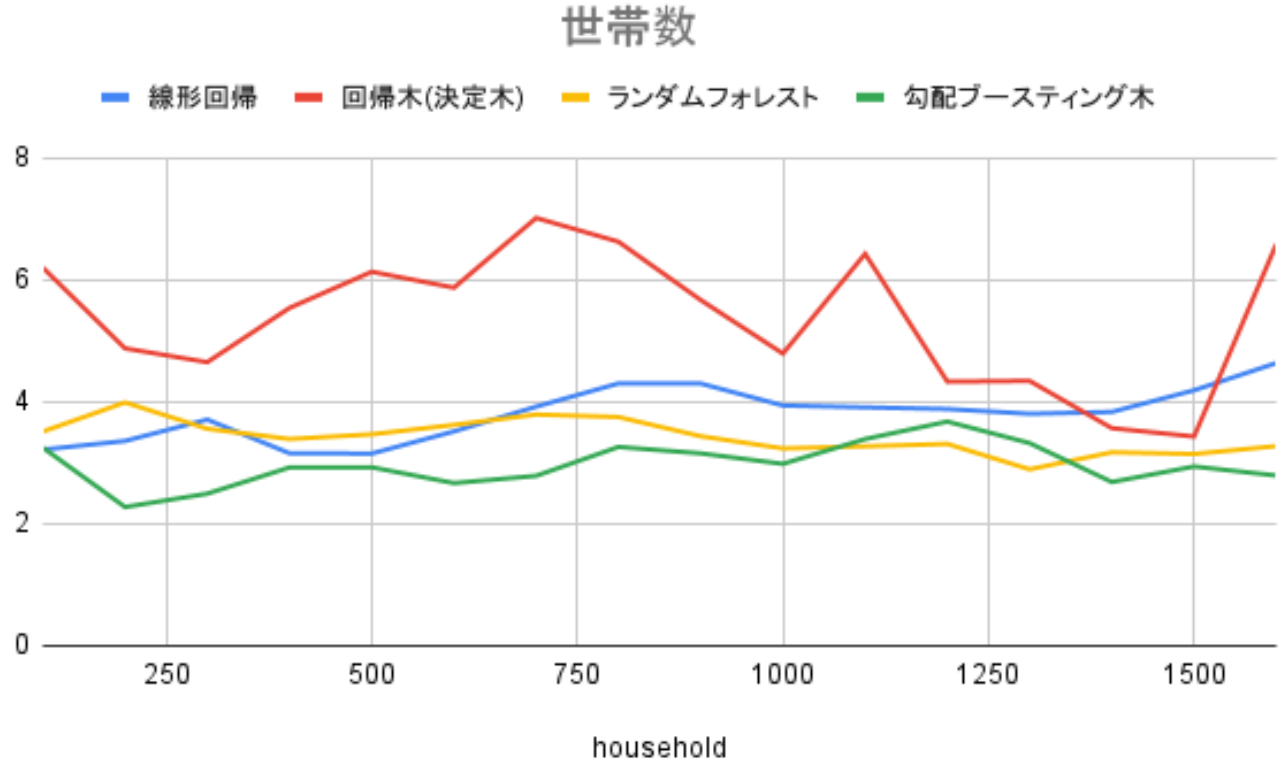
回帰木 (決定木)	実際の人口	10	100	1000
玉東町	5241	13813	1106	1159
門真市	117937	71296	121770	114259
山陽小野田市	60209	31644	27941	46328
宮代町	33514	18611	18611	18511
留萌市	19234	18611	30658	34811
大津市	344552	63299	270085	256005
砥部町	20510	49530	6017	6420
栄村	1642	13813	1075	2953
美波町	6071	31644	21513	5282
みなべ町	11988	12796	7906	5883

まず線形回帰について、人口の予測で負の値がでているのはかなりひどいです。学習データが 10 個の場合はほとんど当てにならない精度をしていますが、100,1000 個の場合についてはかなり実際のデータと近いものも存在します。ただ、この 2 つの精度について大きな差があるようには感じません。
回帰木 (決定木) についても学習量が 10 この場合はかなり的外れな予測が多いように見えます。線形回帰とは違ってちゃんとすべての予測が正の数になっています。
次に、予測の精度を測ってみたいと思います。単純に人口の差をとってしまうと、大都市であれば無条件に値が大きくなってしまうので、以下の数式で定めたいと思います。
※ただし、N はサンプル数、A を統計上の人口、B を予測した人口とします。

$$\frac{1}{N}\sum_{i=0}^{N}\frac{|A-B|}{A}$$

0 に近ければ精度が良く、元の人口の 2 倍以上の値を予測すれば 1 を超えることもあり得ます。
適切な学習量とモデルを探るため、学習データを 100 1600 市区町村まで 100 ずつ増やしながらそれぞれ精度をグラフにすると、以下の通りになります。





人口、世帯数両方に共通することとして、回帰木はかなり精度が低く、勾配ブースティング木の精度が比較的高いことがわかります。また、世帯数の予測の方が精度が高く見えます。学習量についてはあまり違いが見られませんでした。

2.5　世界の他地域でも予測できるのか

国連が公開している人口 10 万人以上の都市の統計を利用して、今のモデルが世界の他地域に適用できるか試してみます。

ランダムフォレスト	実際の人口	10	100	1000
Saudi Arabia - Jeddah	3430697	125943	487349	616187
Botswana - Francistown	103422	27105	33455	15225
New Zealand - Christchurch	392100	170602	561612	479461
Peru - Talara	102355	58940	59620	84279
Italy - Messina	221788	172539	545927	959142
Russia - Bratsk	242604	81483	138756	180935
Morocco - Ouarzazate	131103	29025	23680	29184
India - Cuttack	610189	75156	144407	92001
Bolivia - Oruro	216724	87998	67434	93057
United States - Billings	119960	109210	201819	154315

勾配ブースティング木	実際の人口	10	100	1000
Saudi Arabia - Jeddah	3430697	124591	423063	650126
Botswana - Francistown	103422	20557	28452	22567
New Zealand - Christchurch	392100	205298	554581	405102
Peru - Talara	102355	69435	29574	90102
Italy - Messina	221788	200102	412702	896973
Russia - Bratsk	242604	74619	171280	190775
Morocco - Ouarzazate	131103	24569	18894	18345
India - Cuttack	610189	62036	222939	86854
Bolivia - Oruro	216724	59413	52505	100622
United States - Billings	119960	140838	151548	141974

ぱっと見で国内より明らかに精度が落ちていることがわかりますが、実際に精度を測ってみると以下のようになります。

精度	10	100	1000
ランダムフォレスト	0.5915473522	0.7227790174	0.8134995744
勾配ブースティング木	0.5963988875	0.6393394947	0.7440039412

ある程度の精度が出ていた国内と違い、データ数が少ないほど精度が上がってることを考えると、国内のデータで作ったモデルは他地域でほとんど通用していないことがわかります。

2.6　まとめ

　いくつか反省点があります。
まず、説明変数の選び方が最適ではなかったと思います。一般的に互いに相関が強い説明変数を選ぶことは避けるべきですが、あまり考えずにやっていました。"school"と"college"、"hospital"と"doctors"と"clinic"と"dentist"など、まとめることができそうなものも多かったです。
他にも、今回扱った変数だけでは、人口が違う自治体でも完全に一致してしまうことがあり、他の変数を追加するべきでした。学習データについても、世界の他都市に応用させるなら国内だけのデータで学習させるのはよくなかったです。
ただ、国内については一定の精度で人口が予測できたので、概算が目的であれば使えるかもしれません。
最後に学習データについて、総務省の Excel をベースに OverpassAPI を Python で叩きながら集めたのですが、クエリが多すぎてほぼ丸一日かかってたみたいなので、使いたい方がいれば下に配布しておきます。2024 年 4 月時点でのデータなのでご了承ください。
https://github.com/stranger86952/Japanese-Data-By-Municipalities

第 3 章

簡易的なブラウザを自作する

78 回生　秋葉千歳

3.1　はじめに

はじめまして、78 回生の秋葉千歳 (TitoseDot) です。普段はウェブサイト/ゲームを作っています。好きな言語は Java、TypeScript です (名刺交換)。最近、HTML 仕様を読んでいて自分でもブラウザを作ってみたいなと思ったので、この機会に実装してみることにしました。この記事では、HTML 仕様書 (HTML Living Standard) にやや準拠したオレオレブラウザを作っていこうと思います。

3.2　パーサー

3.2.1　パーサーの概要

私達が普段何気なく使用している Web ブラウザですが、裏では結構複雑な工程を踏んで動作しています。そのうちの一つがパーサーです。パーサーは、インターネットやファイルなどから受け取ったデータを意味のある構文として解釈するものです。具体的には以下の画像の流れで動作します。

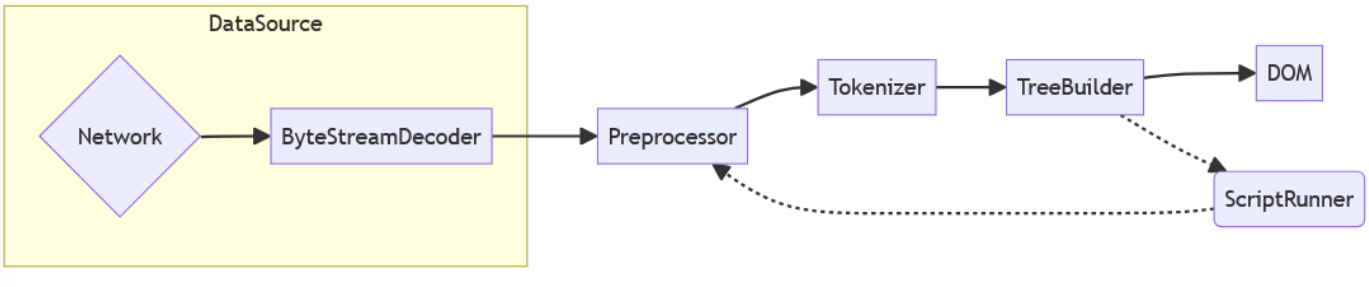


図 3.1　Parser の概要

3.2.2　ByteStreamDecoder の実装

> `HTML Living Standard 13.2.3 意訳`
> ネットワークやローカルファイルから一番最初に渡されるデータは、文字列を特定のエンコード規則に従ってエンコードした結果である Byte Stream(意味を持ったバイト列) です。
> `ByteStreamDecoder` は Tokenizer がデータを処理しやすくするために、`Byte Stream` を `Stream`

of Code Points(文字列) に変換します。

要は、ByteStreamDecoder は UTF-8 などのエンコードに従って文字列を復号化するものです。ご丁寧なことに、HTML Living Standard には ByteStreamDecoder の動作も規定されているので、概ねその通りに実装していきます。

例えば、BOM 付き UTF-8 などが送られてきた場合には、BOM Sniff Algrithm に従ってデコード形式を決定するように規定されています。デコードされたバイト列は、処理系が使用するエンコード規則の文字列に変換されるのが一般的です (UTF-8 など)。今回の実装でも、内部では UTF-8 として文字列を扱うので、入力された文字列は UTF-8 に変換されます。

3.2.3 Tokenizer の実装

この時点で既に発狂しそうなのですが、次は Tokenizer の実装をしていきます。

> HTML Living Standard 13.2.5 意訳
> Tokenizer の実装は、以下に述べるステートマシンを利用したように動作するものとする。
> - Data state から開始する。
> - 大部分の state は入力文字列を 1 つ消費 (読み取り) するが、副作用を持つものもある。
> - 別の state に変更し、(現在の文字を再消費/次の文字を消費) する場合と同じ state のまま次の文字を消費する場合がある。
> - 一部の state は別の state に変更する前に何個かの入力文字を消費する場合がある。
> - 一部の場合に限り、Tree Construction の動作中に Tokenizer の state を変更する場合がある。
> - etc...

要は、Tokenizer の仕様は有限オートマトンを用いて表されており、実装はステートマシンを用いて行ってくださいということですね。ご丁寧なことに、HTML Living Standard には 80 種類ほどの state がすべて列挙されており (血涙)、これを忠実に実装するだけで Tokenizer としての動作は網羅しているということになります。これらの要件を満たすためには、Tokenizer は構造体、クラスに準ずるものとして実装し、最低限

- state
- state の戻り状態

を保持する必要があります。HTML 仕様の有名な Rust 実装である、html5ever は他にも

- Tokenizer が受け取ったオプション
- Tokenizer が発行したトークンを返すための Sink
- EOF に到達したかのフラグ
- 現在の文字列 (再消費用)
- 再消費するかのフラグ
- LF 改行を無視するかのフラグ
- BOM を確認するかのフラグ
- 現在のタグの種類

- 現在のタグの名前
- etc. . .

を保持しており、Tokenizer 主体の設計になっていることが伺えます。しかし、今回の実装においては Tokenizer が肥大化することを避けるため、Tokenizer は各種ステートを保持、呼び出しを行い、各種ステートに処理を記述していく形にします。

3.2.4 TreeBuilder の実装

無事に魂が抜けたところで次は TreeBuilder の実装を行います。
TreeBuilder の実装に関しては概要をまとめることすら困難なため、簡単な説明だけ行います。

> TreeBuilder は、Tokenizer が解析、発行したトークンを受け取り、階層化された木へと変換します。

マジの意訳をすると Tokenizer によって文節に区切り、TreeBuilder によって機械が扱いやすい構造体に変換します。仕様上だと Tree Construction Stage として分離されていますが、実際の実装では Tokenizer と融合合体している場合も多いです。また、TreeBuilder は本来、<script>タグを見つけた場合 JavaScript エンジンを用いてそれを実行するとされているのですが、JavaScript にまで対応していると~~部誌の締切に間に合わなくなる~~寿命が終わるため、今回は泣く泣く対応を見送りました。

3.3 描画

本来であれば、CSS への対応を行わなければならないのですが、CSS に対応していると太陽系が滅ぶまでの時間がかかってしまうので、今回はスタイルを Renderer に埋め込んで描画を行います。Renderer の実装ですが、今回は OpenGL を用いてインタプリター的な動作を行うようにします。モダンブラウザだとレイヤーで描画結果を保持し、それぞれのレイヤーを合成して描画を行う Composition などと呼ばれるような最適化を行っているのですが、今回は描画できるだけで ok ということにします。Tree Construction Stage にて構造化されているため扱いやすいデータとなっているので、再帰的に構造体を探索し、情報通りに描画していけばヨシ!です。

~~Rust での実装が期限までに完成していないので、代わりに JavaScript の概念コードを載せておきます…~~

```
function rendering(dom) {
    if (dom.hasChild()) {
        square(dom.x, dom.y, dom.w, dom.h);
        for (const child of dom.getChildren()) {
            rendering(child);
        }
    } else {
        text(dom.content);
    }
}
```

最適化を考えないなら、かなり自明な実装を行うことができます。実際にはここに CSS などが入ってくるため、見るだけで目がチカチカする実装が必要になります。

3.4 さいごに

Too Long なクソ記事、味わっていただけましたか?おそらく目が滑りに滑って全く話が入ってこない

かと思います。今回はこんなバカみたいな題材を選んでしまってかなり後悔したわけですが、次回はもうちょっとアイデアよりの部誌を書きたいなと思っています。ここまでの閲覧、誠にありがとうございました。是非チャンネル登録、いいね、通知もお願いします。

第 4 章

CyberSakura 第 3 回出場レポ

77 回生　Sug

4.1　はじめに

数年ぶりにコード組んで意外と楽しいな、となった文化祭準備の今日この頃。77 回生の Sug です。よろしくお願いします。コードを組まないパソコンの大会に出たので、それについて書いていこうと思います。

4.2　CyberSakura とは？

「CyberSakura は、2021 年にスタートした日本の未来のサイバーセキュリティ分野を担う人材の育成を目的とした教育プログラムです。日本国内の中学生・高校生・高専生（1〜3 年）を対象に、競技会を開催し、仮想環境でセキュリティの脆弱性を発見・修正すること等で獲得するスコアを競います。競技会前には、自学習と練習の期間を設けます。

具体的には、オンライン形式で実施される予選までの学習と競技、最後に福井県鯖江市で行われる決勝があります。これらを通して、コンピューター・ネットワーク・セキュリティの知識を実践的に学習し、制限時間内にチームで課題を解決する能力を育みます」

`https://cybersakura.jp/whatis`

といってもどういう競技かよくわかりませんね。わかりやすく言うと、「とある会社のパソコンのセキュリティを強くするという名目で、社員の弱いパスワードを直したり会社に認められていないアプリを消したりしてポイントを得る」競技です。

これの対象となっている OS は Windows, Ubuntu, Windows Server があります。この OS の仮想環境 (VM) や問題を作っているのは CyberPatriot という外国の会社なので、問題文も当然英語です。ハァ。

4.3　予選

というわけで同級生のもちもち君に誘われてやることもなかったので、僕、もちもち、きゅうたの 3 人でチーム「京都人の一日」で出ることになりました。予選ラウンドは Windows, Ubuntu の 2 個の OS でそれぞれ 100 点ずつ、計 200 点での勝負となります。

予選はずっと僕ときゅうたが Windows、もちもちが Ubuntu を担当していました。VM を開くとデス

クトップに README, Forensic Question1, 2 というファイルがあります。READ ME には英語でストーリーが書いてあり、ストーリーでは企業が〜〜したいと思っている、社員のパスワードが弱い、ユーザーのグループを管理してほしいみたいなことが書いてあるので、それをやると点数が入ります。

予選の一か月前に練習ラウンドといい予選には関係ないけど練習ができる機会があり、そこで出た問題と READ ME, Forensic Question1, 2 を解き Windows で 90 点を取りました。一方、もちもちが Linux で 48 点を取り 138 点で 3 位。結果として福井に行けることになりました。

4.4 決勝

決勝ラウンドは Windows, Ubuntu, Windows Server の 3 個の OS でそれぞれ 100 点ずつ、計 300 点での勝負となります。

さて、決勝へ行く...1 か月前にもちもちから連絡が。

「ごめんフライトの時間が調整つかなくて cybersakura いけなくなった」

ワァ...3 人から 2 人へとなりました。ん？ OS の数より少なくない？ 何か変じゃない？ (察しが悪い)

そして、本番当日。3/24 に福井県鯖江市へと向かいました。ちなみに交通費と宿泊費は無料です。福井について、いただいた弁当を食べて開会式を行い早速競技となります。競技時間は 4 時間です。

最初は僕が Windows Server、きゅうたが Ubuntu を担当していました。予選とは違い、決勝では READ ME に書いてない＆予選/練習ラウンドの過去問に無くても直さないといけないところが同じくらいの点数分あります。Forensic Question は READ ME とは関係なく、このファイルが通信しているポートは何？ などの問題を解けば点数が入ります。CTF っぽいです。

3 時間かけて Windows Server で Forensic Question を 2 問解き、予選の時に直すべきだったところを確認して直し 32 点取った後、Ubuntu へと移動しました。Ubuntu では残り 1 時間ほどしかなく、またあまりコマンドに慣れていなかったのもあり Forensic Questions と認証されてないユーザー消すのとアプリ消すので 14 点取って終了。Windows Server も Ubuntu ももうちょっと勉強しておけば... ってのと Linux はそもそも 1 時間で解くのが難しいのがあり何とも言えない感じです。

一方で、きゅうたの方は Windows に 4 時間かけて 36 点取っていました。過去にやったところを直してそこからが進まなかったそうです。実際、Google Chrome のアップデートを全部のアカウント (15 個くらいある) で行い既定のアプリにするってのを地道にやっても入らなかったりするし厳しいな〜ってなってました。

結局合計 14+36+32=82 点で 5 位でした。他のチーム、実は 4 人いるのがデフォルトらしく半分しかいない中ではよくやれたといえるのかもしれないですが、勉強不足もあったので微妙です。いや休みがいなかったら 3,4 位が同率 139 点なので頑張れば 3 位にはなれそうでしたけど。

4.5 おわりに

今回の部誌は画像で説明できるところが少なくあいまいな記事になってしまい申し訳ありません。なので、CyberSakura, Cyber Patriot の公式サイトを見ていただけると幸いです。リンクは

```
https://cybersakura.jp/
https://www.uscyberpatriot.org/home
```

こちらになります。また、ニュースとなりましたのでこちらも見ていただけると幸いです。

https://www3.nhk.or.jp/lnews/fukui/20240324/3050017397.html

第 5 章

プリプロセッサで遊ぶ

*

79 回生　CATCAT

5.1　はじめに

79 回生の CATCAT です。もう高 1 にもなるのにまだ部誌を書いたことがないどころかゲーム制作以外何かをやった記憶がないので書きます。

5.2　プリプロセスとは

プリプロセスというのはいわゆる前処理のことで、C 言語などにおけるソースコードを文字列として扱う機能のことです。#define や#include などが主な例です。主な構文としては

```
#define A B
#define F(x) 1+x
A//   ①
F(3)//   ②
```

こうすることで①は B に展開され、①は 1+3 に展開されます。ここで重要なのは、プリプロセッサはあくまでマクロであり、計算をしてくれません (一部例外あり)。F(3) を展開した結果は 1+3 であって 4 ではないことに注意が必要です。今回はほぼこれだけを使ってプログラミングをします (は?って思ってる人は正しいですたぶん)。

5.3　Boost Preprocessor を使ってみる

boost とは C++ 標準化委員会の委員により設立された多くの機能を備えた C++ ライブラリです。便利なライブラリがたくさんありますが今回はその中でも Boost.Preprocessor に焦点を当てていきます。このライブラリは上に書いたようなマクロを利用して四則演算や配列や条件分岐、ループなどを定義したすごいものです。

5.4　とりあえず書いてみる

例えば九九の表を作ってみます

```
#include<iostream>
#include<boost/preprocessor.hpp>
#define MUL1(z,c,d) BOOST_PP_REPEAT(9,MUL2,BOOST_PP_INC(c))
#define MUL2(z,c,d) BOOST_PP_MUL(BOOST_PP_INC(c),d)
int main(){
    std::cout<<BOOST_PP_STRINGIZE(BOOST_PP_REPEAT(9,MUL1,_))<<std::endl;
}
//のは使えなかったのでここだけwandboxCPPBoostCです++
```

このコードはプリプロセスの段階で

```
int main(){
    std::cout<<"1␣2␣3␣4␣...␣9
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣2␣4␣6␣8␣...␣18
                  :             :
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:             :
                 8 16 24 32 ... 72
                 9 18 27 36 ... 81"<<std::endl;
}
```

のように展開され、ソースコードに埋め込まれます。

5.5 BOOST_PP_STRINGIZE

プリプロセッサの関数型マクロには#記号というものがあり、例えば#hoge などとすると、hoge を文字列化することができます。そして、上のコードに出てきた BOOST_PP_STRINGIZE はこのような実装になっています。

```
#define BOOST_PP_STRINGIZE(text)
BOOST_PP_STRINGIZE_I(text)
#define BOOST_PP_STRINGIZE_I(text) #text
```

なぜこのような遠回しな実装がされているかというのは次のコードを実行するとわかります。

```
//上のコードの続きから
#define MY_STRINGIZE(text) #text
#define TABLESIZE 1024
MY_STRINGIZE(TABLESIZE)//  ③
BOOST_PP_STRINGIZE(TABLESIZE)//  ④
```

このコードを処理させると、③は TABLESIZE と展開され、④は 1024 と展開されます。プリプロセッサは#や##(連結記号) などを優先的に処理して、そのあとに引数のマクロ展開を行います。ただ、BOOST_PP_STRINGIZE の場合 BOOST_PP_STRINGIZE_I に展開されるときに引数を完全にマクロ展開するため、このような結果の違いが生まれます。

5.6 BOOST_PP_REPEAT

普通のプリプロセッサマクロは再帰的に展開できない仕様になっています。

```
#define foo 4+(foo)
foo
```

例えば、この場合の foo は 4+(foo) と展開され、() の中の foo はこれ以上展開されません。しかし、前半のコードの中では BOOST_PP_REPEAT の中で BOOST_PP_REPEAT が使われています。これは BOOST_PP_REPEAT の内部で再帰的にならないようにある工夫がされているからです。

5.7 参考

BOOST_PP_REPEAT の仕組み

https://zenn.dev/hikarin/articles/5329647ec2a542653f68

The C Preprocessor

https://gcc.gnu.org/onlinedocs/gcc-7.2.0/cpp/Self-Referential-Macros.html#Self-Referential-Macros

https://gcc.gnu.org/onlinedocs/gcc-7.2.0/cpp/Argument-Prescan.html#Argument-Prescan

字句の結合や文字列化を行う際のマクロ置換動作をよく理解する

https://www.jpcert.or.jp/sc-rules/c-pre05-c.html

第 6 章

灘中入試算数はプログラミングで倒す

79 回生　Cofey

6.1　はじめに

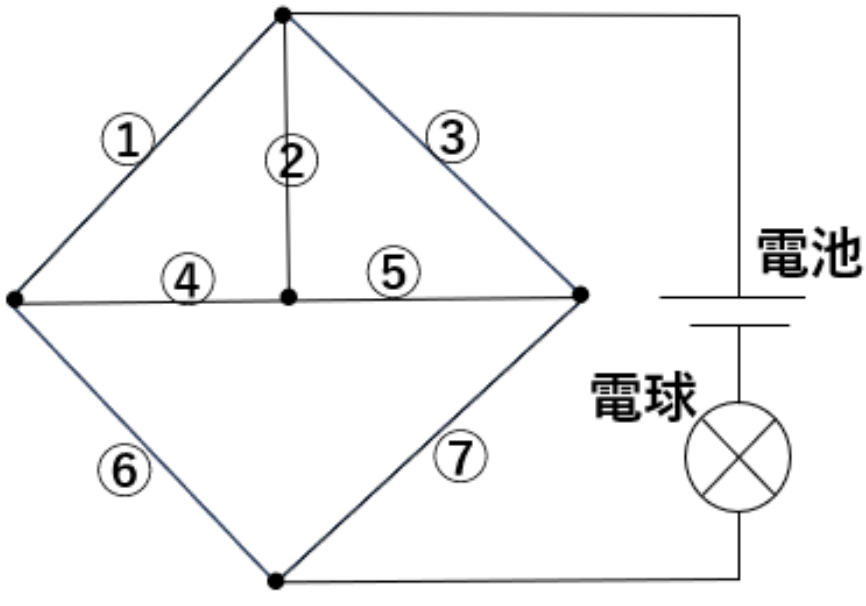
こんにちは。高１の Cofey と申します。普段は灘校生らしく勉強を頑張っています（の予定です）。ついこないだ中学入学してためになったね～とか言ってた気がしたらもう高校生になってしまいました。光陰猶矢。

さて、今回は２０２４年度灘中入試算数１日目の大問７を解いていきたいと思います。まだ解いていない方にはネタバレになるかもしれませんので、ご注意ください。

6.2　問題

問題文は次のようなものでした。

> 下の図のような電池１個、電球１個、１～７のスイッチ７個を含む電気回路がある。電球が点灯するようなスイッチのオン・オフの仕方は何通りか。



あまりやりたくない場合の数の問題ですが、解いていきましょう。

6.3 解く

スイッチ６と７のオン・オフの仕方４種類による場合分けで解きます。

- ６と７の両方がオンのとき
  電球がつかないのはオンのスイッチの組み合わせが２、４、５、４５、そしてすべてオフのときの５通りです。よって、この場合は $32-5=27$ 通りとなります。
- ６と７のうち片方のみがオンのとき
  対称性より、６のみがオンの場合を考えてそれを２倍すればよいです。１がオンの１６通りでは、電球は必ずつきます。１がオフのときは電気は２→４→６と通るか、もしくは３→５→４→６と通ることができれば電球がつきます。よって、１がオフのときに電球がつくのはオンのスイッチの組み合わせが２４、２３４、２４５、３４５、２３４５の５通りです。以上より、６のみがオンのとき電球がつくのは $16+5=21$ 通りとなり、７のみがオンのときも同じく 21 通りです。

６と７の両方がオフのときは、必ず電球はつきませんから、答えは $27+21\times2+0=69$ 通りとなります。

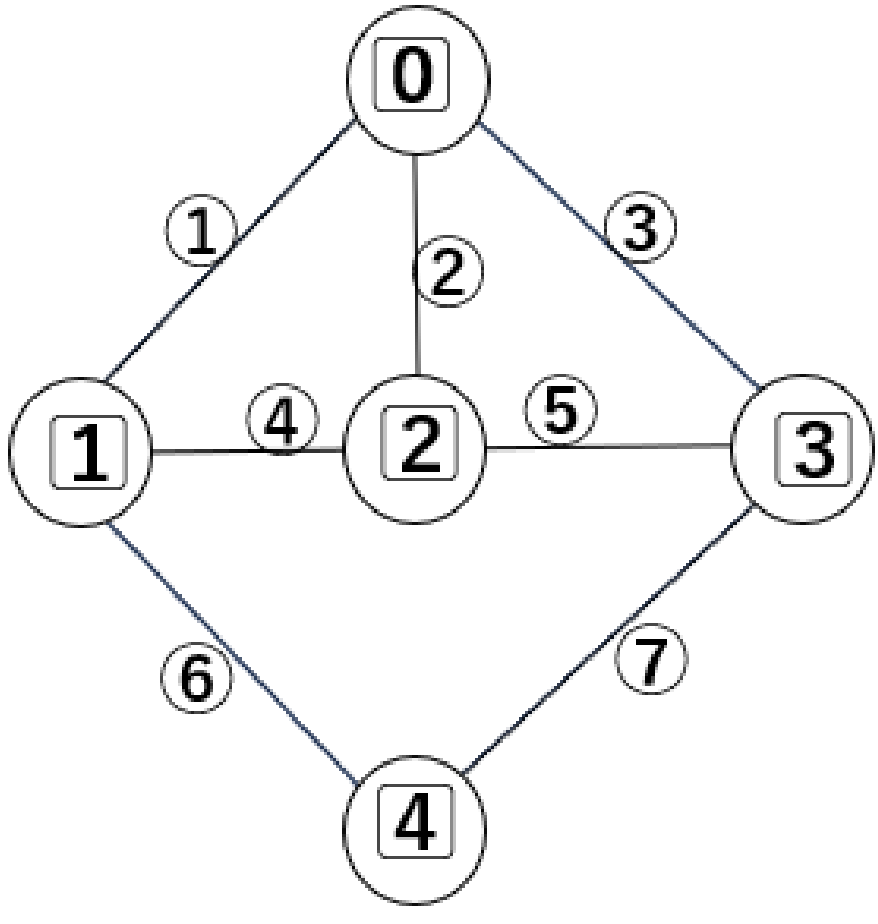
6.4 おわりに

と、このように面倒な場合分けが必要となり、時間をかけて丁寧に解けば正解はできますが単純に面倒です。

そこで、プログラミングしましょう！

この図は、下の図のように０～４の５つの頂点を持つグラフで、スイッチがオンのときだけその部分に辺を張るというふうに見ることができます。この図では、頂点０から頂点４にたどり着けるということが電球がつくことを意味します。

そして、スイッチのオン・オフの仕方１２８通りすべてについて頂点０から頂点４にたどり着けるか判定するプログラムを書けば、正しい答え、６９を導き出すことができます。


0
1
2
3
4
5
6
7
1
2
3
4

第 7 章

DirectX を使ってみる

79 回生 かしは

7.1　はじめに

初めまして。パソコン部に入ってるのか入ってないのかのような生活をしていました 79 のかしはです。当然ながら初執筆です。もう高 1 というのに NPCA 問わず部誌書くの今年が初めてなんですね。よくないです。

みなさんは DirectX をご存じでしょうか？ DirectX とは Microsoft 社が開発したゲーム開発用の API 群のことです。現在最新版は DirectX12 であり、今も使われています。DirectX の中にもさまざまな機能があります。例えば　、Direct3D、DirectInput(廃止?)、DirectWrite などです。しかし、一番よく使うのが Direct3D なので実質それが DirectX 扱いされていることもあります。

今回は DirectX でゲームか何かを作っていきたいなと思います。なお自分は初心者なので DirectX9 という比較的簡単なバージョンの DirectX で作っていきます。C++ を使います。DirectX12 は難しい事で知られており、自分のような初心者には厳しいからです。初心者なのでどこか情報が違っていたら申し訳ないです。

7.2　予備知識～Win32API～

Windows API について説明しておきます。一般的に Windows のアプリは図 1-1 のような構造をしています。

プログラム内でメインループというイベントの監視をするループがあります。ユーザーがキーボードを触ったり、画面のボタンを押したりというイベントがあるとこのメインループの中でプログラムがメッセージを受け取り、それをウィンドウプロシージャというイベント時に動く関数に渡します。具体的にソースコードを見てみましょう。

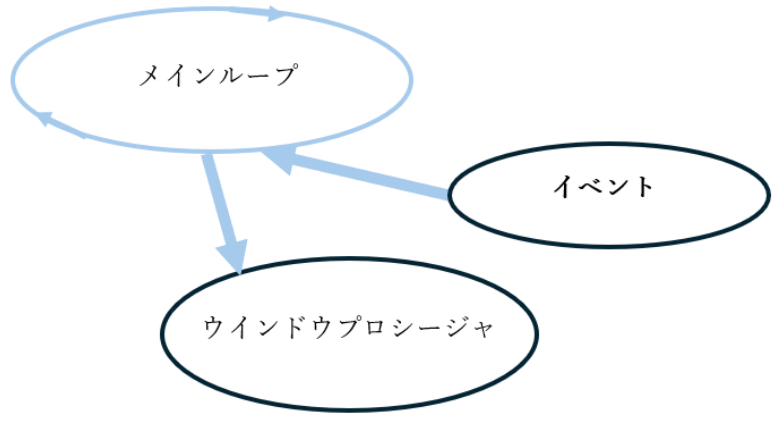


図 7.1　図解

ソースコード 7.1　Win32API

```
#include <windows.h>
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
}
```

基本的にこの {} の中にいろいろな処理を書いていきます。この wWinMain 関数が最初に呼ばれる関数でつまり、その中の処理が最初に実行されます。

ソースコード 7.2　Win32API

```
WNDCLASSEX wcex;
    //構造体のサイズ
    wcex.cbSize = sizeof(WNDCLASSEX);
    //クラスのスタイル
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    //ウィンドウプロシージャを指定
    wcex.lpfnWndProc = WndProc;
    //補助メモリ
    wcex.cbClsExtra = 0;
    //補助メモリ
    wcex.cbWndExtra = 0;
    //インスタンスハンドル
    wcex.hInstance = hInstance;
    //アイコン
    wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
    //カーソル
    wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
    //背景ブラシ
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    //メニュー名
    wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_WINDOWSPROJECT1);
    //クラス名
    wcex.lpszClassName = szWindowClass;
    //小さいアイコン
    wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
    //登録
    if (!RegisterClassEx(&wcex))
    {
        MessageBox(NULL,
            _T("の処理に失敗しましたRegisterClassEx"),
            _T("Sample02"),
            NULL);

        return 1;
    }
```

まず、wWinMain 関数の中で最初にすることは WNDCLASSEX 構造体を宣言して登録することです。この構造体はアイコンやカーソルをどれにするかなど画面を作るための様々な情報が入っています。この構造体にはウィンドウプロシージャはどの関数にするのかという情報も含まれています。これを登録した後は CreateWindows 関数を使って、この構造体の情報をもとに画面を作ります。この時、ウィンドウハンドルという PC が指定している画面の管理番号も作られます。

ソースコード 7.3　Win32API

```
//がウィンドウハンドルHWND
HWND hWnd = CreateWindow(
        szWindowClass,
        szTitle,
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        WINDOW_WIDTH, WINDOW_HEIGHT,
        NULL,
        NULL,
        hInstance,
        NULL
        );
    // ウィンドウが生成できなかった場合
```

```
    if (!hWnd)
    {
        MessageBox(NULL,
            _T("ウィンドウ生成に失敗しました!"),
            _T("Sample02"),
            NULL);
        return 1;
    }
```

CreateWindow した後は ShowWindow 関数、UpdateWindow 関数を実行します。

ソースコード 7.4　Win32API

```
    //は関数の第４引数nCmdShowwWinMain
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
```

そしてメインループを書いて wWinMain 関数に書くことはもう終わりですね。

ソースコード 7.5　Win32API

```
    //メインループ で繰り返しwhile
    //→からのメッセージを取得GetMessageOS
    //→メッセージを翻訳TranslateMessage
    //→メッセージをウィンドウプロシージャに渡すDispatchMessage
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (int)msg.wParam;
```

次はウィンドウプロシージャについて見ていきましょう。case WM_DESTROY とはウィンドウを閉じろというメッセージが来たとき：という意味です。そうしたら PostQuitMessage で画面を閉じます。

ソースコード 7.6　Win32API

```
// ウィンドウプロシージャ
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
        break;
    }

    return 0;
}
```

まとめると画面は WNDCLASSEX 構造体作成→ CreateWindow → ShowWindow → UpdateWindow の順に作っていきます。

7.3　予備知識～Direct3D～

Direct3D の仕組みについて説明します。描画を担うのは DirectGraphics です。

7.3.1　初期化

まず IDirect3D9 という DirectGraphics の大元のインターフェイスを作成します。この IDirect3D9 はマシーンの描画能力を調べることができます。そして Direct3D のデバイス、IDirect3DDevice9 を作成します。これがよく使うもので、実際にいろいろ描画するものです。

IDirect3D9 は Direct3DCreate9 関数で作成します。その後に D3D_PRESENT_PARAMETERS 構造体を作成し、この構造体を使って CreateDevice 関数で IDirect3DDevice9 を作成します。その次に ViewPort の設定をします。これはウインドウの中で Direct3D が描画するところの範囲を指定します。Direct3DDevice.SetViewPort(&D3DVIEWPORT9) 関数でそのデバイスにおけるビューポートを設定します。この D3DVIEWPORT9 は ViewPort の設定をする構造体です。

ソースコード 7.7　Direct3D

```
bool InitDirectX(HWND window_handle)
{
    // インターフェース作成  を作成IDirect3D9
    g_pD3DInterface = Direct3DCreate9(D3D_SDK_VERSION);
    if (g_pD3DInterface == NULL)
    {
        // 作成失敗
        return false;
    }
    //g_pD3DPresentParam という名前の型構造体を宣言D3DPRESENT_PARAMETERS
    g_pD3DPresentParam = new D3DPRESENT_PARAMETERS;
    if (g_pD3DPresentParam == NULL)
    {
        return false;
    }
    ZeroMemory(g_pD3DPresentParam, sizeof(D3DPRESENT_PARAMETERS));の要素を弄っている
    g_pD3DPresentParam
    // バックバッファの数 => 1
    g_pD3DPresentParam->BackBufferCount = 1;
    // バックバッファのフォーマット => D3DFMT_UNKNOWNフォーマットを知りません()
    g_pD3DPresentParam->BackBufferFormat = D3DFMT_UNKNOWN;
    // ウィンドウモード設定 => 定数で切り替え
    g_pD3DPresentParam->Windowed = true;

    // スワップエフェクト設定 => ディスプレイドライバ依存
    // スワップエフェクト => バックバッファとフロントバッファへの切り替え方法
    g_pD3DPresentParam->SwapEffect = D3DSWAPEFFECT_DISCARD;

    // の作成DirectDevice
    if (FAILED(g_pD3DInterface->CreateDevice(D3DADAPTER_DEFAULT,
        D3DDEVTYPE_HAL,
        window_handle,
        D3DCREATE_HARDWARE_VERTEXPROCESSING | D3DCREATE_MULTITHREADED,
        g_pD3DPresentParam,
        &g_pD3DDevice)))
    {
        return false;
    }

    // ビューポートパラメータ
    D3DVIEWPORT9 view_port;

    // ビューポートの左上座標
    view_port.X = 0;
    view_port.Y = 0;
    // ビューポートの幅
    view_port.Width = g_pD3DPresentParam->BackBufferWidth;
    // ビューポートの高さ
    view_port.Height = g_pD3DPresentParam->BackBufferHeight;
    // ビューポート深度設定
    view_port.MinZ = 0.0f;
    view_port.MaxZ = 1.0f;

    // ビューポート設定
    if (FAILED(g_pD3DDevice->SetViewport(&view_port)))
    {
        return false;
    }

    return true;
}
```

これで初期化は完了です。

7.3.2　描画

図 7.2　描画

この描画のステップはメインループ内で１秒に何回も繰り返す処理になります。バッファ (画面) にはフロントバッファとバックバッファがあって、フロントバッファで絵をユーザーに表示している間にバックバッファでプログラムが描画をします。そして描画し終わったらフロントバッファとバックバッファを入れ替えます。

描画は５ステップです。

1. まず Clear() で画面（バッファ）をまっさらにします。
2. 次に BeginScene() で描画の開始を知らせます。
3. それで描画して
4. EndScene() で描画を終了したことにします。
5. そして、バックバッファからフロントバッファに Present() 関数でデータを送ります。

ソースコード 7.8　Direct3D

```
void Draw()
{
//^^e2^^91^^a0
    g_pD3DDevice->Clear(0L,
        NULL,
        D3DCLEAR_TARGET,            // 初期化するバッファの種類
        D3DCOLOR_ARGB(255, 0, 0, 0),// フレームバッファの初期化色
        1.0f,                       // バッファの初期値Z
        0);                         // ステンシルバッファの初期値
//^^e2^^91^^a1
    g_pD3DDevice->BeginScene();
//^^e2^^91^^a2
    g_pD3DDevice->EndScene();
//^^e2^^91^^a3
    g_pD3DDevice->Present(NULL, NULL, NULL, NULL);

}
```

3. の描画処理の中身ではテクスチャ (画像) を読み込んで、テクスチャをセットして DrawPrimitiveUp 関数で描画をします。もしくは DrawSubset 関数を使う場合もあります。

7.3.3　行列変換

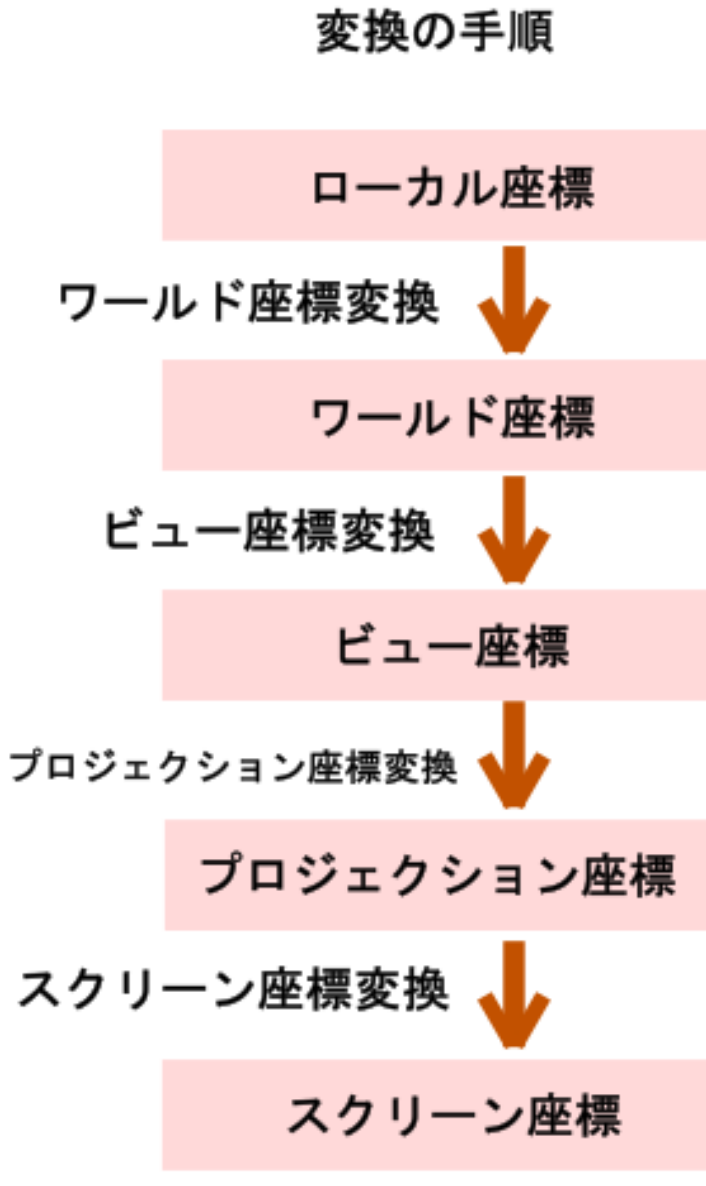


図 7.3　行列変換

具体的な先ほどの 3. の描画の中身で何をするのでしょうか。

まず、３ D において面の頂点の位置を決めます。決めたらその座標を 2D で表示しやすいように変換する必要があります。これがとても大変です。

３ D のプログラムを作成するときに行列を掛けることで座標を変換する必要があります。行列変換はワールド座標変換、ビュー座標変換、プロジェクション座標変換、スクリーン座標変換の順に行います。

最初ストラクチャはローカル座標で書かれています。これはストラクチャのある特定の点での座標を 0,0,0 とする座標表記の方法です。しかし、ほかのストラクチャも混ぜて設置する段階になったら、それだと困るので、ある絶対の一点を基準にして複数のストラクチャを配置します。これがワールド座標です。ワールド座標変換行列には、移動、回転、拡大の３種類があります。

ビュー座標変換とはカメラが原点になるように座標を変換することです。そして、プロジェクション座

標変換とは遠近感の存在しないビュー座標からプロジェクション座標に変換することです。視錐台とよばれる跳び箱状のカメラが見える部分を立方体に変えるものです。スクリーン座標変換は立方体の見える範囲を平面にするものです。これらの変換で行列という物を使います。行列は大学の範囲ではありますが行列変換の仕組みは非常に興味深いのでぜひ調べてみてください。

コードを見てみます。

ソースコード 7.9　Direct3D

```
D3DXMATRIX mat_world, mat_trans, mat_rot, mat_rotx, mat_roty, mat_rotz, mat_sca;
//初期化
D3DXMatrixIdentity(&mat_world);
D3DXMatrixIdentity(&mat_rot);
D3DXMatrixIdentity(&mat_trans);
D3DXMatrixIdentity(&mat_sca);
//移動
D3DXMatrixTranslation(&mat_trans, 0, 0, 10);
//回転
D3DXMatrixRotationX(&mat_rotx, D3DXToRadian(0));
D3DXMatrixRotationY(&mat_roty, D3DXToRadian(0));
D3DXMatrixRotationZ(&mat_rotz, D3DXToRadian(0));

D3DXMatrixMultiply(&mat_rot, &mat_rot, &mat_rotx);
D3DXMatrixMultiply(&mat_rot, &mat_rot, &mat_roty);
D3DXMatrixMultiply(&mat_rot, &mat_rot, &mat_rotz);
//スケール
D3DXMatrixScaling(&mat_sca, 1.0f, 1.0f, 1.0f);
mat_world *= mat_sca * mat_rot * mat_trans;
g_D3DDevice->SetTransform(D3DTS_WORLD, &mat_world);
```

これがワールド座標変換のコードです。最後の行列をかけているところでは行列をかけ合わせることで行列それぞれの動作を含んだ matworld を作成しています。

ソースコード 7.10　Direct3D

```
D3DXMATRIX matproj, matview;
D3DXVECTOR3 camera_pos(0, 0, -10);
D3DXVECTOR3 eye_pos(0, 0, 0);
D3DXVECTOR3 up_pos(0, 1,0);

D3DXMatrixIdentity(&matview);
D3DXMatrixLookAtLH(&matview, &camera_pos, &eye_pos, &up_pos);
g_D3DDevice->SetTransform(D3DTS_VIEW, &matview);

D3DVIEWPORT9 vp;
g_D3DDevice->GetViewport(&vp);
float aspect = (float)vp.Width / (float)vp.Height;
D3DXMatrixPerspectiveFovLH(
    &matproj,
    D3DXToRadian(45),
    aspect,
    0.1f,
    500.0f
);
g_D3DDevice->SetTransform(D3DTS_PROJECTION, &matproj);
```

これがプロジェクション座標変換のコードです。このような形で DirectX は行列変換を行っています。

7.4　実装

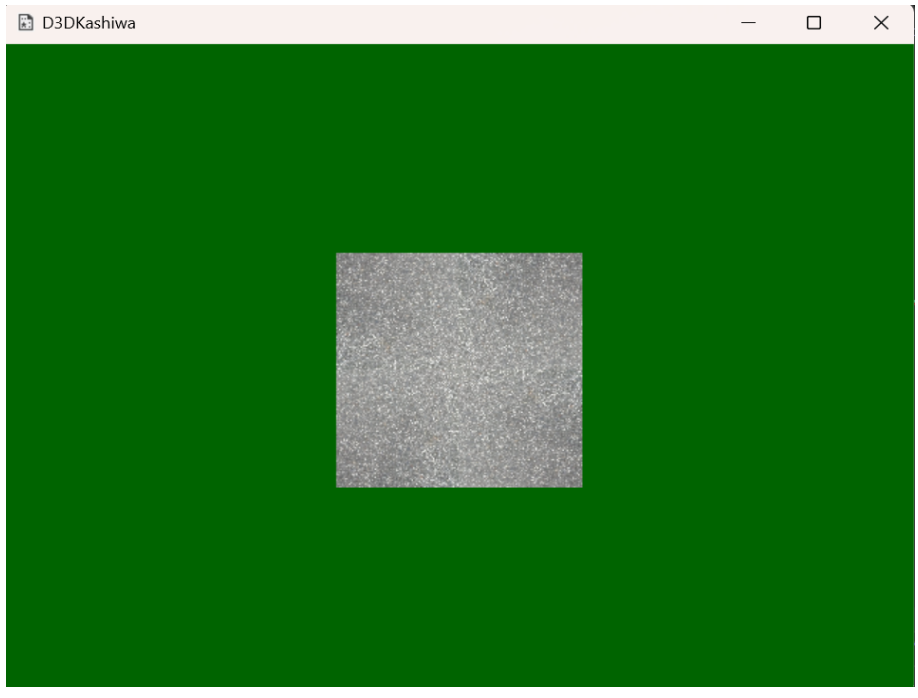


図 7.4　画面

これらのことを踏まえて実装していきます。テクスチャを張り付けて簡単な画面にしたのが上の図４です。背景が緑なのは Clear（）するときに Clear する色を緑色に設定したからです。

頂点は図５のようにリストを作って管理します。6 頂点ありますが実際の画面では 4 頂点しかありません。これはどういうことかというと、四角形は２つの三角形で出来ているので 3 × 2 個頂点ができているのです。上３つが三角形 1、下３つが三角形 2 の座標です。頂点は時計回りに指定していきます。この指定の仕方をトライアングルリストといいます。他にも指定の仕方はあと２種類ほどあります。例えば {-3,-3,0,0,1} となっているとき最初の３つが x,y,z 座標です。今回 Z 座標は全部 0 なので X,Y 座標を見ると時計回りになっていることがわかります。

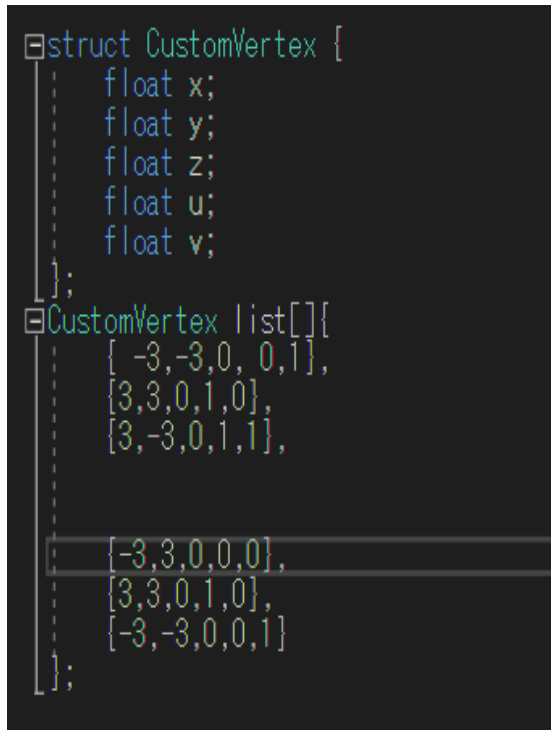


図 7.5　頂点

7.5　まとめ

DirectX はゲームの根幹となるシステムでコードを書くとゲームの構造がよりわかりやすくなると思います。是非皆さんも実装してみてください。

7.6　参考文献

yttm-work
https://www.yttm-work.jp/directx/directx_index.html
ゲームつくろー
http://marupeke296.com/DirectXMain.html
一週間で身につく WIN32 プログラミングの基本
https://c-lang.sevendays-study.com/win32/index_win32.html

第 8 章

Discord の Bot を作ってサーバーで動かしてみた

79 回生 かつどん

8.1 はじめに

こんにちは。79 回生のかつどんです。灘校生活の半分が終わってしまい震えています。あと 10 年は卒業したくない。最近フリーレンにめっちゃはまってます。好きなキャラはユーベルとメガネ君です。

さて、去年部室のサーバーの中の 1 つの管理をすることになり、それでマイクラのサーバーを建てたりしていろいろ遊んでいるのですが、その中に友達と遊ぶ用の Discord の Bot を動かすというのがあります。今回は今動かしている 2 つの Bot について書きます。よろしくお願いします！

1 つめの Bot マルコフ連鎖

これはただ遊びで作ったある友人の架空の発言を作るネタ Bot です。文章におけるマルコフ連鎖というのは、たくさんの文章を単語ごとに区切って、ある単語の次に来る単語を確率的に決めていって文章を作るというものです。文章だとわからないと思うので具体例を挙げて説明したいと思います。

- 僕 は 葬送 の フリーレン が 好き だ 。
- 僕 の 好き な キャラ は ユーベル と メガネ君 だ 。
- 僕 は 13 巻 の 発売 が 楽しみ だ 。

この 2 つの文章を見たとき、例えば「僕」という単語の次に来る単語は 67 %の確率で「は」、33 %の確率で「の」、「が」の次に来る単語は 50 %で「好き」、50 %で「楽しみ」ということになります。このように単語の次にどの単語が来るかの確率を計算し、その確率に従って次の単語を決めていき、文章を生成します。この 2 つで考えると、

- 最初は 100 %で「僕」
- 「僕」の次は 67 %で「は」、33 %で「の」 「は」が選ばれたとする
- 「は」の次は 33 %で「葬送」、33 %で「ユーベル」、33 %で「13 巻」「ユーベル」が選ばれたとする
- 「ユーベル」の次は 100 %で「と」

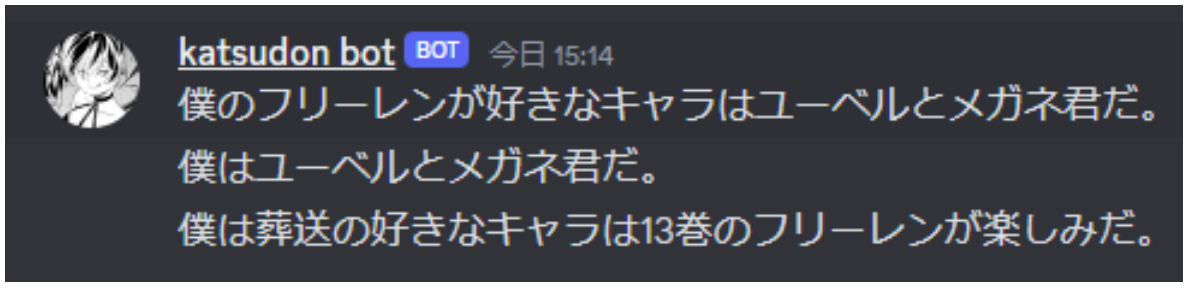
というように生成されていき、「。」にたどり着くまで続いていきます。これをもっと大量の文章で行っ

てその人が言いそうな文章を生成する、というのがマルコフ連鎖です。お分かりいただけたでしょうか？僕はこれを友人のやばい発言を集めて行いました ()

今回は Python を用いてマルコフ連鎖とそれを使った Bot を実装しました。生成の流れを簡単に説明すると、

- 単語で分けられた大量の友人の発言を用意する (自動で日本語の文章を区切ってくれる MeCab っていうモジュールがあるが、友人の発言が特殊すぎてうまくいかなかったので手作業で分けた)
- 単語のリストを作る
- 単語ごとに次の単語に何が何回出てきたかを数えて確率を計算する
- 1 分に 1 回先ほどのように文章を生成する
- 生成された文章を Discord に送信する

という流れです。 試しにさっきの例文で動かしてみた結果を貼っておきます。



Discord に送るのには discord.py というモジュールを使います。ただメッセージを送るだけでなくこの Bot で使っている定期送信や次に使うメッセージに応答する機能などもあります。

8.2 2 つめの Bot レート計算

これは割と実用的な Bot だと思います。NPCA 御用達の競技プログラミングサイト「AtCoder」のレートシステムを使って複数人で一緒にゲームをするときにレートを付けて遊んでみよう！ という Bot です。ちなみに事の発端は Discord のボイチャで遊べる Putt Party というゴルフゲームをレートを付けてやろうっていうノリになったことです。この Bot の基本的な流れは

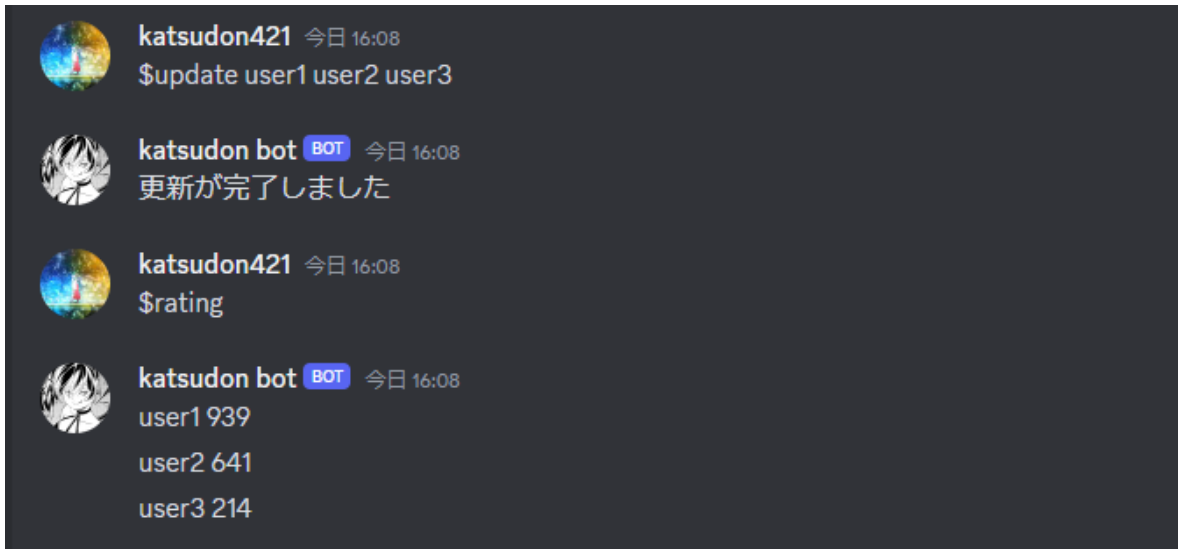
- ユーザ登録 (ユーザ名と初期化されたレートを登録する)
- 複数人でゲームで対戦する
- 順位を入力する
- レート更新 (入力された順位と試合前のレートから参加した人それぞれについて新しいレートを計算し更新する)
- レート表示 (レートが高い順に参加者を並べ替え Discord に送信する)

って感じです。

Bot を動かしてみた様子も載せておきます。

最初は user1〜3 の 3 人がレート 0 の状態です。「update user1 user2 user3」というのは 3 人でゲームをしたら 1 位が user1、2 位が user2、3 位が user3 になったという結果を登録してレートを更新しています。下に表示されているのが更新後のレートです。

詳しいレートの計算方法や実装についてはややこしすぎるのでここでは割愛します。



8.3 最後に

ここまでお読みいただきありがとうございました。みなさんもぜひオリジナルの Bot を作って遊んでみてください！ それでは！

第 9 章

$\sqrt{2}$ に整数を掛けてほとんど整数にする

79 回生　number_cat

9.1　はじめに

こんにちは。はじめまして。79 回生の number_cat と申します。普段は競技数学とか競技プログラミングとかをしてます。パソコンのことがよくわかっていないので数学について書かせていただきます。

9.2　$\sqrt{2}$ とは

$\sqrt{2}$ は $x \times x = 2$ を満たす正の実数で、その値は $1.41421356\cdots$ です。$\sqrt{2}$ は無理数であることが知られており、これはすなわち $p\sqrt{2}$ が整数となるような正整数 p は存在しないということです。しかし、ちょうど整数にならなくても、$p\sqrt{2}$ がめちゃくちゃ整数に近くなるような正整数 p はあるかもしれません。あるとすればどのような数なのかを調べてみたいと思います。

9.3　実際に探してみた

どうやって探しましょうか。手計算はさすがに大変そうですし、ここはコンピュータの力を借りてみましょう。とりあえず、$\left|p\sqrt{2} - n\right|$ が 0.01 未満となるような整数 n が存在するようなものを探して、それが小さく更新されるところの最初のいくつかを表にしてみました。

表 9.1　整数に近いもの

p	$p\sqrt{2}$
99	$140.00714\cdots$
169	$239.00209\cdots$
557	$816.00125\cdots$
985	$1393.00035\cdots$

なるほど。 $99\sqrt{2}$ ってほとんど整数なんですねえ。知りませんでした。でも規則性はなかなか見えないですね。私がコンピュータで見つけた一番整数に近そうなものは $388883860\sqrt{2}$ で、これは $549964828.9999999627\cdots$ と、3.7×10^{-8} くらいしか整数との差がありません。しかし、もっともっと整数に近いものを探したくありませんか？ 探したくてたまりませんね。しかしコンピュータは 1 秒間に 10^8 回程度の計算しか（人間に比べれば十分すごいですが）できません。そのためそれぞれの p につい

て $p\sqrt{2}$ を計算していてはこれ以上探すのは難しそうです。どうしましょうか。

9.4 ひらめき

ここで、上の表の $p = 169$ に注目してみましょう。これはつまり $169\sqrt{2} \approx 239$ ということです。(「≈」はほとんど等しいという意味の記号です）これは $239 - 169\sqrt{2} \approx 0$ と言い換えられますが... 勘のいい方ならもうお気づきでしょう。そうです。実は $239 - 169\sqrt{2} = (\sqrt{2} - 1)^7$ なんです。$\sqrt{2} - 1 < 1$ なので、以下が成り立ちます。

$$1 > (\sqrt{2} - 1)^1 > (\sqrt{2} - 1)^2 > (\sqrt{2} - 1)^3 > \cdots > (\sqrt{2} - 1)^n > \cdots > 0$$

では例えば、 $(\sqrt{2} - 1)^{100}$ について考えてみましょう。これを展開すると、ある正整数 a, b を用いて $a - b\sqrt{2}$ と表されます。これがほとんど 0 だということが分かったので、 $a - b\sqrt{2} \approx 0$ 、つまりは $b\sqrt{2} \approx a$ ということになります。これは、 $b\sqrt{2}$ がほとんど整数だということを意味します！ 実際に $(\sqrt{2} - 1)^{100}$ を展開してみましょう。こういうときはプログラミング...！ ですが今回は Wolfram Alpha という高度な計算が可能なウェブサイトを使って計算します。非常に便利なので皆さんも活用しましょう。$(\sqrt{2} - 1)^{100} = a - b\sqrt{2}$ とおくと、 a, b は、

$$a = 947411251496369334178730799209000017937, b = 669920920505516376634389067131823137 72$$

となります。そのため、 $669920920505516376634389067131823137 72\sqrt{2}$ がほとんど整数だということがわかります！ 実際、

$$669920920505516376634389067131823137 72\sqrt{2} \approx 947411251496369334178730799209000017937 - 5.28 \times 10^{-39}$$

であり、めちゃくちゃ整数に近いです。すげー。

9.5 背景の紹介

ここからはこの話題に関する背景知識についての紹介です。詳しい証明は省くため、気になった方は文献を見ていただけると助かります。雰囲気をつかんでもらえたら幸いです。

9.5.1 $\sqrt{2}$ を有理数で近似

私達が探し求めていたものは $a\sqrt{2} \approx b$ を満たす整数 a, b の組ですが、これは $\sqrt{2}$ を有理数 $\frac{b}{a}$ で近似することとつながっています。($a\sqrt{2} \approx b$ を両辺 a で割るとわかります）そのため、以下では $\sqrt{2}$ を有理数で近似する話から入ります。

9.5.2 良い近似

ある実数 x を既約分数（これ以上約分できない分数） $\frac{p}{q}$ で近似したとしましょう。このとき、$|x - \frac{p}{q}| < \frac{1}{q^2}$ を満たすならば $\frac{p}{q}$ は x の「良い近似」であるということにします。雑に探してもなかなか良い近似は見つかりませんが、例えば $\frac{140}{99}$ は $\sqrt{2}$ の良い近似です。実際、 $|\sqrt{2} - \frac{140}{99}| \approx 7.215 \times 10^{-5}, \frac{1}{99^2} \approx 1.020 \times 10^{-4}$ で $\frac{1}{99^2}$ のほうが大きいことが分かります。これは前に出てきた

$99\sqrt{2}$ がほとんど整数であることと関係しています。つまり私たちの目的は $\sqrt{2}$ の良い近似を求めることだったわけです。ですが、前に出てきたもの、例えば $\frac{816}{557}$ は良い近似でないことに注意してください。$(\sqrt{2}-1)^n = a - b\sqrt{2}$ とおいて、 $b\sqrt{2}$ がほとんど整数だという話をさっきまでしてましたが、ここから得られる $\sqrt{2}$ の近似、 $\frac{a}{b}$ は良い近似なのでしょうか？ 実はすべての正の整数 n について、$(\sqrt{2}-1)^n = a - b\sqrt{2}$ と置いたときの $\frac{a}{b}$ は $\sqrt{2}$ の良い近似であることが証明できます。すごいですね。しかし、この話は $\sqrt{2}$ のときは成り立ちますが、 例えば $\sqrt{31}$ にすると成り立ちません。つまり、ある正の整数 n について、 $(\sqrt{31}-5)^n = a - b\sqrt{31}$ と置いたときの $\frac{a}{b}$ は $\sqrt{31}$ の良い近似でないのです。実際 $n=5$ としてみると良い近似にならないことが分かります。では、この 2 と 31 の違いって何なんでしょう？

9.5.3 連分数

ここで正則連分数の登場です。正則連分数とは以下のような形をした分数のことを言います。（ただし a_0 を整数、 $a_1, a_2, a_3, \cdots$ を正の整数とします）

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}}$$

これを以下略記して $[a_0; a_1, a_2, a_3, \cdots]$ と書きます。すべての実数はこの正則連分数の形で表すことができます。実数 x を正則連分数の形で表したものを x の正則連分数展開といいます。x が有理数のとき、x の正則連分数展開は有限回で打ち切りになり、 x が無理数のときは無限に続くことが知られています。そして、 x が無理数のときに正則連分数展開を有限回で打ち切ったとき、それは有理数になりますがこれは必ず x の良い近似となります。この結果を用いると先ほどの $\sqrt{2}$ のときに全部良い近似だけど $\sqrt{31}$ のときにはそうではないことの説明ができます。正則連分数展開は $a_0, a_1, \cdots$ と順に求めていくことができ、$\sqrt{2} = [1; 2, 2, 2, \cdots]$ と 2 の繰り返し、$\sqrt{31} = [5; 1, 1, 3, 5, 3, 1, 1, 10, \cdots]$ と $1, 1, 3, 5, 3, 1, 1, 10$ の繰り返しだとわかります。 $\sqrt{2}$ の正則連分数展開を a_n までで打ち切ったものが $(\sqrt{2}-1)^{n-1} = a - b\sqrt{2}$ と表したときの $\frac{a}{b}$ と一致するのです！ これが $\sqrt{31}$ では起きないというわけです。 $\sqrt{31}$ の良い近似を得るには $(\sqrt{31}-5)^n$ を展開するよりも正則連分数展開をしたほうがよいですし、 $\sqrt{2}$ の良い近似を得るにしろ、正則連分数展開をしても全く問題ないということです。実は、「実数 x を $\frac{p}{q}$ で近似して $|x - \frac{p}{q}| < \frac{1}{2q^2}$ が成り立つなら、それは実数 x の正則連分数展開を打ち切ったものである」という定理があるので、正則連分数展開は良い近似を得るのにとても強い武器であることが分かります。余談ですが、円周率の近似値として用いられる $\frac{22}{7}$ は円周率の正則連分数展開を途中で打ち切ったものです。

9.5.4 さらに良い近似

連分数を用いれば実数 x の良い近似を見つけるのにはもう困らなくなりました。ここからは良い近似よりも厳しい条件を満たす、「さらに良い近似」について紹介します。（いずれも証明を追えていないのでかなり雑になります）

フルヴィッツの定理

「任意の無理数 x に対し互いに素な整数 p, q であって $|x - \frac{p}{q}| < \frac{1}{\sqrt{5}q^2}$ となるものが無限個存在する」というもので、 $\sqrt{5}$ の部分をこれ以上大きくすると成り立たなくなります。 $\sqrt{5}$ が出てくるのが面白いですね。

リウヴィルの定理

「任意の有理数体上次数 n の代数的無理数 x に対しある定数 $c(x) > 0$ が存在して任意の整数 $p, q > 0$ であって $|x - \frac{p}{q}| > \frac{c(x)}{q^n}$ が成り立つ」というものです。この定理は超越数であるかどうかの判定に使えます。

ロスの定理

「任意の代数的無理数 x に対し任意の $\epsilon > 0$ に対してある定数 $c(x, \epsilon) > 0$ が存在して任意の整数 $p, q > 0$ であって $|x - \frac{p}{q}| > \frac{c(x, \epsilon)}{q^{2+\epsilon}}$ が成り立つ」というものです。これはリウヴィルの定理の改良版です。この定理からは「任意の代数的無理数 x に対し、任意の $\epsilon > 0$ に対して $|x - \frac{p}{q}| < \frac{1}{q^{2+\epsilon}}$ を満たす有理数 $\frac{p}{q}$ は有限個しかない」ということが分かります。実数 x に対して「 $|x - \frac{p}{q}| < \frac{1}{q^\kappa}$ を満たす有理数 $\frac{p}{q}$ は有限個しかない」という性質を満たす κ の下限を x の無理数度といい、この定理から代数的無理数の無理数度が 2 であることが分かります。他に、 e の無理数度は 2 、 π の無理数度の上限が 7.103 程度、 π^2 の無理数度の上限が 5.095 程度であることが分かっているそうです。

9.6 おわりに

いかがだったでしょうか。少しでも数学の楽しさを共有出来ていたらうれしいです。ここまで読んでくださりありがとうございました！

9.7 参考文献

マスオ. "実数を分数で近似する【ディリクレのディオファントス近似定理】". 高校数学の美しい物語. https://manabitimes.jp/math/2765, (最終閲覧日 2024-05-01)

水谷治哉. 連分数とディオファントス近似. http://www.math.sci.osaka-u.ac.jp/koukai/mizutani2018.pdf, (最終閲覧日 2024-05-01)

Weisstein,Eric W. "Irrationality Measure". Wolfram MathWorld.

https://mathworld.wolfram.com/IrrationalityMeasure.html, (最終閲覧日 2024-05-01)

橋本竜太. 実数の有理数近似と連分数展開. https://www.ma.noda.tus.ac.jp/u/ha/SS2006/Data/Hokoku/hash (最終閲覧日 2024-05-01)

第 10 章

OpenUtau をやる

80 回生　michael

10.1　はじめに

こんにちは。80 回生（中３）の michael と申します。中３までってめっちゃ早いですね。時間は大切に使いましょう（自戒）。普段は文藝同好会で俳句を詠んだり小説を書いたりしています（パソコンと全然関係無いですね）。パソコン部には中１から入っていて、中２からは競プロをやっています。弱いです……。

この記事では僕の普段の活動から離れて、パソコンを使って作曲（いわゆる DTM）をしたという報告を書いていきます。ボカロが好きなので OpenUtau を使ってボーカルも入れます（UTAU はボーカロイドではないという指摘はやめてください）。

10.2　使ったソフトなど

使った作曲用ソフトその他諸々についてもせっかくなので解説を入れていきます。

10.2.1　ハード

使っている PC の OS は windous10、オーディオインターフェース（入出力される音と PC との間の架け橋となる）は買っていません。MIDI キーボード（鍵盤ハーモニカのハーモニカ抜きみたいな形をしていて打ち込んだ音をそのまま PC に入力できる機械）は便利そうなので買いました。

10.2.2　Cakewalk

こいつは DAW といって音楽制作をできるソフトフェアです。音楽制作ができると言ってもこのソフトだけで作ることは（基本的には）できず、後述する VST 音源かサンプルの音源を持ってきて DAW 側が呼び出せるようにする必要があります。無料で、しかも VST 音源付属なので選びました。インストール方法、使い方はネットに情報がわんさかあります。

ざっくり用語解説

DTM

Desk Top Music の略で、パソコンを使用して音楽を作成編集することの総称です。後述する DAW

や VST などが必要になります。

DAW
: Digital Audio Workstation の略で、 MIDI やオーディオを扱い音楽を制作することの出来る DTM ソフトウェア、音楽制作ツール全般を指します。

VST
: Virtual Studio Technology の略で、音を出す「VST インストゥルメント」と音を加工する（例えばノイズ除去などの）「VST エフェクト」の二種類に大別されます。

MIDI
: Musical Instrument Digital Interface の略で、電子楽器の演奏データを機器間で転送・共有するための共通規格です。楽譜みたいなもんです。

10.2.3 OpenUtau

二つ目のソフトです。これも無料でインストール可能な、オープンソースの歌声エディタです。MIDI をインポートする、もしくは OpenUtau のソフトで音を入力して、その後このソフトで歌詞を打ち込むことで、別途に外部からダウンロードした UTAU 音声ライブラリ（DAW の VST インストゥルメントに相当）から歌を合成することができます。

10.3 本編

Cakewalk でごく簡単な曲を作って OpenUtau で歌を付けるところまでやってみます。

まずは曲のリズムを付けるドラムを作っていきます。事前にプロが演奏したものを録音したものがサンプル音源として公開されているので、これをそのまま使用するだけです。Cakewalk と一緒にダウンロードできる BandLabAssistnt からドラムパターンの素材をダウンロードして、ファイルを Cakewalk にドラッグアンドドロップ。このループ素材は BPM が 120、長さが 8 秒でしたのでこの曲も BPM120、長さは二倍の 16 秒で作っていきます（別にループ素材のテンポを弄ることもできるのですが……）。

次は和音を作っていきます。これは DAW で打ち込みをして作ります。Cakewalk にインステュルメントのトラックを作り、Cakewalk 付属の VST 音源 Bass Guitar を指定。ピアノロールビューを起動し鍵盤をクリックすると、音が鳴ることを確認します（MIDI キーボートを Cakewalk の環境設定で指定していれば、これを演奏しても音が鳴ります）。コード進行に著作権はないらしいので好きな曲のコード進行を写しました。ごめんなさい。

最後にメロディを作ります。和音と同様にトラックを作り、VST 音源には Electric Piano を指定。ハネリズムを使いました。コードごとにアボイドノートっていう使うと不協和音になってしまう音があるらしいので気をつけていきます。ここまでで音楽の歌唱以外の部分が完成しました。最後に OpenUtau の方で歌を付けていきます。

OpenUtau 用の UTAU 音源をダウンロードします。等身大ヒューマノイドロボット、足立レイの音源をダウンロードしました。どうやら zip 版だと連続音と単独音のファイルあたりがややこしいようで少し苦労しました。歌詞は全部「あ」にします。何も思いつかなかったので……。

仕上げに VST エフェクトをかけます。Bertom - Denoiser、Boost11 を使いました。ファイルとして出力して完成です。

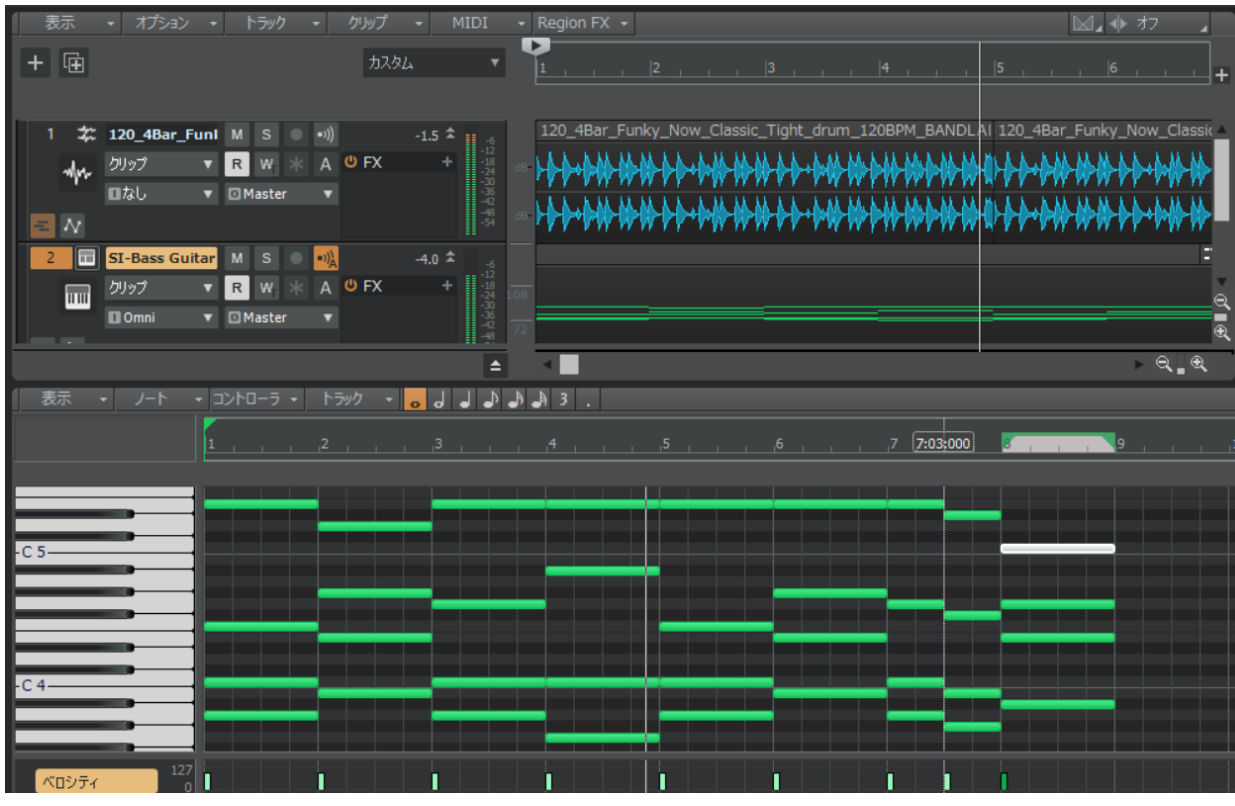


図 10.1　制作画面：Cakewalk　和音まで

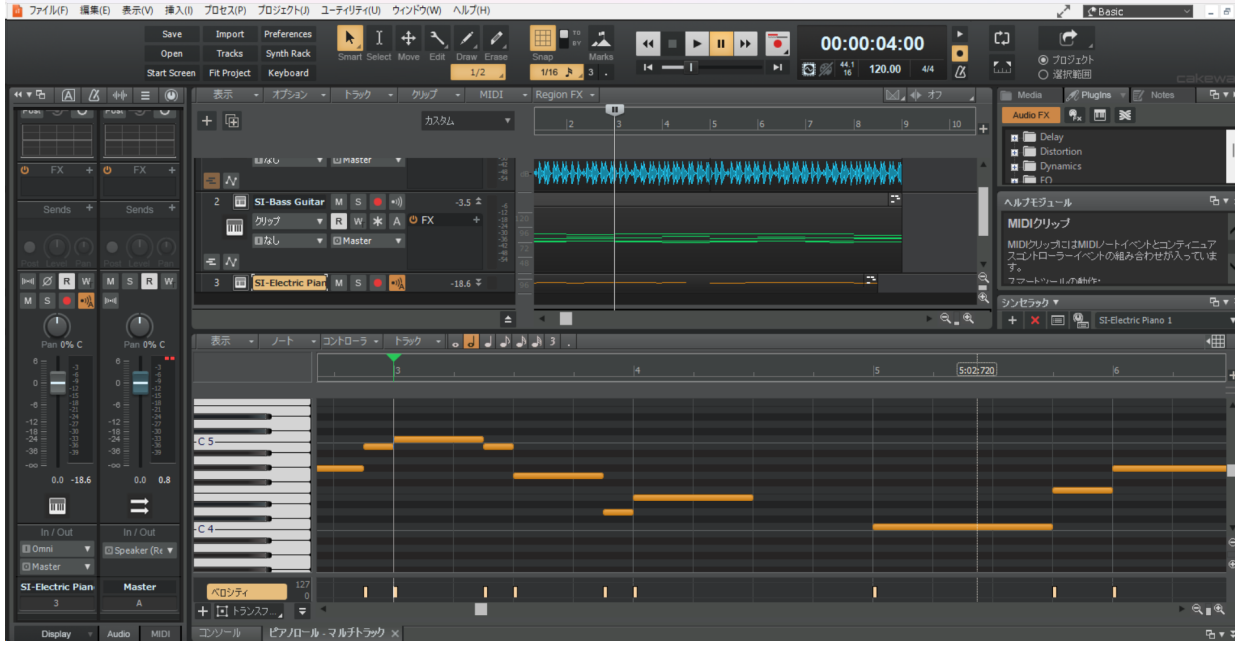


図 10.2　制作画面：Cakewalk　メロディまで

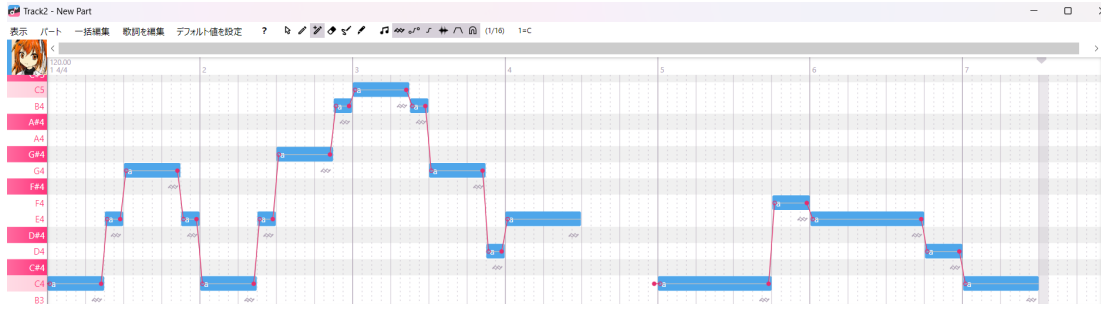


図 10.3　制作画面：OpenUtau　音程を入力する、パートごとのエディタ

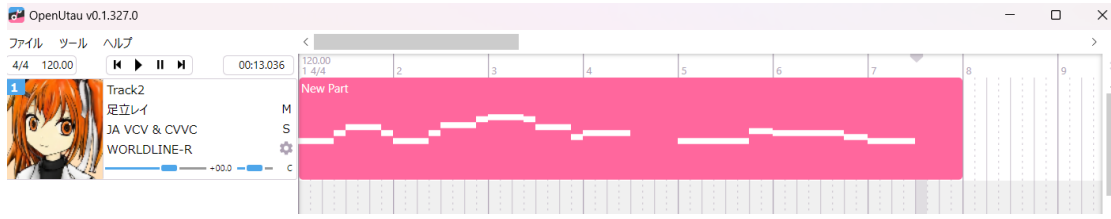


図 10.4　制作画面：OpenUtau　全体

10.4　おわりに

OpenUtau 要素が少なくて申し訳ありません。ここまで読んで下さった読者の方本当にありがとうございます！