

**National**

**vol.37**

**COPYRIGHT  
S.R.D.G**

## 目次

顧問挨拶	1
代表挨拶	2
—活動報告—	
黒澤 敬真 総合工学科 II類 1年	「ゆっくり MovieMaker で動画を作る」 3
小幡 充輝 総合工学科 II類 1年	「Unity で本田圭佑じゃんけんゲーム作成」 5
須貝 湊 ロボティクスコース 2年	「技術ブログを作った話」 7
大高 康介 ロボティクスコース 3年	「JavaScript でカレンダーを制作する」 13
鈴木 佑 ロボティクスコース 3年	「Unity でオセロゲームを作成する」 15
富山 結都 ロボティクスコース 5年	「弾幕結界」 17
鈴木 晴人 ロボティクスコース 5年	「いざ、Nim を学ばん」 19
—プログラミングコンテスト—	
競技部門	
大高 康介 ロボティクスコース 3年	
鈴木 佑 ロボティクスコース 3年	
佐藤 至 ロボティクスコース 4年	23
—プロコン旅行記—	
鈴木 佑 ロボティクスコース 3年	25
編集後記	29

## 顧問挨拶

---

目標への手順は細切れに

ソフトウェア研究部会 顧問 北島宏之

先日 10/14(土)～15(日), 第 34 回全国高等専門学校プログラミングコンテストが, 福井工業高等専門学校の主管の下, 越前市のサンドーム福井を会場に開催され, ソフ研からは競技部門に出場しました. 結果としては決勝トーナメント進出は叶わなかったものの, コロナ前の規模まで戻った大きな大会となり, 参加部員はみな貴重な経験と勉強ができたように思います. プロコン旅行記は, おそらくどなたかが執筆されるように思いますので, 当方からは今回のプロコンと合わせて, 日々の雑感の中から 1 つだけ思うところを書き, 顧問からのご挨拶とさせて頂ければと思います.

昨今のプロコンにおいて, 課題部門や自由部門の本選に進んだ作品等は, どれも企画や設計から, 調査, 実装, 実験や評価までかなりの時間をかけてまとめていることが見て取れます. どのチームの学生さんも地道に時間をかけて頑張ってきた様子が伺え, 高専生の様々な力を感じられることは一教員としてとても嬉しくなります. 合わせて, 入賞作品の完成度の高さや, 下世話ですが企業賞の豪華さに憧れを抱き, ソフ研から課題部門や自由部門でも入賞を目指したいと思うところです. ただその一方で, 目標をこれら部門での入賞に掲げると, いわゆるハードルが高すぎると感じて開始時から気持ちが削がれてしまうのではと危惧することも少なくありません. 実際, 我々大人も, 時間を要する仕事が目の前にあるとなかなか手を付け始められないものです(私だけ?).

そこで, そのような時は最終的な目標を小さな目標に分割し, その小目標を 1 つずつ越えて行くようになるとなんとかなるように思います. 例えば, 上記に挙げた“企画”段階であっても, 多くの時間を要すると想像することで高いハードルを自ら設定してしまいやる気を削ぎかねません. なので, 例えば, 身の回りを見渡す, 課題を見つける, 見つけた課題を皆で持ち寄る, 持ち寄った課題を整理して絞る, 課題を解決する方法を提案する. さらに, 解決方法の提案についても, 皆で案を持ち寄る, 整理する, 案を絞る, 決定する, というように分割します. これら 1 つ 1 つを順に実行しますが, ただしその際に必ず間に十分な休憩を強制的に挟みます. こうすることで, 順に 1/3 終わり, 半分終わり, 8 割が終わった, といった状況が着実に得られます. うまく行けば, 休憩に入る際に次の手順に進みたい気持ちが残っていることで, より意欲が大きくなったり状態で次の手順に入ることも可能です. で, 気付いたら最後まで終わっていると. 以上, 北島オリジナルよりもっともらしく書きましたが, 何とか仕事術やモチベーションを維持する方法といった記事や書籍にも同じような内容が見られますので, 機会があればお試し頂ければと思う次第でした.

## 代表挨拶

---

### 会長挨拶

ロボティクスコース 3年 鈴木 佑

「Rationale」を手に取って頂き、ありがとうございます。3年前からの新型コロナウイルスによる高専祭の自粛も今年で取り扱われ、ようやくコロナ前と同じ規模で高専祭を行えるようになりました。各展示の飲食制限が緩和され、私達ソフトウェア研究部会も軽食を提供できるようになり、高専祭にいらっしゃった方達に楽しんでいただけたと思います。

さて、先々週の10/14(土)~10/15(日)に福井県で行われた第34回全国高専プログラミングコンテストに出場しました。ソフトウェア研究部会では、毎年参加していますが、コロナウイルスの影響もあり、現地に行くのは3年ぶりとなります。結果は初戦敗退とあまり芳しくなかったものの、参加者全員で力を入れたプログラムなので、全力を出せたと思います。

また、長らく掲載出来ていなかったプロコン旅行記や解法アルゴリズムなどを再びRationaleに掲載しているのでそちらもお読みいただければ幸いです。

最後にはなりますが、この「Rationale」の作成に関わった部員の皆様ならびに顧問の北島先生、佐藤先生、そして今「Rationale」を読んで下さっている全ての方々にこの場を借りてお礼申し上げます。

---

### 副会長挨拶

ロボティクスコース 2年 須貝 湊

Rationalを手に取って頂きありがとうございます。昨今のパンデミックも徐々に収まり、高専祭も昨年に引き続き一般公開となりました。弊部では本誌が主な活動報告となっており、特に一般向けに活動をアピールする上では最も重要なものとなっています。ゆえに、本誌を頒布することのできる一般公開が復活し継続されることの大変喜ばしいことです。

さて、「地球沸騰」とまで言われた暑さも少しづつ収まり、涼しい季節がやってきました。あんなにうるさく鳴いていた~~昼~~のセミの声も聞こえなくなり、夜になればスズムシの鳴き声が聞こえています。しかし今年は気温が不安定で、かなり暑い日と寒い日が数日置きに入れ替わる特殊な気候となっています。そのため、急激な温度変化による免疫力の低下、ひいてはインフルエンザの流行などを引き起こしています。しっかり上着を着て体温調節する、手洗いうがいを徹底するなど予防に努めましょう。

本誌は例年通り弊部ホームページにてWeb版が公開されています。Web版ではPDF版 Rational以外にも追加のメディアなどが公開されていますので是非一度開いてみてはいかがでしょうか。

最後に、Rational制作のために寄稿して頂いた部員各位、ご協力頂いた顧問各位、そして何より本誌を読んでくださっている読者各位に厚く御礼申し上げます。

# ゆっくりMovieMakerで動画を作る

## 1. はじめに

今回私は「ゆっくり MovieMaker4(以下、ymm4 と表記)」という動画編集ソフトウェアを使って動画を制作しました。ここでは、その大まかな流れについてまとめています。

## 2. ソフトウェアのダウンロード

まずは ymm4 を Web サイト「饅頭遣いのおもちゃ箱」(図 1)からダウンロードします。今回は諸事情により、読み上げソフト「AquesTalk」が入っていない「ゆっくり MovieMaker Lite」をダウンロードしました。

このままでは文章を読み上げることができないので、商用・非商用問わず無料で利用できる読み上げソフト「VOICEVOX」を公式 Web サイト(図 2)からダウンロードします。



図 1 饅頭遣いのおもちゃ箱



図 2 VOICEVOX の Web サイト

総合工学科 1年 黒澤 徹真

## 3. 素材の用意

### 3.1.立ち絵

動画に使うキャラクターの立ち絵を用意します。今回は VOICEVOX のずんだもんが台本を読み上げるので、ニコニコ静画から「ゆっくりずんだもん素材」をダウンロードしました(図 3)。

ダウンロードした zip ファイルを解凍し、PSD ファイルを保存します。ymm4 を開き(図 4)、①キャラクター設定からずんだもんを選択します。②立ち絵の種類を「PSD ファイル」に設定します。先ほど保存した PSD ファイルを参照し、ファイルマークの隣にあるペンのマークをクリックします。③目や口に瞬き、口パクの素材を追加することで、アニメーションするようになります。

### 3.2.その他素材

背景、効果音、BGM 等の素材は必要になつたら探してダウンロードしていきます。主に使用した Web サイトのリンクを末尾に掲載しておきます。各素材は利用規約等にしつかりと目を通してから使用していきます。



図 3 ゆっくりずんだもん素材

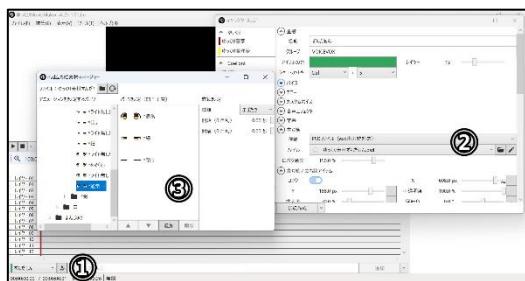


図 4 立ち絵の設定

#### 4. 動画制作

ここから動画の制作に移ります。台詞を入力したり、ダウンロードした素材を貼り付けたりすることで動画を構成していきます(図 5)。この時に、レイヤーの番号が大きいほど手前に表示されるので、背景<キャラクター<字幕 のように配置します。背景をレイヤー0、立ち絵をレイヤー14、字幕をレイヤー15 に配置しました。また、台詞を入力してうまく読み上げできなかった場合は、アクセントが来る場所にアポストロフィ(')を打つことで調整が可能です。

動画が完成したら何度か見返し、ミスがないか確認します。こうしてミスが見当たらなければ、動画を出力していきます。今回完成させた動画のスクリーンショットを図 6 として掲載します。

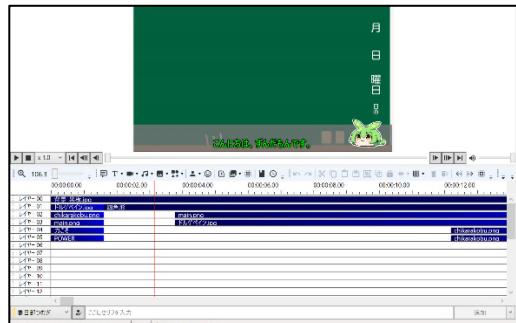


図 5 ymm4 のプレビューとタイムライン



図 6 完成した動画

#### 5. 終わりに

最後に、今回作成した動画の QR コードを図 7 として掲載しておきます。今後も 3~10 分程度の動画を YouTube に投稿していく予定です。もし興味がございましたら、そちらもご覧いただけたと幸いです。



図 7 動画のQRコード

#### 饅頭遣いのおもちゃ箱

<https://manjubox.net/>

#### VOICEVOX

<https://voicevox.hiroshima.jp/>

#### ニコニコ静画

<https://seiga.nicovideo.jp/>

#### 効果音ラボ(使用した効果音)

<https://soundeffect-lab.info/>

#### 騒音のない世界(使用したBGM)

<https://noiselessworld.net>

#### ニコニ・コモンズ

<https://commons.nicovideo.jp/>

# Unity で本田圭佑とじゃんけんするゲームを作成する

総合工学科 1年 小幡 充輝

## 1. はじめに

今回私は Unity を使用してじゃんけんゲームを作成しました。言語は C#、開発環境は Unity、Microsoft Visual Studio 2022 です。

## 2. 作成の流れ

### 2.1. 絵を描く

絵描きソフトの Fire Alpaca を使用し、本田圭佑(図 1)、ゲー、チョキ、パーを描きました。できた画像をドラッグアンドドロップで Unity に実装します。画像を PNG 形式で保存しなかった場合、Unity で画像が真っ黒に表示されてしまうので、保存する場合は PNG 形式で保存してください。

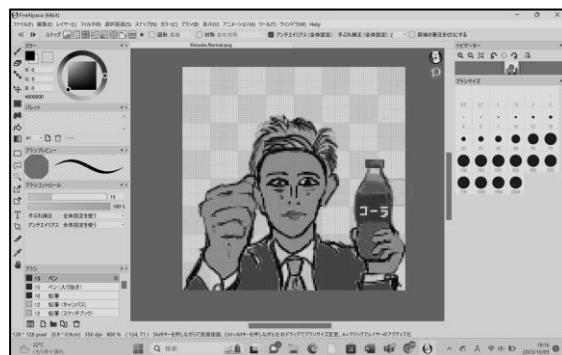


図 1 実際に描いた本田圭佑

### 2.2. 効果音、BGM をダウンロード

YouTube にある本田圭佑がペプシコーラをかけてじゃんけんしている動画を YouTube にある動画の音声をダウンロードできる dirpy(図 2)で音声をダウンロードし、必要な部分だけ Audacity(図 3)というソフトで切り取り、Unity に実装しました。

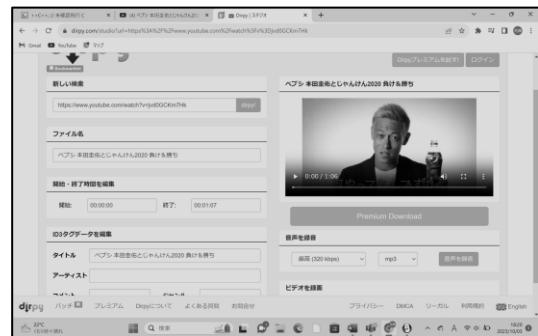


図 2 dirpy で動画の音声をダウンロード



図 3 Audacity で音声を切り取る

### 2.3. プログラムを書く

リスト 1 がじゃんけんを行い、その結果によって画面の切り替えを行うプログラム(リスト 1 のプログラムはゲーを選んだ場合のプログラム)です。じゃんけんの勝ち負けにより加算されるスコアを保存する変数を a、じゃんけんを行った回数を保存する変数 b を定義し、a、b の値によって勝ちか負けかを判別します。Unity に実装した画像や効果音をプロクラム上で設定できるようにします。

じゃんけんを行うプログラムでは、乱数を使用し敵がゲー、チョキ、パーのいずれかをランダムで出すようにします。勝ったら a の値を 1 加算し、

じゃんけんを行うごとにbを増やしていく、a が2以上になった時点で勝利画面に移動し、b が5で a が2に満たないとき、ゲームオーバー画面に移動します。また、画面にテキストを表示できるコードを書き、今自分がどのくらいのスコアかわかるようにします。これで完成です。

### リスト 1 じゃんけんを行うプログラム

```

int a = 0;
int b = 0;
private AudioSource audioSource;
public AudioClip ZyankenSE;
public AudioClip Youwin;
public AudioClip Youlose;
void Start()
{
    audioSource =
GetComponent<AudioSource>();
}
public void GuButton()
{
    audioSource.PlayOneShot(ZyankenSE);
    Invoke("CaseGu", 1.5f);
}
public void CaseGu()
{
    ++a;
    --FinishNumber;
    FinishText.text = $"あと:{FinishNumber}回";
    enemyManager.enemyHand =
Random.Range(0, 3);
    enemyManager.Judgement();
    if (enemyManager.enemyHand == 0)
    {
        judgementText.text = "あいこ";
    }
    if (enemyManager.enemyHand == 1)
    {

```

```

        judgementText.text = "かち";
        ++b;
        score++;
        ScoreText.text = $"SCORE : {score}/5";
    }
    if (enemyManager.enemyHand == 2)
    {
        judgementText.text = "まけ";
    }
    Debug.Log("a" + a);
    Debug.Log("b" + b);
    if (b >= 2)
    {
        SceneManager.LoadScene("victoryScene");
    }
    else if (a >= 5)
    {
        SceneManager.LoadScene("LoseScene");
    }
}
```

### 3. 終わりに

今回はプログラムに関してのレーショナルを書きましたが、作曲にも興味があるので次回は作曲でレーショナルを書きたいと思います。ですが、今回の作成でプログラミングにも興味が出てきたので、これからはプログラムも並行して学習したいと思います。また、今回はプログラムを書くのにかなり時間がかかっててしまったり、プログラムも粗削りなものだったので、今後さらにプログラムへの理解を深めていきたいです。

### 参考文献

#### 未確認飛行 C

<https://ufcpp.net>

#### Unity

<https://unity.com>

# 技術ブログを作った話

## 1. はじめに

こんにちは。私は Web 系の開発を主にやっており、使っている技術の紹介や備忘録などを残したいと思い立ちました。

しかしせっかくなら既存のサイトを使うよりだったら自分で作った方が面白そうだ、と思い今回は技術ブログを作りました。

### 1.1. 注意

[\*数字]は脚注を表します。

またフォルダ/ディレクトリに関しては読みやすさを考慮してフォルダに統一しました。

ソースコードは掲載の際に重要部分のみ抜粋しているため、実際のものとは異なります。リポジトリをご覧ください。

## 2. 使用技術の選定

ブログを作るにあたってまず

- プログラミング言語
- フレームワーク[\*1]
- ホスティングサービス[\*2]

を決める必要があります。

今回は上 2 つに関してはそこまで悩まず、とりあえず自分の慣れている構成で進めようと思い、それぞれ TypeScript[\*3]、Next.js v13[\*4]を選びました。

ホスティングサービスに関しては最初は持っているサーバーでやろうかとも思ったのですが、もっと気軽に作りたかったので Vercel[\*5]を選びました。また、ブログの記事のデータを置くために国産ヘッドレス CMS[\*6]サービスである microCMS も使うことにしました。

そしてブログのソースコード[\*7]を管理するた

ロボティクスコース 2 年 渡波 空

めに Git[\*8]を使い、Vercel で公開するために GitHub[\*9]にソースコードを置くことにしました。

[\*1] 開発に便利な機能がまとまった縁の下の力持ちのような物

[\*2] 実際に公開する時に利用するサーバーなどを提供するサービス

[\*3] Microsoft 社が主導して開発している言語で、JavaScript とは上位互換の関係にある。様々な機能が追加されたり JavaScript に先行して新機能が追加されたりするなど強力な言語

[\*4] Vercel 社が主導して開発している Web アプリケーションのフレームワークで Meta 社主導の React がベースとなっている。近年では Cookpad がサービス開発に取り入れるなど注目が集まっている

[\*5] 主に Web アプリケーションなどのホスティングに特化しているサービスで Next.js の開発元である同名の会社が運営している

[\*6] CMS とは記事や画像などのデータを管理する仕組みのこと。一般的にそれらを表示する機能まで付いてことが多い (WordPress など)が、ヘッドレス CMS の場合データを保持するだけで表示は別のシステムに丸投げする形となる

[\*7] 実際に書かれたプログラムそのもののこと

[\*8] ソースコードを自分の好きなタイミングで保存し、いつでも保存した時点に戻れる、いわばバックアップのようなことができる代物。神

[\*9] 世界最大級の Git のホスティングサービス。同名の会社が運営していたが数年前に Microsoft 社に買収された

### 3. 開発

#### 3.1. プロジェクトの作成

プロジェクトは `create-next-app`[\*10]を使って作成し、`yarn`[\*11]を使いつつパッケージ[\*12]を入れました。

主なパッケージは

- `microcms-js-sdk`(microCMSへのアクセスを簡単にする公式提供ツール)
- `microcms-richedit-processor`(記事のデータを処理しやすくする microCMS 社員による非公式ツール)
- `bootstrap`(旧 Twitter 社が開発している、Web サイトのパーティ集のようなもの)
- `react-bootstrap`(↑ Bootstrap を React 系向けに使いやすくしたもの)
- `rehype-〇〇`(記事のデータを処理してより最適な形に簡単に変えるためのツール。今回は `parse-react-stringify` を用いた)
- `unified`(↑ rehype-〇〇を使うためのツール)

となっています。(この他にもいくつか追加している)

これらのツールがあることで非常に開発がしやすくなります。

[\*10] Next.js のプロジェクトをわずか数ステップで作ることができるツール。今回は TypeScript を使う設定、そして Next.js v13 の目玉機能とも言える App Router の設定を ON にした

[\*11] ↓パッケージを管理するパッケージマネージャーの一種。他には `pnpm`、`Node.js` に付属する `npm` などが挙げられる

[\*12] プログラムをまとめて扱いやすくしたもの。ほとんどの場合 ↑ パッケージマネージャーを通して管理する

#### 3.2. 基礎を固める

手始めにトップページを作ります。そのためにはまずページの骨組みを作っていきます。Next.js は v13 から App Router という新しい作り方が使えるようになり、この骨組みを書くためのファイルを(名前さえ合っていれば)自動で検知してくれるようになりました。

リスト 1 が実際のコードです。以後書いていくページの内容は実際のページを出力する際にこの骨組みに埋め込まれます。

リスト 1: レイアウト・全ページ共通部分を記述するファイル

```
import 'bootstrap/~'; // Bootstrap の CSS 読み込み
import '@/app/globals.scss'; // 全体の CSS 読み込み

// フォント読み込み
const NotoSansJP = Noto_Sans_JP({
  /* ~略 ~ */
});

// ページに関する情報(タイトル・説明文など)
export const metadata = {
  /* ~略 ~ */
};
```

骨組みができたのでトップページの内容を書いていきます。App Router では URL に対応する場所のファイルを自動でページ内容として読み取ってくれます。リスト 2 がその内容です。

リスト 2: ページ内容を記述するファイル

```
export const metadata = {
  title: 'Top'
};

// トップページ
export default function Home() {
  return (
    <>
  );
}
```

これでもうトップページが完成しました。次はいよいよブログシステムの作成です。

### 3.3. ブログシステムの構築

#### 3.3.1. API の作成

ブログのデータ(記事内容・タイトル・カテゴリ...)は microCMS で管理します。なのでブログ側からそのデータにアクセスする手段が必要となります。

そのためにまずは microCMS の管理画面にログインし Web API[\*13]を作成します。



そして作成した API にアクセスするために microcms-js-sdk を使います。

リスト 3 では API のクライアントを作成し、別ファイルから使いやすくしています。

#### リスト 3: API クライアントを作成したファイル

```
import { createClient } from 'microcms-js-sdk';

export const microCMSClient = createClient({
  // API にアクセスするための鍵
})
```

あとは microCMSClient の get メソッド[\*14]を 使うことでデータの取得を行うことができます。

\*[13] Web 上でアクセスしてデータを取得・作成したり処理を行わせたり様々なことをするもの。今回は特にデータの取得のために使う

\*[14] メソッドとは方法・筋道などの意味を持ち、プログラミングにおいては処理をまとめたもの、つまり処理の方法のこと

#### 3.3.2. 記事ページの作成

さて、下準備も終わりいよいよ本命の記事 ページの作成です。

記事のページなので最低限タイトルと内容は必要ですが、それだけだと味気ないので機能としてカテゴリー・タグを追加してタイトルの下に表示させることとし、ついでに目次も作ります。

さらに情報がいつのものか分かった方が読者も安心できると思うので作成日時・最終更新日時も表示させます。

記事の取得に必要な ID は URL で指定します。

App Router はフォルダ名に[名前]を使うとその [名前] の部分を取得してプログラム内部で使えるようにしてくれる所以これを用います。

リスト 4 が実際のコードです。

#### リスト 4: 記事ページのファイル

```
// ブログページ(id 取得)
export default async function BlogPage({ params: { id } }) {
  // 記事データ
  const post = await microCMSClient.get({
    // 取得した ID
    contentId: id,
    // 図 1 の時に作った API のエンドポイント欄
    endpoint: 'blogs'
  });

  // 目次
  const tableOfContents = createTableOfContent(
    post.content
  );
}
```

ここで記事自体のページが完成しましたが、このままでは URL を共有しない限りアクセスは至難の業です。(ID がランダム英数字のため) そのためには記事一覧を作成します。

#### 3.3.3. 記事一覧の作成

microCMS の API は ID を指定しない場合データをリストでまとめて取得することができる所以これを活用します。

要は記事ページのデータ取得部分を少しいじ

るだけではほぼ出来上がります。

リスト 5 は記事一覧ページのソースです。

#### リスト 5: 記事一覧ページのファイル

```
export default async function BlogHome() {  
    // 記事データのリストを取得  
  
    const blog = (await microCMSClient.getList({  
        endpoint: 'blogs'  
    })).contents; // リストは contents から配列を取得  
  
    return (  
        // ...  
    );  
}
```

#### 3.3.4. カテゴリー・タグの実装

さて、これで記事とその一覧を表示する部分を作ったわけですが、これを「データ」とその一覧を表示、と解釈するとカテゴリー・タグも同様に実装することができます。

違いとしては、それぞれのカテゴリー・タグのページではそれらを持つ記事の一覧を表示させてるので記事一覧がほぼすべてのベースになる、といったところでしょうか。

ではリスト 6、リスト 7 に実装していきます。

#### リスト 6: カテゴリー一覧ページのファイル

```
export default async function Categories() {  
    // カテゴリーリストを取得  
  
    const cats = (await microCMSClient.getList({  
        endpoint: 'categories'  
    })).contents;  
  
    return (  
        // ...  
    );  
}
```

#### リスト 7: カテゴリーページのファイル

```
export default async function CategoryPage({ para  
ms: { id } }) {  
    // カテゴリー情報(=名前)を取得  
  
    const cat = await microCMSClient.get({  
        contentId: id,  
        endpoint: 'categories'  
    });  
  
    // カテゴリーに属する記事リストを取得  
    // ...  
}
```

```
// 左下から続く  
  
return (  
    // ...  
  
    <h1> '{cat.name}' カテゴリーの記事一覧  
</h1>
```

タグも似たような実装となります。カテゴリーと違い複数指定できるため記事の取得条件が「タグと一致」ではなく「タグリストに含む」になるぐらいでしょうか。

リスト 8、リスト 9 がコードですが実際リスト 6、リスト 7 とそこまで違いがないのが分かります。

#### リスト 8: タグ一覧ページのファイル

```
export default async function Tags() {  
    // タグリストを取得  
  
    const tags = (await microCMSClient.getList({  
        endpoint: 'tags'  
    })).contents;  
  
    return (  
        // ...  
    );  
}
```

#### リスト 9: タグページのファイル

```
export default async function TagPage({ params: {  
    id  
} }) {  
    // タグ情報(=名前)を取得  
  
    const tag = await microCMSClient.get({  
        contentId: id,  
        endpoint: 'tags'  
    });  
  
    // タグに属する記事リストを取得  
  
    const blog = (await microCMSClient.getList({  
        endpoint: 'blogs',  
        queries: {  
            // ...  
        }  
    }));  
  
    return (  
        // ...  
    );  
}
```

これで記事へのアクセスがしやすくなりました。次はいよいよ記事の公開です。

### 3.4. デプロイ

システムの開発も一段落したのでいよいよ Vercel で公開させる作業、デプロイを行います。

と言っても GitHub ヘソースコードをアップロードするだけで Vercel が自動で検知して公開まで全部やってくれるのでそこまで苦労はしません。

Vercel で自分の GitHub アカウントを連携すると Vercel 上でプロジェクトを作成する際に GitHub からソースコードを持ってくるようにすることができます。

プロジェクトの設定は初期状態で問題ないのでそのまま作成。

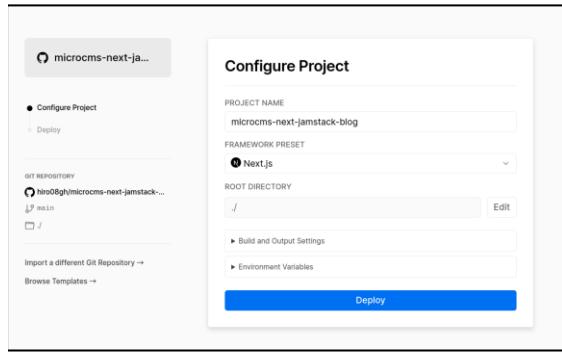
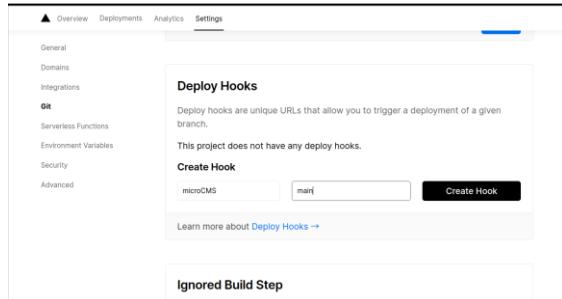


図 2 のボタンに「Deploy」と書いてある通りプロジェクトを作成した時点で 1 回デプロイされます。

問題がなければ約 1 分ほどで自動的に公開されます。

このままだとブログシステムを更新した時にしか記事のデータを取得しないので microCMS 側でデータを変更した場合にもデプロイさせます。

そのためにはまず Vercel で Webhook[\*15]を作成します。



次に作成した Webhook の URL を microCMS 側に登録します。

これで記事を書いたり消したりしてもしっかりと

数分で反映されるようになります。



図 4. Webhook 設定画面

[\*15] Web API の逆とも言える物で、Web API がデータを与えてくれる物とすると Webhook はデータを与えられて動く物です

### 4. 動作確認

さて、ここまでシステムを組んできたので実際に動作するか確かめましょう。

これはブログなので動作確認はもちろんテスト記事の投稿とします。

テスト記事を投稿すると 3.4 で作成した Webhook によって自動でサイトが更新されるので 1 分ほど待ってからアクセスします。



図 5. ホームページ  
上のメニューから Blog、記事一覧と進みます。



投稿したテスト記事が見えました。

早速見てみましょう。



無事にテスト記事を確認することができました。  
目次も問題なく生成できています。

## 5. 完成した感想

特につまづくこともなく無事に動いてくれたの  
でよかったです。

実はサイトを作ったことは既にあって、私含め  
数人で活動してるゲーム開発集団のホームページを  
VPS で運営しています。今回、それとは  
別の方向性での Web 開発ができてとても刺激  
が得られました。

また、Next.js v13 の App Router が実装されて  
間もないということもあり、知らない間に便利な  
機能が多数追加されていたことなど多くの知見  
を得ることができました。

まだまだ改善点はたくさんありますが、今後は  
この技術ブログを改善しながらここにぼちぼち  
アウトプットしつつもっと Web 技術を究めようか  
と思います。以上です。

## 6. 参考文献

microCMS + Next.js で Jamstack ブログを作っ  
てみよう | microCMS ブログ

<https://blog.microcms.io/microcms-next-jamstack-blog/>

Next.js でカテゴリー機能を実装してみよう |

microCMS ブログ

<https://blog.microcms.io/next-category-page/>  
microCMS のリッヂエディタで取得できる値を、

いい感じに処理するライブラリの紹介

[https://zenn.dev/d\\_suke/articles/e18352797bb](https://zenn.dev/d_suke/articles/e18352797bb)  
e00bdabb6

microCMS で目次を作成する

<https://blog.microcms.io/create-table-of-contents/>

## 7. 参照

リポジトリ

<https://github.com/ms0503/tech-blog>  
技術ブログ

<https://ms0503-tech-blog.vercel.app>  
実際のテスト記事

<https://ms0503-tech-blog.vercel.app/blog/tvvy60x1kcv>  
microCMS

<https://microcms.io>  
Vercel  
<https://vercel.com>



# JavaScript でカレンダーを制作する

## 1. はじめに

この文章では「Visual Studio Code」というコードエディタを利用したプログラミングについて説明します。今回私は、このコードエディタを利用して簡単なカレンダーを制作しました。使用した言語は JavaScript で、マークアップ言語に HTML、スタイルシート言語に CSS を活用しました。

## 2. 実際の動作とプログラム

### 2.1. 動作解説

プログラムを実行した画面が図 1 です。[>>]をクリックすると翌月のカレンダーが表示され、[<<]をクリックすると先月のカレンダーが表示されます。[Today]を押すと今日の日付が含まれる年月のカレンダーが表示され、テキストボックスに任意の年を入力していくつかのキーを押すとその年の 1 月のカレンダーが表示されます。



図 1 実際の画面

### 2.2. カレンダーを表示するプログラムの解説

リスト 1 はカレンダーを表示するときのアロー関数です。このアロー関数では HTML の table 要素の中

ロボティクスコース 3 年 大高 康介

の tbody 要素に、カレンダー 1 週間分の日付を格納した tr 要素を子要素としてその月の週数だけ追加します。

例えば 2023 年 9 月の場合は週数が 5 なので、tbody 要素に 5 つの tr 要素が追加されることで図 1 のようなカレンダーが表示されます。

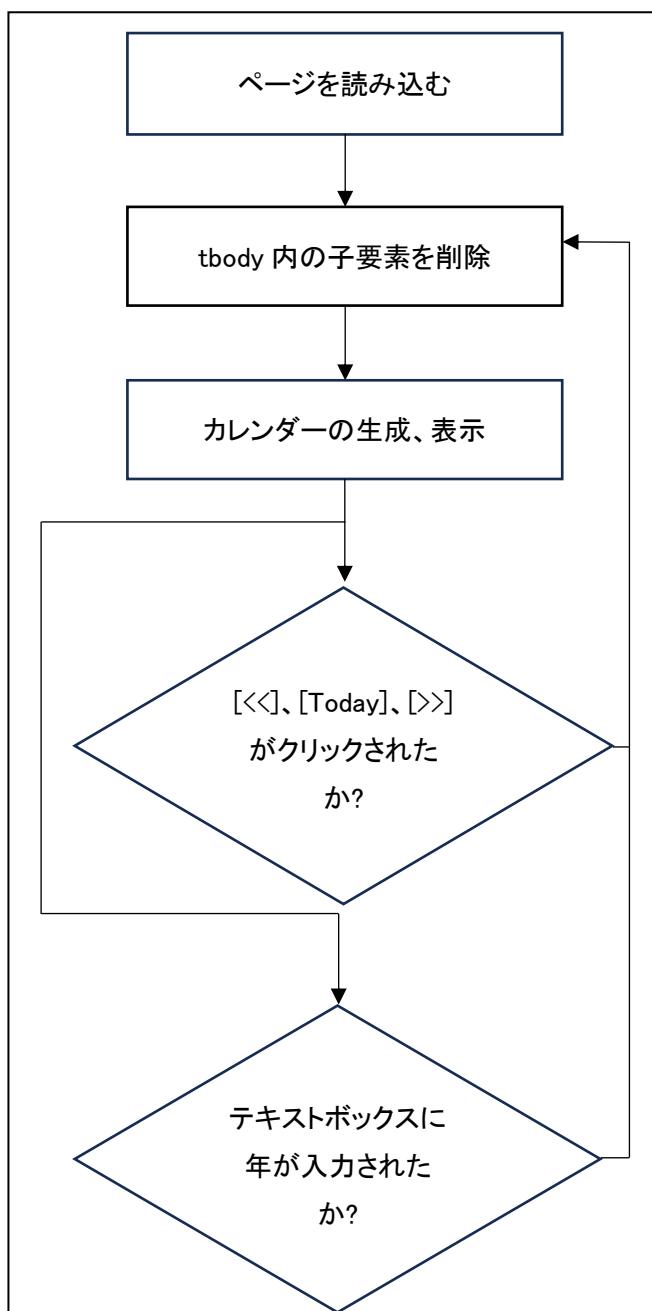
### リスト 1 アロー関数

```
//カレンダー1ヶ月分の日付が格納されている配列  
を dates として渡す  
const setWeeks = (dates) => {  
    //1週間の日付を格納する空の配列  
    const weeks = [];  
    //カレンダーの週数を計算  
    const weeksCount = dates.length / 7;  
    //weeks に weeksCount 回 1週間分の日付を  
    //格納  
    for (let i = 0; i < weeksCount; i++) {  
        weeks.push(dates.splice(0, 7));  
    }  
    //weeks に含まれる要素 1 つを week として、そ  
    //の要素数だけ td を生成、内容を変更した後にそ  
    //れらを tr に追加して tbody に追加する  
    weeks.forEach(week => {  
        const tr = document.createElement('tr');  
        week.forEach(date => {  
            const td = document.createElement('td');  
            td.textContent = date.date;  
            //date のスタイルを決定する関数  
            setDateStyle(date);  
            tr.appendChild(td);  
        });  
        tbody.appendChild(tr);  
    });  
};
```

### 3. プログラムの流れ

図 2 はカレンダーを表示する際の処理の概要です。Load イベントによってページが読み込まれると tbody 要素から子要素をすべて削除して現在の年月のカレンダーを表示します。その後、[<<] や [Today]、[>>] をクリック、またはテキストボックスに任意の年を入力すると現在表示されているカレンダーの tbody 要素から再度子要素を削除して tr 要素を tbody 要素に追加することでカレンダーを再表示します。

図 2 プログラムの流れ



### 4. まとめ

私は今回 JavaScript で簡単なカレンダーを制作しましたが、一番苦労した箇所は取得したカレンダーの日付を実際にウィンドウに表示することでした。あまり詳しく説明することができませんでしたが、一週間ごとに日付を分割して新しい配列にするまでにとても手間取りました。また、今回は祝日を考慮せずにカレンダーを作成したので、実際のカレンダーとは少し異なるものとなっています。様々な WEB サイトを参考に試行錯誤してカレンダーを完成させましたが、急ごしさの中途半端なものとなってしまったので機会があればもう一度カレンダーを作成してみたいです。

現在は JavaScript を中心にプログラミングの学習をしているので、もっと効率よく性能の良いものが作れるように更に学習を進めていきたいと思っています。最後まで読んでいただきありがとうございました。

### 参考文献

- 「JavaScript ガイド - MDN Web Docs」  
<https://developer.mozilla.org/ja/docs/Web/JavaScript/Guide>
- 「TechAcademy」  
<https://magazine.techacademy.jp/magazine/>

# Unity でオセロゲームを作成する

## 1. はじめに

私は今回 Unity というゲームを作成できるアプリケーションを使用し、オセロゲーム製作を行いました。開発環境として Microsoft Visual Studio(バージョン1.7.3.4)、Unity(バージョン 2021.3.1f)です。この記事を最後まで読んでいただけたら幸いです。

## 2. 実際の動作とプログラム

### 2.1. 動作解説

プログラムを実行した初期の盤面が図 1 です。右上に現在のターンを左上に石の数を表示し、置くことができる盤の位置の色を変更しています。ルールは一般的なオセロと同じなので、それぞれの処理のプログラムを解説していきます。

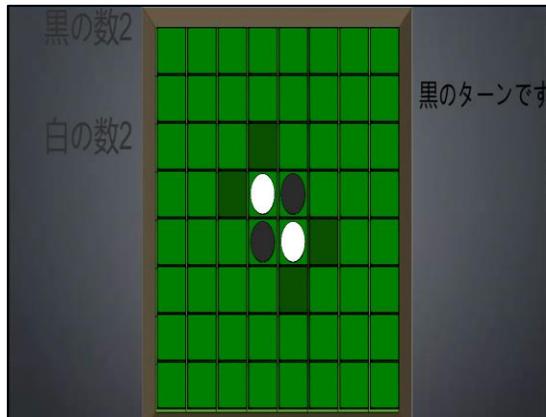


図 1 実際の画面

### 2.2. ゲーム始めの解説

始めにゲーム開始時に駒を置くための基盤を  $8 \times 8$  作成します。この基盤には子オブジェクトとして黒と白の石を格納します。子オブジェクトとは格納されたゲームオブジェクトを親オブジェクトとして扱い、子オブジェクトは親オ

ロボティクスコース 3 年 鈴木 佑  
プロジェクトの様々な状態を同期するオブジェクトとなります。その中には親と子の位置関係も自由に設定できるため、子オブジェクトとして設定することで後述の石の状態(黒、白、空白)の変更が行いやすくなります。状態が黒の場合は黒の石、白の場合は白の石を同じ位置に表示します。この基盤をクリックした時に石を置く関連の処理を行うようにし、関連の処理で色を変更しています。

### 2.3. 石のプログラム解説

ここから石のプログラムについて解説します。  
初めに石の設置とひっくり返す処理です。  
オセロの石が置ける箇所は石が隣接しているかつ、8 方向(右、左、上、下、右上、右下、左下、左上)に置く石と同じ色がある場合に置くことができ、8 方向置いた石と同じ色の石の間にある相手の石の色をひっくり返します。今回私が考えた処理は、基盤をクリックした場合に、まずクリックした基盤の位置と置く石の色を取得します。その後、置く予定の石の色があるかどうか、基盤の端になるまで 8 方向分探索します。例として、上方向の探索についてのプログラムをリスト 1 に示します。上の探索の場合は縦座標の  $y$  が範囲外になるまで、1 回ごとに  $y$  を 1 増やすループ処理を行います。 $y$  が 1 増えると調べる基盤が 1 つ上に行きます。7 行目で、指定された基盤の状態(白、黒、空白)を判別し、9~19 行目でそれぞれの処理に移っています。9~11 行目の空白の場合の処理は、石を置くことができないので、その時点でループを中止しています。12~22 行目石がある場合はまず、現在のターンが白か黒

で置く色を判別し、色が違う場合は石が同じ色が見つかるまで上方向のループをそのまま続けます。色が同じ場合は、19 行目の関数を実行し探索で見つかった同じ色の石の座標がクリックした基盤の座標になるまで、逆にマイナスをループし 1 回のループごとに間にある石の色を変更しています。なお、オセロは同じ色が隣接している場合は置くことができないので、16~18 行目に例外として処理しています。これを 8 方向分繰り返し、判定しています。

### 3. 終わりに

私は今回、Unity でオセロゲームを作成しました。高専祭の直前にプログラミングコンテストがあり、その調整に時間が割かれてしまったため、当初の目的だった対戦 AI の作成ができませんでした。ちなみに、オセロの対戦 AI はマス目ごとに点数をつけ、何処に置いたら不利、有利かを判別して石を置いています。なので、評価関数の数値次第で AI をどんどん強くすることができます。このオセロゲームは高専祭で展示しているため、もし展示までに AI の作成が間に合っていたら、ぜひ対戦してみてください。最後までお読みいただきありがとうございました。

### 参考文献

<https://lets-csharp.com/c-sharp-de-othello/>  
C#でオセロを作ってみる

### リスト 1 上方向を探索するプログラム

```
1.   for ( ; y < 9; )
2.   {
3.       y++;
4.       Statecount++;
5.       if (y == 9) {
6.           return State;
7.       }
8.       State = board.Empties(x,
9.       y, State );
10.      if (State == StoneState.Empty) {
11.          break;
12.      }
13.      if (State != put) {
14.          continue;
15.      }
16.      if (State == put) {
17.          if (Statecount == 1) {
18.              break;
19.          }
20.          ReverseYM(x, y, Statecount,
21.          put);
22.          break;
23.      }
}
```

# 弾幕結界

## 1. はじめに

今回私は Unity を用いて 2d 弾幕 STG を作成した。この記事では、その具体的な弾幕の作り方を解説していく。

## 2. はったり弾幕

### 2.1. はったり弾幕とは

最初にはったり弾幕を作る。はったり弾幕とは見た目が派手で避けるのが難しそうだが、避け方さえ知っていれば簡単に避けられる弾幕のことを言う。要ははったりが効いた弾幕の事だ。今回は大量の弾が自機に向かっているように見えるが、実はチョン避け(自機に向かってくる弾を、少しだけ自機が動いて避ける行為)で避けられる弾幕を作る。

### 2.2. 模倣からの作成

有名なはったり弾幕の式神「仙狐思念」(東方妖々夢 ~ Perfect Cherry Blossom.)を参考にして、今回は弾幕を作る。参考元の弾幕は、自機に向かう弾幕とそれに対して大量の平行に進む弾幕を放つ。弾が大量にあるが、実は真ん中の弾だけを避ければいいという見掛け倒しの弾幕だ。

作り方としてまず、真上に向かって弾を打ち続けるプレハブを作る。それをコピーして図 1 のように配置する。黒い球が弾で、親のプレハブの中に入れてある。その親には、常に自機の方向を向くリスト 1 のスクリプトを、アタッチする。こうすることで、この弾幕たちは自機に向かって発射されるようになる。この弾幕は避けるには、中央の縦列だけがプレイヤーに向かっているので、

ロボティクスコース 5年 富山 結都

中央とその隣の弾幕たちの合間に移動すれば避けられる。ここまで出来たら弾幕を派手にしよう。単純に弾の並びを増やしていくば見た目はどんどん派手になる。見た目は避けるのが難しそうだが、難易度は変わらないのだ。ただし列を増やすときは奇数列 × 奇数列でないといけない。今回は  $3 \times 3$  の列だがこれが偶数だと、自機を避けていく弾幕になる。

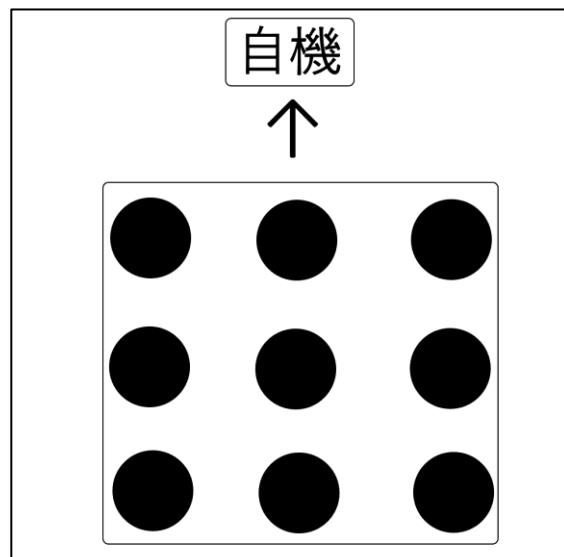


図 1 弾幕の配置

#### リスト 1 常にプレイヤーの方向を向くスクリプト

```
1. private GameObject target;;
2. void Start(){
3.     target = GameObject.Find("Player");
        //Player という名前のオブジェクトの方
        向を常に向く
4. }
5. void Update(){
6.     This.gameObject.transform.LookAt
        (target.transform.position);}
```

### 3. 回転系の弾幕

次に回転系の弾幕を作る。回転系とは敵が画面中央あたりに鎮座し、そこを中心として回転する弾幕を発射する。これを攻略するにはプレイヤーは回転しながら弾幕を避け、中央の敵のライフを削る必要がある。

今回は回転弾幕と自機外しの弾幕、二つを作る。自機外しとは自機を常に狙わず、何もしなければ被弾しない弾幕だ。ただしプレイヤーが大きく動くと被弾する、というものもある。なぜこれが必要なのかというと、単純に回転弾幕を作るとプレイヤーが、最高速度で動くことで簡単に避けられる。ここに自機外しを入れると、プレイヤーが最高速度で避ける際に、自機外しに被弾する。つまりプレイヤーに、遅すぎず早すぎない移動を求めることが出来る。

まずは回転弾幕を作る。最初に真上に向かって打ち続けるプレハブを作る。これに回転するスクリプトをアタッチする。こうすることで弾が打ち続けながら回転し、螺旋の弾幕を生み出す。回転するスクリプトはリスト 2 に書く。

次に自機外しを作る。これは今までの繰り返しで、真上に向かって打ち続けるプレハブを作る。そこにリスト 1 の常に自機の方向を向くスクリプトをアタッチする。このままではただ単に、自機を狙う弾幕になるので、このプレハブのインスペクターの回転を変える。環境によって変える軸が何処かは別だが、変えたら弾幕は自機を避けるように飛んでいく。回転の角度が狭いと、少し動いただけで被弾するので角度の微調整が必要になるだろう。ただしこのままの状態だと片側が空いているので、簡単に避けられる。なので、先ほどのプレハブをコピーし、回転の角度にマイナスをつけると、自機を挟み込む自機外しの弾幕が完成する。図 2 の自機外し弾幕は、それぞれ Y 軸の角度が 40 度、-40 度となっている。図 2 の右下にある  $\Psi$  のような見た目の物は自機で、画面中央の四角の敵から、回転弾幕と自機外し弾幕が放たれている。

#### リスト 2 回転させるスクリプト

```
1. public Vector3 pos;  
2. void Update() {  
3.     transform.Rotate  
(new Vector3(pos.x, pos.y,  
            pos.z) * Time.deltaTime);  
4.     Application.targetFrameRate = 150;  
5. }
```

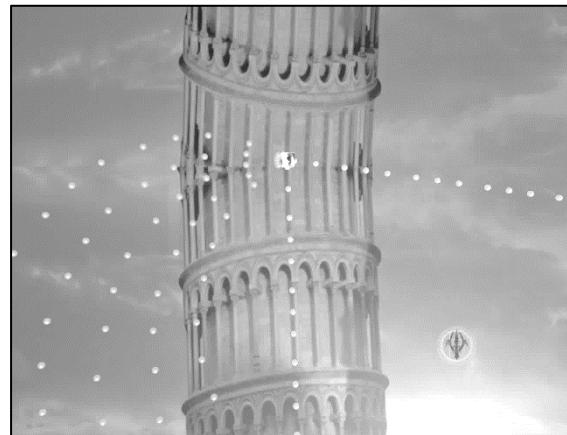


図 2 実際の回転弾幕と自機外し弾幕

#### 4. おわりに

今回はプログラミングをほとんどしていない。昔作ったゲームから少し変えたプログラムを持ってきたからだ。そもそもプログラミングを忘れていたので何とか誤魔化そうと思ったのだ。しかしゲームはプログラミングだけでない、数字や配置を変えるだけで、同じプログラムを使いまわしても違うゲームが作れる。最近はプログラムが書けなくても、ゲームをある程度作れると思う。だからこそゲームはアイデアが命だ、と心に言い訳をしながらこれを書いている。実際作れたので、気楽にこれからもゲームを作っていくたい。

#### 参考文献

CodeGenius

<https://codegenius.org/>

Unity

<https://unity.com>

# いざ、Nim を学ばん

## 1. はじめに

今回はプログラミング言語 Nim について、基本的な構文とそれらの応用を備忘録的にまとめました。少しでも Nim に興味を持ってくれたり、これから Nim を学ぼうとしている人の一助となれば幸いです。

## 2. Nim について

Nim(図 1)は 2005 年に開発が開始された、比較的新しいプログラミング言語です。以下のような特徴を持っています。

- 静的型付け・コンパイル言語
- Python に似た構文
- C 言語並みのパフォーマンス
- 自己完結型

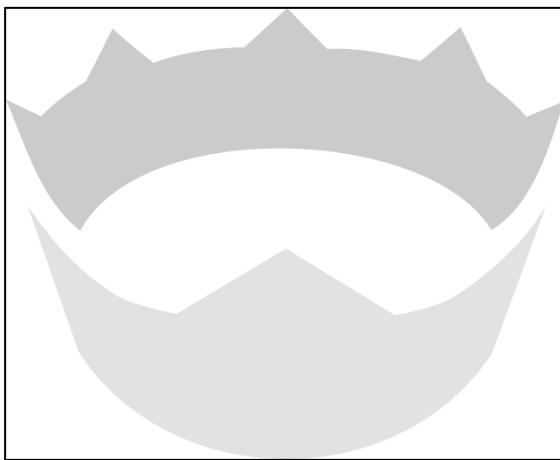


図 1 Nim のロゴ

## 3. 基本構文

### 3.1 Hello World

リスト 1 は「Hello, World!」を出力するだけのプログラムです。

コンパイルと実行をおこなうにはリスト 2 のコマンドを実行します。

ロボティクスコース 5年 鈴木 晴斗

#### リスト 1 Hello, World!の出力

```
1. echo ("Hello, World!")
```

#### リスト 2 コンパイルと実行

```
1. nim c -r hello.nim
```

## 3.2 変数と定数

変数と定数はリスト 3 のように宣言します。

`var` は変数を宣言するときに使います。変数 `a` のように変数名のあとにセミコロンをつけ、そのあとに型名を書きます。Nimコンパイラには Rust や Go といった他の言語のように型推論の機能があります。そのため変数 `b` のように型名を省略することも可能です。

`let` も変数の宣言をするときに使います。`var` とは違い、宣言をするときに初期化を行う必要があります。宣言した値は実行時に確定します。そのため一度宣言した値を変更することはできません。

`const` は定数を宣言するときに使います。宣言した値はコンパイル時に確定します。`let` と同じように値を変更することはできません。

#### リスト 3 変数と定数の定義

```
1. var a: int = 1
2. var b = 2
3. var                      # まとめて宣言
4.           name = "taro"
5.           age = 20
6.
7. let d = "Apple"
8. let e = "Orange"
9. d = d & e                  # エラー
10.
11. const s = "Hello, " & "World!"
```

### 3.3 基本的なデータ型

Nim の基本的なデータ型は次の通りです。

- int, int(8, 16, 32, 64)
- uint, uint(8, 16, 32, 64)
- float, float(32, 64)
- bool
- char
- string
- cstring

### 3.4 入力と出力

入力を行うにはリスト 4 のように行います。コンソールから得られる入力は文字列型なので、場合によって他の型に変換する必要があります。

#### リスト 4 入力を受け取る

1. # string
2. let a = stdin.readLine
- 3.
4. # string -> int
5. let b = stdin.readLine.parseInt
6. b += 1
- 7.
8. # string -> float
9. let c = stdin.readLine.parseFloat

出力を行うにはリスト 5 のように行います。  
echo は引数を自動で文字列に変換します。  
そのため 3 行目の引数が数値型の場合でも機能します。また echo はプロシージャ(複数の処理をまとめて、外部から呼び出し可能にしたも)のであるため、()を省略することもできます。

他には 7 行目のような方法で出力することもできます。

#### リスト 5 出力をする

1. echo("abc")
2. echo( 12345 )
3. echo( "Apple", "Orange", "Grape" )
4. echo "abc"
- 5.
6. writeLine( stdout, "Nim" )

### 3.5 配列

リスト 6 のように array を使うことで静的配列(固定長配列)を宣言することができます。

#### リスト 6 array を使って静的配列の宣言

1. var arr: array[5, int]
2. arr = [1, 2, 3, 4, 5]
- 3.
4. var arr2: array[1..3, string]
5. arr2 = ["Apple", "Orange", "Grape"]

リスト 7 のように seq を使うことで動的配列を宣言することができます。

#### リスト 7 seq を使って動的配列の宣言

1. var seq1: seq[int]
2. seq1 = @[1, 2, 3, 4, 5]
3. seq1.add(6)
- 4.
5. var seq2 = newSeq[int](5)

### 3.6 ループ

リスト 8 のように for や while を使うことでループ処理ができます。

#### リスト 8 ループ処理

1. # 1 から 10
2. for i in 1..10:
3.     echo i
- 4.
5. # 1 から 9
6. for i in 1..<10:
7.     echo i
- 8.
9. var i: int = 0
10. while i < 10:
11.     echo i
12.     i += 1
- 13.
14. while true:                           # 無限ループ
15.     echo "Nim"

また、リスト 9 のように使うことで配列を走査することもできます。

### リスト 9 for を使った配列の走査

```
1. for value in arr:  
2.     echo value  
3.  
4. for index, value in arr2:  
5.     echo index, ":", value
```

## 3.7 条件分岐

リスト 10 のように if ステートメントを使うことで条件分岐ができます。

### リスト 10 条件分岐

```
1. for i in 1..10:  
2.     if i == 3:  
3.         break  
4.     elif i == 5:  
5.         continue  
6.     else:  
7.         echo i
```

## 3.8 関数

リスト 11 のように proc ステートメントを使うことで関数を定義することができます。

1 行目の `printMessage()` は引数も返り値もない関数です。ここでは返り値の型として `void` と書いています。書かない場合でも暗黙的に返り値のない関数と見做されるため、省略することも可能です。

2 行目の `getLength()` は引数に文字列型、返り値に数値型をとる関数です。引数 `s` の長さを `s.len` で取得し `return` で返します。Nim では関数の引数はデフォルトで不变なので注意が必要です。変更可能な引数にするには `var` を引数に追加します。

3 行目の `isEven()` では返り値の型を書いているにもかかわらず、`getLength()` のように `return` を使っていません。Nim では関数のブロック内に暗黙的に `result` 変数を持っています。そのため関数内で `result` という変数に値を代入すると、`return` を書かなくても自動で返り値を `return` してくれます。

4 行目の `getFrontChar()` では `return` も `result` 変数への代入もありません。Nim では最後に書かれた変数(この場合変数 `c`)が暗黙的に返されます。

### リスト 11 関数の定義

```
1. proc printMessage(): void =  
2.     echo "Min Min Nim Nim"  
3.  
4. proc getLength(s: string): int =  
5.     let len: int = s.len  
6.     return len  
7.  
8. proc isEven(n: int): bool =  
9.     if n mod 2 == 0:  
10.        result = true  
11.    else:  
12.        result = false  
13.  
14. proc getFrontChar(s: string): char =  
15.     let c: char = s[0]  
16.     c
```

## 4 応用

### 4.1 数当てゲーム

入力した数が答えの数より大きいか、小さいかを頼りに数をあてるゲームです。リスト 12 がそのコードです。

答えは標準ライブラリの 1 つである `random` を import し、1 から 100 の間で乱数を生成します。実行するごとに違った値になるように、5 行目の `randomize()` でデフォルトの乱数生成器を初期化しています。

8 行目からは `while` で無限ループを回し、プレイヤーからの入力を受け付けます。標準ライブラリの `strutils` を import し、`parseInt()` を使うことで入力されたものを数値型に変換します。`if` 文で入力値と答えの大小を比較し、該当するメッセージを出力しています。答えと同じ値が入力されたときはメッセージを表示した後に、`break` を使いループを抜け出します。

### リスト 12 数当てゲームのコード

```
1. import std/random
2. import strutils
3.
4. # 亂数生成器の初期化
5. randomize()
6. let answer = rand(1..100)
7.
8. while true:
9.     let input = parseInt(readLine(stdin))
10.    if input < answer:
11.        echo "Too small"
12.    elif input > answer:
13.        echo "Too big"
14.    else:
15.        echo "Correct!!"
16.        break
```

## 4.2 エラストテネスのふるい

エラストテネスのふるいとは、古代ギリシアの科学者エラストテネスが考えたとされる、N 以下のすべての素数を発見するための方法です。

方法は簡単です。最初に 2 を選び、それ以外の 2 の倍数にチェックをしていきます。次に 3 を選び 3 の倍数に、その次は 5 を選び…と続けていき、最終的にチェックがついていない数が素数となります。

リスト 13 は 2 から 2023 までの素数を求めるコードです。2 から 2023 までの bool 配列を用意し、すべての要素を `true` で初期化します。次に `for` で 2023 まで回し、`i` の倍数なら `isprime[i]` を `false` にします。最後に、`isprime` を走査し、`true` なら `index` を出力します。

### リスト 13 エラストテネスのふるい

```
1. let N: int = 2023
2. var isprime: array[2..2023, bool]
3.
4. for i in 0..2023:
5.     isprime[i] = true
```

```
6. for i in 2..2023:
7.     if not isprime[i]:
8.         continue
9.     var j = i * 2
10.    while j <= N:
11.        isprime[j] = false
12.        j += i
13.
14. for index, value in isprime:
15.     if value:
16.         echo index
17.     else:
18.         continue
```

## 5. おわりに

今回初めて Nim について勉強しました。Python を少し触ったことのある私にとっては、Nim のコードは特段躊躇することもなく、可読性の高いコードを書くことができました。過去に Rust の冗長さを前に挫折した経験があるので、簡潔に早く動作するコードを書けるのはとても魅力的に感じました。

一方で日本語のドキュメントが少なく、欲しい情報にたどり着くのに少し苦労しました。公式サイトにあるチュートリアルなどを翻訳しながら進めることがおすすめします。

今回の勉強で Nim のすべての機能を網羅したわけではないので、これからも Nim についての勉強を、特に並列処理の実装やゲーム開発などについてトライしていきたいです。

## 参考文献

Nim

<https://nim-lang.org/>

NimWorld

<https://2vg.github.io/Nim-World/>

[Nim] 個人的逆引きリファレンス

<https://flat-leon.hatenablog.com/>

# 第 34 回全国高等専門学校 プログラミングコンテスト競技部門

大高 康介 ロボティクスコース 3 年

鈴木 佑 ロボティクスコース 3 年

佐藤 至 ロボティクスコース 4 年

## 1. 始めに

私たちソフトウェア研究部会は毎年、全国高等専門学校プログラミングコンテスト(以下、プロコン)に参加しています。今年は福井県越前市で開催されました。プロコンには競技部門、課題部門、自由部門 の3部門があり、今年は競技部門に参加しました。競技部門では、運営委員会が用意した競技に対して各自が用意したプログラムで、他校と早さや正確さを競い合います。

## 2. ルール

### 2.1. フィールドの各属性

今回の競技部門は 2 チーム対戦型の陣取りゲームでプロコン側が指定したフィールドを基に最終的にポイントの多いチームが勝者となります。具体的に図 1 を用いて説明します。フィールドの属性は 6 種類あり、順番に解説していきます。まず、フィールドにある○は職人でプレイヤーはこの職人を操作し、ゲームを進めていきます。職人の役割は 8 方向(上、右、左、下、右上、右下、左上、左下)のマスに移動する「移動」、後述の城壁を 4 方向(上、右、左、下)のマスに建設する「建築」、城壁を破壊する「解体」があります。職人が上記いずれかの行動を行った場合、ターンが終了するまで、その職人は行動することができません。2 つめは池で、

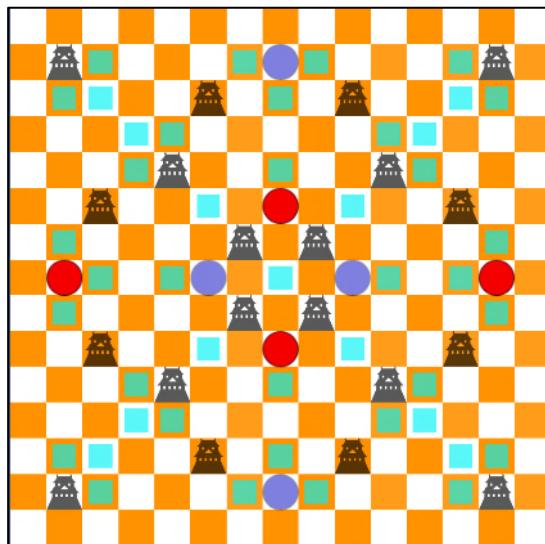


図 1 実際の画面

マスの中にある口が池となります。フィールドに初期状態からあり、池があると職人は移動することができません。3 つめと 4 つめは城壁と陣地です。城壁は各職人が 1 ターンに 1 度建設することができ、どちらのチームの職人が建築したかによって色が変化します。城壁が設置してある箇所は職人が移動できず、城壁は職人が破壊することができます。陣地は同じチームの城壁が囲んだマスが陣地となります。陣地は城壁と異なり破壊することができません。5 つめは城です。城は池と同じ様にゲームの初期状態から設置されており、城を陣地にした場合は通常の陣地ポイントよりも高くポイントがもらえます。6 つめは中立で上記いずれかの属性を含めない状態を指します。

このルールは城>陣地>城壁の順番でポイントが定まっており、ターン数は 30~200 のランダムで決まり、職人が2~6、フィールドは11~25の正方形の形となっています。

## 2.2. 解法アルゴリズム

この節から実際の解法アルゴリズムについて解説していきます。まず今回の競技部門はより多くのポイントを取ったチームが勝利する為、職人にはなるべく多くのポイントを取得できるように指示を出すのが理想です。そこで鍵となってくるのが、城です。城はフィールドの大きさ関係なく、安定してポイントが多く取得できます。そこで、Mini-max 法を用いて計算を行い、最善手を選択させる形になりました。Mini-max 法はオセロや将棋のような盤面の情報を参照するゲームによく用いられる方法で、お互い最善手を指す事を前提とし、深さを定め、行動を決めます。深さとは、Mini-max 法が先読みを行う手法であり、理想であれば、全ての手の計算を予め行う事なのですが、そうした場合計算量が莫大になってしまふため、先読みに制限を設ける必要があります。そこで、どこまでのターンまで読むかを制限したものが深さとなります。始めに調べる深さを定め、現在の地点で味方の職人が行える行動を全て調べた後、深さを 1 進め、相手のターンに同じ様に相手の職人が行える行動を全て調べます。これを最初に定めた深さまで繰り返し、定めた深さになった場合、評価関数を基に取得できるポイントを計算します。この評価関数はポイントを計算する際に予め決めておくもので、今回は配列で管理し、城の周辺を大きくして絶対に陣地にならない端のマスは低くしています。お互い最善手を尽くすことを前提としているので、深さが相手のターンの場合、一番

小さい値(mini)自分のターンの場合は一番大きい値(max)とし、最善手を定めます。

これをターンが来るごとに繰り返していき、ゲームを進めていきます。

## 3. 終わりに

今回の競技部門はファイナルステージに出場することができませんでしたが、この開発は良い経験になりました。プロコンの公式サイトには競技部門の大会のアーカイブが残っている YouTube のリンクが有るのでぜひ見てみて下さい。ここまで読んで下さりありがとうございました。

## 参考文献

高専プロコン公式サイト

<http://www.procon.gr.jp/>

オセロゲーム開発 ~ミニマックス探索法~

<https://uguisu.skr.jp/othello/minimax.html>

今年度の競技部門開発リポジトリ

<https://github.com/Sofken-natori/Procon>

# プロコン旅行記

ロボティクスコース 3年 鈴木 佑

## 1. 始めに

このプロコン旅行記は、10/14,10/15 に福井県で開催された第 34 回プログラミングコンテスト(以下プロコン)の旅行記です。プログラミング等の専門的な話は無くしているので、軽い気持ちで読んでいただければ幸いです。

## 2. 出発からプロコン会場まで

### 2.1. 10/12 夜行バスで移動

今回のプロコンは新幹線での移動も検討しましたが、台風での運休も考え、夜行バスで移動となりました。10/13 にエントリーが必要なため、前日の 22:30 に仙台駅に集合し、出発しました。今回、顧問の北島先生も含めて 4 人で参加しましたが、全員が夜行バス未経験で、誰も夜行バスの事前情報無しで乗り込みました。実際の写真を図 1 に載せます。…思ったより狭い。



図 1 夜行バス車内

実際に乗った経験がある人なら分かると思いますが、とても寝づらいです。車の振動、走行音、微妙に付いている明かりもあって中々寝付けず、半寝半起き状態で朝 8、金沢駅に到着しました。もし、夜行バスに乗る機会がある人がいるなら、アイマスクか耳栓を持っていくことをお勧めします。

### 2.2. 10/13 移動日と観光

そんな状態で朝 8 時に金沢駅に到着しました。ここから 14:00 の特急新幹線に乗り、福井県に移動するのですが、まだ時間があるため、各自自由行動となりました。私は取り敢えず、朝食を取ろうと思い、朝食を取れる所を探しました。金沢駅は中々の大きさを誇り、飲食店や、お土産屋等、色々なお店が並んでいました。ただ、時刻は朝 8 時で開けている店があまり無く、腹も減っていたため、結局近くのマックに行くことになりました。自分は、朝マックが初めてで、ソーセージマフィンを頼みましたが普段食べる昼のマックとは違い美味しかったです。

その後、金沢駅周辺を見物し、9 時にお土産を幾らか購入しました。そこから、時間まで各カフェを転々としながら、プロコンの調整をしていました。昼食は、金沢駅にあるラーメンを食べました。麺の細さが、濃厚なスープと絡んでとても美味しかったです。

昼食を食べ終わった後、集合場所に行き、福井県武生駅行きの特急に乗りました。新幹線で 1 時間位乗り、2 日間宿泊するホテル「ホテルクラウンズ武生」のある武生駅に到着しました。ホテル(図 2)は駅の目の前にあり、直ぐ

にチェックインしました。



図 2 宿泊したホテル

ホテルの室内は、プロコン大会運営が用意したホテルということもあり、広かったです。さらにサービスとして、無料ドリンクや、漫画、中にはお茶漬けなど、サービスも豊富でした。

そこから、プロコン会場の「サンドーム福井」のあるサンドーム西駅に乗りました。と言っても武生駅から 1 駅位すぐに到着したのですが、そこから会場までが遠い。最寄り駅から会場までが 1.5km ほど、徒歩で 20 分かかりました。せめてシャトルバスでも運営側が出て欲しかった…。と皆で文句を言いながら会場へ向かいました。

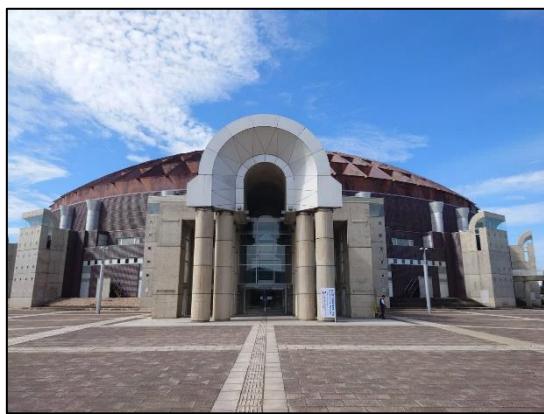


図 3 サンドーム福井

会場(図 3)へ到着した後は、前日は受付のみでしたが、会場には参加している高専生が多く、緊張感がありました。ホテルへ帰宅した

後は、夕食を取れる場所を探し、近くの定食屋でチャーハンを頂きました。唐揚げもサクサクでおいしかったです。その後、ホテルに戻り、自分の部屋でプログラムの最終調整を行った後、就寝しました。

### 3. 10/14 1 日目

#### 3.1. 移動

当日は午前 6:00 に起床し、朝食はホテルのバイキングで食べました。当日必要な持ち物を確認し、7:20 にホテルを出発して、8:00 に会場へ到着しました。

会場につくと、それぞれ指定された席に座り、開会式を見ました。午前 10:00 には動作確認である模擬戦が行われるため、当日に渡された試合情報を確認しつつ、準備を行いました(図 4)。

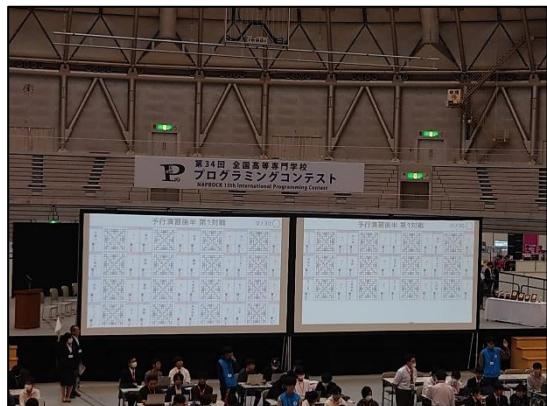


図 4 実際の会場の様子

#### 3.2. 本戦

大会側のサーバーの通信を行えるかどうかの確認を行った後、第一試合に入ります。私達のチームは持ち込んだ 3 つのパソコンのうち 2 つが直前で使えなくなるなど、トラブルはありましたが、最終的には予定通りに実行しました。その後は、チームで対戦のフィールドの確認と調整を行い、昼になったら運営側から購入した弁当を頂きました。おかげの種類が多く、美味しかったです。そして 12:30 に本戦

の為、ステージに行きました。今回のプロコンは 1 対 1 の陣取りゲームで、決められたターンでポイントをより多く取ったチームが勝利します。本戦は第一試合と第二試合の 2 試合連続で行い、第一試合では都立(品川)高専さん、第二試合は東京高専さんとの試合でした。結果としてはどちらの試合もエラーで職人(駒)が動かず、2 試合とも敗北し、翌日の敗者復活戦に持ち込む事になりました。プログラムはどんなに入念にチェックを行っても、エラーが発生してしまう時は起こってしまうものだなと思いました。試合が終了した後は、会場で貰ったパンフレットを見ながら、他のチームの試合を見ていました。パンフレットには他のチームの解法アルゴリズムが掲載されており、中には自分の思いつかないような物もあり、とても参考になりました。17:00 になると、学生交流会が始まり、福井や高専のクイズが出題されました。そのクイズの中の一つで「ぶっちゃけ高専は？」で「絶対オススメ！」が正解だったのですが、「絶対やめとけ！」の票が僅差だったのが面白かったです。クイズが終わった後は、色々な高専の学生の方達と名刺交換をし、ホテルへ帰宅しました。顧問の北島先生は先生同士の情報交換会に行っていたため、夕食はホテル近くのスーパーとコンビニで買った惣菜で済ませました。先日の時点である程度調整を終えていたため、ホテルに備え付けてあつた漫画を読み、疲れもあったので早く就寝しました。

#### 4. 10/15 2 日目

##### 4.1. 敗者復活戦

昨日と同じ様に朝 6:00 に起床し、朝食はまたホテルのバイキングを頂きました。本来は 2 日目の夜に夜行バスで出発する為、アタッシュケース等の荷物をプロコン会場に持ち込む

予定でした。ですが、夕方に雨の予報が出ていたため、急遽到着時間を遅らせて、フロントに荷物を預けてから出発しました。会場についた後は、敗者復活戦が直ぐに行われる為、早速準備に取り掛かりました。前日のエラーは直り、万全の状態でした。敗者復活戦のルールは運営側が用意した AI と全チーム同時に対戦し、最もポイントの高かった 2 チームがファイナルステージに進めるのですが、試合が始まった直後に運営側からの通信が途絶えました。これは全チーム共通のようで、運営側が原因を調べて、復旧に急いでいました。数十分後、再戦の指示が出たので、再び一から試合を始めました。しかし今度は自分達のチームを含めた一部のチームの対戦相手 AI が動かないという、また新しいエラーが発生しました。他の多くのチームの対戦相手は動いているため、試合はそのまま進みました。相手が動いていないので、私達のポイントは上位に入っており、当然仕切り直しになりました。ただ、またこの様なエラーが発生する可能性があったのか今度は最初から相手が動かない状態でのポイント勝負という形に変更になりました。試合の結果は 8 位と、ファイナルステージにはいけませんでしたが、力を出せたので良かったと思います。余談ですが、私達のチーム名が「働け職人」というチーム名なのですが、1 日目と 2 日目どちらもどちらかの職人は動かなかったので、このチーム名には多分何かの呪いがあると思います。

敗者復活戦が終了した後は、見たい決勝戦まで時間があったので、競技ブースとは離れた自由部門・課題部門や企業ブースを各自見に行きました。企業ブースはリクルートや ABEMA、pixiv 等の有名企業も参加しており、改めてプロコンの規模の大きさを実感しました。

自由部門や課題部門の展示は各高専のアイデアを形にする技術力に驚嘆し、競技部門とはまた違った面白さがありました。お昼になつたら1日目と同じ様に、購入した弁当を食べました。具材が1日目と異なっていたため飽きがありませんでした。

午後になって、決勝戦を会場で見学しました。決勝戦は福井高専対熊本高専の試合で、2本先取です。結果は福井高専が2試合連続で勝利し、優勝しました。福井高専さんの解法アルゴリズムは自分達のチームと方向が類似していたため、方法を変えるだけで、これだけ強いAIを作成することができると勉強になりました。

決勝戦が終わった後は、閉会式で終了となります。閉会式では、競技部門と自由部門、課題部門の受賞式となるのですが、特に凄いと感じたのは、企業賞です。この企業賞は各協賛企業の方達が選んだチームにそれぞれ贈られるのですが、副賞が良く、30万円分のパソコンや、ゲーミングチェア、ペンタブレットなど魅力的な物が多かったです。企業賞の受賞のほとんどは課題部門や自由部門がほとんどを占めており、来年は企業賞を目指して、競技部門以外にも出てみるのも良いのではないかと不羨ながら思いました。

#### 4.2. 宮城へ帰宅

閉会式が終わった後は、ホテルの荷物を受け取りに武生駅へ行き、そこから金沢駅行きの特急に乗りました。金沢駅に着いた時刻で、時刻が20:00になっており、全員で金沢駅にある海鮮丼の店で夕食を取りました。



図5 海鮮丼

自分が頼んだのは漬け丼(図5)だったのですが、どの魚にも味が染みており、とても美味しかったです。夕食を食べた後は、夜行バス出発の22:30まで近くのマックでくつろぎました。金曜日の朝もマックで朝食を取ったため、マック始まり、マックで終わった旅でした。

時間になつたら夜行バスに乗車し、疲れもあつたのでよく寝られました。仙台駅についた後は、現地解散で私は真っ直ぐ家に帰りました。

#### 5. 終わりに

3年ぶりのプロコン旅行記如何でしたでしょうか。結果としてはファイナルステージにはいけませんでしたが、良い経験になったと思います。来年は1~2年生にも参加を進めて、自由部門や課題部門の大勢参加などもおこなってみたいです。ここまでお読み頂きありがとうございました。

## 編集後記

猛暑も終わり、随分と涼しくなってきました。さて、「Rational」vol.37 いかがでしたでしょうか。今年は新入部員も多く、「Rational」に寄稿して下さった部員の方たちも増え、編集長としては嬉しい限りです。高専祭では、実際にゲームや Web サイトを制作した人の話が聞けるので、時間があれば、ぜひ聞いてみてください。また、この「Rational」を読んで、プログラミングに少しでも興味を持っていただけたのなら幸いです。

最後にはなりますが、今回の「Rational」の制作にあたり寄稿して下さった部員の皆様、校正をしてくださった先輩方、顧問の北島先生と佐藤先生、そして読んで下さった皆様、全ての方に感謝を申し上げます。ありがとうございました。

編集長 鈴木 佑

### Rational Volume.37

発行日 10月28日

発行団体 仙台高専(名取) ソフトウェア研究部会

印刷所 仙台高専(名取) 事務棟複写室

#### STAFF

代表 鈴木 佑

編集 鈴木 佑

校正 鈴木 晴斗 富山 結都 鈴木 佑

表紙 大高 康介

執筆 ソフトウェア研究部員

製本 ソフトウェア研究部員

プロコン旅行記 鈴木 佑