KLab Tech Book 9

1. マイクラプログラミング(統合版)いろいろやってみた
2. 実行できる Python 標準ライブラリ
3. 進化計算コンペティションを支える技術
4. C# 向け JSON デシリアライザ Hurisake.JsonDeserializer を作った
5. オンライン対戦を支える独自シリアライズフォーマット
6. モデム通信しかサポートしていない昔のネット対戦ゲームを ブロードバンド・光回線で行うには

KLab Tech Book Vol. 9

2022-01-22 版　　**KLab** 技術書サークル 発行

はじめに

このたびは本書をお手に取っていただきありがとうございます。本書は KLab 株式会社の有志にて作成された KLab Tech Book の第 9 弾です。

KLab 株式会社では主にスマートフォン向けのゲームを開発していますが、本書ではこれまでどおり社内で有志を募り、エンジニアが業務との関連によらず好きな内容で記事を執筆し、デザイナーの方に表紙などを協力していただくことで、一冊の同人誌として仕上げました。

何年も活動を続けてメンバーが変わりつつも、有志による社外への情報発信の場として継続して発行できているのは、社内のエンジニアが変わらず技術に対する興味や探究心を持ち続けているからだと思います。

本書を通して、そんな KLab の雰囲気を読者のみなさまにも感じてもらえるとさいわいです。

梅澤 寿史

お問い合わせ先

本書に関するお問い合わせは tech-book@support.klab.com まで。

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。

目次

第 1 章

マイクラプログラミング（統合版）いろいろやってみた

Masato Yamada / @thunder_hey

1.1　はじめに

Minecraft（マイクラ）はプログラミングの学習題材としても注目されています。特に統合版にはさまざまな言語や方法でのプログラミング環境が用意されており、本章では同じゲームを複数のやり方で実装することにチャレンジした結果を書いてみました。

1.2　どんなゲームを作ったのか

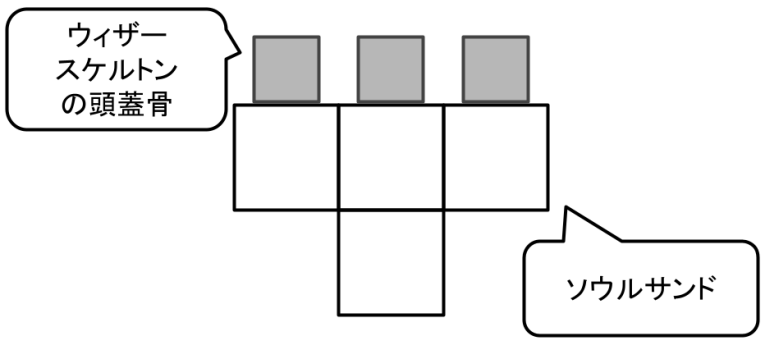
今回作ったのは、マイクラのウィザーという裏ボスを出現させたら負けというゲームです。複数プレイヤーで順番に「ウィザースケルトンの頭蓋骨」をひとつずつフィールドに置いていき、隠された条件を満たしてしまうとウィザーが出現して負けになります。また、出現させてしまった人は罰ゲームとしてウィザーを責任を持って倒さなくてはなりません。

尚、このゲームについてはドズル社の「最終決戦！ ウィザー黒ひげ危機一髪！【マイクラ】最終月例カズクラ 2020」[*1]という動画内のゲームを拝借しています。

[*1] https://www.youtube.com/watch?v=d0dzZ8F3DZ4

出現条件

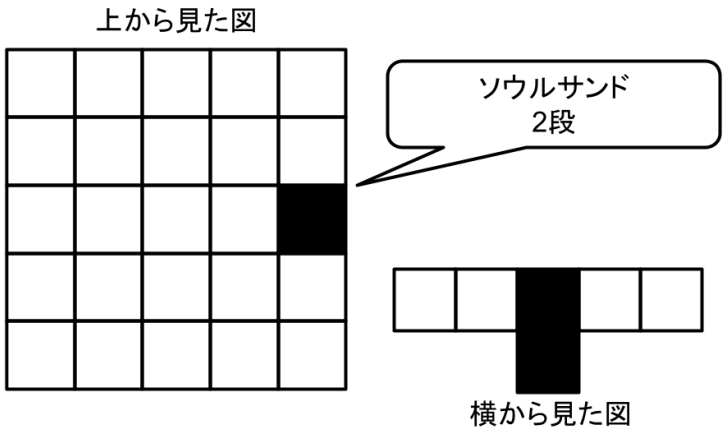
ウィザーの出現条件を図説します。図 1.1 のようにソウルサンドというブロックが T 時に配置された場所の上に、ウィザースケルトンの頭蓋骨というブロックを 3 つ配置するとウィザーが出現します。この形を作ってしまうことが、このゲームにおける負けの状態です。



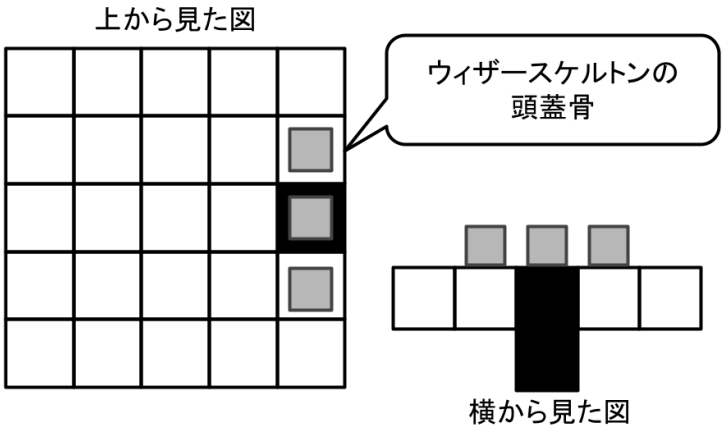
▲図 1.1　ウィザーの出現条件

ゲームのフィールドは図 1.2 のように縦と横に一定範囲ソウルサンドを敷き詰め、その下にさらにソウルサンドを隠して配置します。プレイヤーからはどこが二段になっているかわかりません。

複数のプレイヤーが順番にウィザースケルトンの頭蓋骨を配置し、図 1.3 のように出現条件が作られるまで繰り返します。



▲図 1.2　フィールド



▲図 1.3　出現条件が満たされた時

1.3　ゲームに必要な機能

このゲームに必要な機能は次のふたつとしました。

1. フィールドのリセット
2. 必要なアイテムの付与

フィールドのリセット

- 一定の範囲にソウルサンドを配置する
- 出現条件をランダムにするために、下段のソウルサンドの配置をランダムにする

必要なアイテムの付与

- 装備や消耗品の付与
- ゲームの進行に必要である配置可能なウィザースケルトンの頭蓋骨の付与

1.4　さまざまな方法での実装

1. Scripting API を使った実装

まず最初に挑戦したのが **Scripting API** と呼ばれているものでした。公式の開発情報を読んでも最初の一歩を踏み出すための情報が得られなかったのでネットを彷徨い、自分

に適切なものを探してたどり着いたのが Bedrock Wiki[*2]でした。Windows10 で動くスクリプトということでさっそく試してみました。

https://github.com/yamada-masa/PopupWither/blob/master/scripts/server/

▼リスト 1.1　server.js

```
const systemServer = server.registerSystem(0, 0)
systemServer.initialize = function () {

    // (中略)

    // ボタンを押したときのイベント登録
    this.listenForEvent(
        'minecraft:block_interacted_with',
        (e) => this.onInteracted(e));
};

// ボタンを押したときの挙動
systemServer.onInteracted = function (e) {
    x = e.data.block_position.x,
    y = e.data.block_position.y,
    z = e.data.block_position.z

    // アイテムを付与するボタン
    if (x === 0 && y === 5 && z === 0)
    {
        this.initUser();
    }

    // フィールドをリセットするボタン
    if (x === 0 && y === 5 && z === 1)
    {
        this.initGame();
    }
};

// 必要なアイテムの付与
const playerInitCommands = [
    // ゲームの進行に必要である配置可能なウィザースケルトンの頭蓋骨の付与
    '/give @p skull 128 1 {"minecraft:can_place_on":{"blocks":["soul_sand"]}}',
    // 装備や消耗品の付与
    "/give @p netherite_sword 1",
    "/give @p bow 1",
    "/give @p enchanted_golden_apple 64",
    "/give @p netherite_helmet 1",
    "/give @p netherite_chestplate 1",
    "/give @p netherite_leggings 1",
    "/give @p netherite_boots 1",
    "/give @p arrow 64",
]

systemServer.initUser = function () {
    playerInitCommands.forEach(function(x) {
        systemServer.executeCommand(x, (e) => {});
    });
};

// フィールドのリセット
const gameInitCommands = [
    // 一定の範囲にソウルサンドを配置する
    "/fill 20 6 0 30 6 10 soul_sand",
    // 前回配置したものを削除しておく
    "/fill 20 7 0 30 7 10 air",
    "/fill 20 5 0 30 5 10 air",
]
```

[*2] https://wiki.bedrock.dev/scripting/scripting-intro.html

```
systemServer.initGame = function () {
    // 出現条件をランダムにするために、下段のソウルサンドの配置をランダムにする
    var x = Math.floor(Math.random() * 11) + 20;
    var z = Math.floor(Math.random() * 11);
    command = `/fill ${x} 5 ${z} ${x} 5 ${z} soul_sand`;

    gameInitCommands.forEach(function(x) {
        systemServer.executeCommand(x, (e) => {});
    });
    systemServer.executeCommand(command, (e) => {});
};
```

普段 JavaScript を書かない私ですが、これならいろいろ実現できそうだという感触が得られました。デメリットとしては Widnows10 やサーバーでしか動かないというところでしょうか。

2. Functions で実装

Functions はゲーム内で使えるコマンドをひとまとめにして実行する機能です。これを使っても実装が可能ということが分かったので試してみました。

フィールドのリセット

https://github.com/yamada-masa/PopupWither2/blob/master/functions/

▼リスト 1.2　stage_reset.mcfunction

```
fill 4 5 13 -5 7 22 air
fill 4 6 13 -5 6 22 soul_sand
scoreboard objectives add x dummy
scoreboard players random @e[name=random] x 1 100

execute @e[name=random,scores={x=1}]  ~ ~ ~ fill -5 5 13 -5 5 13 soul_sand
execute @e[name=random,scores={x=2}]  ~ ~ ~ fill -5 5 14 -5 5 14 soul_sand
execute @e[name=random,scores={x=3}]  ~ ~ ~ fill -5 5 15 -5 5 15 soul_sand
execute @e[name=random,scores={x=4}]  ~ ~ ~ fill -5 5 16 -5 5 16 soul_sand
execute @e[name=random,scores={x=5}]  ~ ~ ~ fill -5 5 17 -5 5 17 soul_sand
execute @e[name=random,scores={x=6}]  ~ ~ ~ fill -5 5 18 -5 5 18 soul_sand
execute @e[name=random,scores={x=7}]  ~ ~ ~ fill -5 5 19 -5 5 19 soul_sand
execute @e[name=random,scores={x=8}]  ~ ~ ~ fill -5 5 20 -5 5 20 soul_sand
execute @e[name=random,scores={x=9}]  ~ ~ ~ fill -5 5 21 -5 5 21 soul_sand
execute @e[name=random,scores={x=10}] ~ ~ ~ fill -5 5 22 -5 5 22 soul_sand
execute @e[name=random,scores={x=11}] ~ ~ ~ fill -4 5 13 -4 5 13 soul_sand
execute @e[name=random,scores={x=12}] ~ ~ ~ fill -4 5 14 -4 5 14 soul_sand

(中略)

execute @e[name=random,scores={x=99}]  ~ ~ ~ fill -2 5 21 -2 5 21 soul_sand
execute @e[name=random,scores={x=100}] ~ ~ ~ fill -2 5 22 -2 5 22 soul_sand
```

「出現条件をランダムにするために、下段のソウルサンドの配置をランダムにする」を実現するために記述がかなり増えてしまいました。100 通りあるすべてのパターンについて 1 行ずつ記述する必要がありました。条件分岐や変数の扱いにクセがあるため、気軽にプログラミングするという感じではありませんでした。

必要なアイテムの付与

https://github.com/yamada-masa/PopupWither2/blob/master/functions/

▼リスト 1.3　user_init.mcfunction

```
give @a[c=1] skull 128 1 {"minecraft:can_place_on":{"blocks":["soul_sand"]}}
give @a[c=1] netherite_sword 1
give @a[c=1] bow 1
give @a[c=1] enchanted_golden_apple 64
give @a[c=1] netherite_helmet 1
give @a[c=1] netherite_chestplate 1
give @a[c=1] netherite_leggings 1
give @a[c=1] netherite_boots 1
give @a[c=1] arrow 64
```

こちらはほぼほぼ JavaScript と変わらない記述量でした。ターゲットをどうするかというところが少し他の方法で実装したときと異なる感じとなりました。ちょっと癖があります。

3. WebSocket Server で実装（言語は Python）

次に、私が普段使っている Python を使って何とか自由に書けないかといろいろ調べてみました。WebSocket サーバーを作ってクライアントからつなぐことによってプログラミングできることが分かったので WebSocket サーバーを Python で書いてみることにしました。この方法は先人のブログ記事[*3]を参考にしてみました。

https://github.com/yamada-masa/PopupWither3/blob/master/

▼リスト 1.4　main.py

```
import asyncio
import json
import random
from uuid import uuid4

import websockets

EVENTS = [
    "PlayerMessage",
]

CMDS = [
    # ゲームの進行に必要である配置可能なウィザースケルトンの頭蓋骨の付与
    '/give {} skull 128 1 {"minecraft:can_place_on":{"blocks":["soul_sand"]}}',
    # 装備や消耗品の付与
    "/give {} netherite_sword 1",
    "/give {} bow 1",
    "/give {} enchanted_golden_apple 64",
    "/give {} netherite_helmet 1",
    "/give {} netherite_chestplate 1",
    "/give {} netherite_leggings 1",
    "/give {} netherite_boots 1",
    "/give {} arrow 64",
```

[*3] http://www.s-anand.net/blog/programming-minecraft-with-websockets/

```
]

async def mineproxy(websocket, path):
    print("Connected")

    msg = {
        "header": {
            "version": 1,
            "requestId": "",
            "messageType": "commandRequest",
            "messagePurpose": "subscribe",
        },
        "body": {"eventName": "PlayerMessage"},
    }
    await websocket.send(json.dumps(msg))

    async def send(cmd):
        await websocket.send(
            json.dumps(
                {
                    "header": {
                        "version": 1,
                        "requestId": str(uuid4()),
                        "messagePurpose": "commandRequest",
                        "messageType": "commandRequest",
                    },
                    "body": {
                        "version": 1,
                        "commandLine": cmd,
                        "origin": {"type": "player"},
                    },
                }
            )
        )

    async def stage_reset():

        # 一定の範囲にソウルサンドを配置する
        await send("/fill -5 5 13 4 7 22 air")
        await send("/fill -5 6 13 4 6 22 soul_sand")

        # 出現条件をランダムにするために、下段のソウルサンドの配置をランダムにする
        x = random.randint(1, 10)
        z = random.randint(1, 10)
        await send(f"/setblock {-5 + x} 5 {13 + z} soul_sand")

    async def user_init(name: str):
        for cmd in CMDS:
            await send(cmd.format(name))

    try:
        async for msg in websocket:
            print(msg)
            msg = json.loads(msg)
            if msg["body"].get("eventName", "") == "PlayerMessage":
                text = msg["body"]["properties"]["Message"]
                if text == "stage-reset":
                    await stage_reset()
                elif text == "user-init":
                    player_name = msg["body"]["properties"].get("Sender")
                    await user_init(player_name)

    except websockets.ConnectionClosedError:
        print("Disconnected")


start_server = websockets.serve(mineproxy, port=19131)
print("/connect localhost:19131")
asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

直接的にゲーム内のボタンに機能を割り当てることが手間だったので、コマンドとして実装する形に落ち着きました。WebSocket を制御する部分を作り込む必要があるのでコード量はその分増えましたが、かなり自由度が高いと感じました。

1.5　4. MakeCode で実装 (Python)

最後に試したのが MakeCode[*4]でした。こちらは教育目的と思って後回しにしていたのですが、教育目的だけあってハードルの低さが際立ちました。バックエンドの仕組みとしては先ほどとおなじく WebSocket を使っています（むしろこのためにある機能であったとあとから気づきました)。プログラミングの方法としてブロックプログラミング、JavaScript、Python を選択可能です。私はもちろん Python を選択しました。

https://github.com/yamada-masa/PopupWither4/blob/master/

▼リスト 1.5　main.py

```
def stage_reset():
    # 一定の範囲にソウルサンドを配置する
    blocks.fill(AIR, world(-5, 5, 13), world(4, 7, 22))
    blocks.fill(SOUL_SAND, world(-5, 6, 13), world(4, 6, 22))

    # 出現条件をランダムにするために、下段のソウルサンドの配置をランダムにする
    x = randint(1, 10) -5
    z = randint(1, 10) + 13
    blocks.place(SOUL_SAND, world(x, 5, z))

def user_init():
    # ゲームの進行に必要である配置可能なウィザースケルトンの頭蓋骨の付与
    param = '{"minecraft:can_place_on":{"blocks":["soul_sand"]}}'
    player.execute(f"give @s skull 128 1 {param}")
    # 装備や消耗品の付与
    player.execute("give @s netherite_sword 1")
    player.execute("give @s bow 1")
    player.execute("give @s netherite_helmet 1")
    player.execute("give @s netherite_chestplate 1")
    player.execute("give @s netherite_leggings 1")
    player.execute("give @s netherite_boots 1")
    player.execute("give @s arrow 64")

player.on_chat("stage-reset", stage_reset)
player.on_chat("user-init", user_init)
```

実装してみて感じたのは、WebSocket の制御部分がすべて隠蔽されていて、マイクラを操作するための関数＋標準的な Python の関数が準備されているため、やりたいことに対して直感的に実装できるということでした。

注意点として MakeCode を使ったプログラムを動かすためには Minecraft Windows10 版または Minecraft Education Edition(iPadOS/macOS/ChromeOS/Windows10) が必要ということです。Minecraft Windows10 版では Code Connection for Minecraft が必要になります。これが WebSocket サーバーとして動作するため他の OS では動かないと

[*4] https://minecraft.makecode.com/

いうことになります。あくまで「プログラミングを学ぶ」用途なので「ゲーム作りをする」のには向かないといえます。

1.6 おわりに

全部で 4 種類の方法でゲームを実装してみることを試してみましたが、今回は簡単にできる範囲にとどまってしまいました。もう少し凝ったことを行うことでどのプログラミング手段がより適しているのかなどがはっきりするのではないかと思いました。マイクラでのゲームづくりは必ずしもアセットを用意する必要がないので、プログラミングは得意だけど見た目を作るのが苦手という方にもゲームづくりを楽しめる手段のひとつと感じました。

第 2 章

実行できる Python 標準ライブラリ

Shunsuke Ito / @fgshun

2.1　はじめに

`python -m` 。モジュールを Python スクリプトとみなして実行するオプションです。これによって標準ライブラリを実行したときの動作を紹介します。

2.2　開発で役に立つライブラリ

Python で開発するにあたって手助けしてくれるライブラリたち。それを直接実行した時の挙動です。

venv - 仮想環境の作成

Python 仮想環境を構築します。これで作った仮想環境は activate や Activate.ps1 というスクリプトで有効化できます。複数の Python が入っている環境での各バージョンの使い分けをすることができ、有効化した状態での pip install は元の環境に影響を及ぼしません。不要になったらディレクトリごと削除するだけで掃除することができます。お勧めの使い方として、 venv 環境を常用して大元への pip install を行わないようにしておく、というものがあります。こうすることで、まっさらな環境の準備をこのコマンドひとつで用意できます。

▼リスト 2.1　venv

```
$ python -m venv my_venv
$ . my_venv/bin/activate
(my_venv) $ deactivate
$ rm -r my_venv
```

asyncio - 非同期 I/O

await 文が使用できるインタラクティブシェルを立ち上げます。通常のインタラクティブシェルだとシンタックスエラー扱いになってしまう await 文ですが asyncio 版のシェルであれば await 文の挙動を試すことができます。

▼リスト 2.2　asyncio

```
$ python -m asyncio
>>> import asyncio
>>> await asyncio.sleep(7)
```

timeit - 小さなコード断片の実行時間計測

Python コード片の速度を計測します。たとえば多数の文字列を連結する時、 + 演算の繰り返しと `str.join` の速度差を調べるためには次のようにします。

▼リスト 2.3　timeit

```
$ python -m timeit --setup='s = ""' 'for _ in range(10000): s += "spam"'
200 loops, best of 5: 1.32 msec per loop
$ python -m timeit '"".join("spam" for _ in range(10000))'
500 loops, best of 5: 597 usec per loop
```

pickle - Python オブジェクトの直列化

pickle は Python オブジェクトをシリアライズします。その出力結果は黎明期に使われたバージョン 0 をのぞいてバイナリ形式であり、その内容を人が読むのは困難です。pickle を直接起動するとシリアライズ済みの内容を人が読める形で出力してくれます。入れ子になった巨大リスト・辞書の場合でもインデントして読みやすい形で出力されます。

▼リスト 2.4　pickle

```
$ python -c "import pickle; pickle.dump('spam', open('spam.pickle', 'wb'))"
$ python -m pickle spam.pickle
'spam'
$ python -c "import pickle;pickle.dump(['spam'*16]*2,open('a.pickle','wb'))"
$ python -m pickle a.pickle
['spamspamspamspamspamspamspamspamspamspamspamspamspamspamspam',
 'spamspamspamspamspamspamspamspamspamspamspamspamspamspamspam']
```

pdb - Python デバッガ

Python デバッガである pdb を直接立ち上げます。なお、 pdb 側のオプションにも -m が存在しモジュールを直接デバッグすることが可能です。

▼リスト 2.5　pdb

```
$ python -m pdb -g
usage: pdb.py [-c command] ... [-m module | pyfile] [arg] ...
```

trace - Python 文実行のトレースと追跡

Python コードを実行し、各関数の関係、各文の呼び出し回数、実行順などを得ます。たとえば -T によって各関数の呼び出し、呼び出されの関係を得ることができます。

▼リスト 2.6　trace

```
$ cat a.py
def add(a, b): return a + b
def main(): print(add(1, 2))
main()
$ python -m trace -T a.py
(中略)
*** a.py ***
    a.<module> -> a.main
    a.main -> a.add
```

dis - Python バイトコードの逆アセンブラ

dis.dis による逆アセンブルの結果を出力します。

▼リスト 2.7　dis

```
$ cat hello.py
print('Hello, World!')
$ python -m dis hello.py
  1           0 LOAD_NAME                0 (print)
              2 LOAD_CONST               0 ('Hello, World!')
              4 CALL_FUNCTION            1
              6 POP_TOP
              8 LOAD_CONST               1 (None)
             10 RETURN_VALUE
```

doctest

インタプリタの実行結果を docstring へ貼り付けておくと、例示にもテストにもなる、それが doctest です。そのテストを起動します。成功時には何も表示されず、失敗時にはドキュメントと実際に得られた値を出力します。次の例は加算する関数を誤って乗算を行うよう変更してしまったものに対し、 doctest を実行しています。

▼リスト 2.8　doctest

```
$ cat b.py
def add(a, b):
    """add a to b

    >>> add(1, 2)
    3
    """
    return a * b
$ python -m doctest b.py
File "b.py", line 4, in b.add
Failed example:
    add(1, 2)
Expected:
    3
Got:
    2
（後略、テスト数、テスト結果が続く）
```

2.3　ライブラリのデモンストレーションを行うライブラリ

直接実行すると、自身のデモンストレーションを始めるライブラリを紹介します。

アーカイブファイルを操作するライブラリ

Python にはアーカイブファイルを操作するライブラリがいくつか含まれています。たとえば gzip, zipfile, tarfile 。これらは直接実行すると対応する Unix コマンドの機能の一部を模倣した Python スクリプトとして機能します。

▼リスト 2.9　gzip

```
$ python -m gzip -h
usage: gzip.py [-h] [--fast | --best | -d] [file ...]
（省略、オプションの説明が続く）
```

使ってみて、そしてどのように実装されているのかを読んでみてください。意外に少ない実装であることがわかります。

バイナリをテキスト形式で符号化するライブラリ

Python にはバイナリ形式のデータをテキスト形式とするエンコード方式各種に対応するライブラリが含まれています。たとえば base64, uuencode, quopri 。これらは直接実行して標準入出力やファイルの内容をエンコード、デコードすることが可能です。

▼リスト 2.10　base64

```
$ python -m base64 -h
option -h not recognized
usage: base64.py [-d|-e|-u|-t] [file|-]
        -d, -u: decode
        -e: encode (default)
        -t: encode and decode string 'Aladdin:open sesame'

$ echo spam | python -m base64
C38hbQo=
```

これらは直接実行できることがドキュメントには記載されておらず、 `-h` オプションが欠けていることから察するに、使ってもらうためというよりはライブラリ開発者自身が動作を試すために用意したものなのかもしれません。

calendar

カレンダー出力を行うライブラリです。これを実行すると cal コマンドが生成するような text 形式でカレンダーを作ることができます。このライブラリを用いて cal コマンドを模倣するにはどのようにすればよいのかを紹介するような内容となっています、ぜひ `calendar.py` を読んでみてください。ただし、最初の曜日を月曜から日曜に、日本でよく使われる形へ変更することはできませんでした。ライブラリ自身にはこのための設定項目が存在するのですが、残念ながら直接実行でこの設定に触れる方法はありませんでした。

▼リスト 2.11　calendar

```
$ python -m calendar 2021 12
   December 2021
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

webbrowser

ブラウザを立ち上げて渡された URL を開きます。ブラウザを立ち上げる webbrowser.open 関数の使い方を紹介するかのような内容となっています。

▼リスト 2.12　webbrowser

```
$ python -m webbrowser
Usage: webbrowser.py [-n | -t] url
    -n: open new window
    -t: open new tab

$ python -m webbrowser -t "https://www.klab.com/jp/blog/tech/2021/tbf11.html"
```

http.server

簡易 HTTP サーバーが立ち上がります。これは別の標準ライブラリ socketserver.TCPServer の使い方を示したデモとなっています。2021 年現在、 Python 製 Web フレームワーク各種、サーバー各種があるなか、これを直接使う機会はないでしょう。それでも、Python があれば、それ以上の準備の必要もなくコマンドひとつで HTTP サーバーが立つ、ということを頭の片隅に置いておくと役に立つことがあるかもしれません。

▼リスト 2.13　http.server

```
$ python -m http.server --help
usage: server.py [-h] [--cgi] [--bind ADDRESS] [--directory DIRECTORY] [port]
（省略、オプションの説明が続く）
```

turtle - タートルグラフィックス

教育向けのグラフィックプログラミング、タートルグラフィックスのデモが実行されます。見ていて楽しいです。

2.4　終わりに

-m オプションで標準ライブラリを実行した時の挙動を調べてみました。このほかに、公式ドキュメントには記載がないものの自身の簡易テストを始めるライブラリなどもあり、探していて面白かったです。

第 3 章

進化計算コンペティションを支える技術

Naoki Hamada / @hmkz_

筆者は**進化計算コンペティション**というプログラミングコンペティションの代表幹事を務めています。進化計算コンペティションの特徴は、科学や産業の現場で扱われる本格的な実問題を出題することです。実問題コンペティションは挑戦しがいがありますが、参加者にとっても運営にとっても負担が大きくなりがちです。無理なく毎年開催するためには、システムによる支援が欠かせません。本章では、コンペティションを運営するために開発したシステム **OptHub** について紹介します。実問題コンペティションを継続的に開催するうえで何が課題となっていて、それを OptHub がどう解決したのかを説明します。また、OptHub を用いて進化計算コンペティションを開催するための組織運営についても紹介します。OptHub の使い方については `https://ec-comp.jpnsec.org/ja/tutorial` をご覧ください。

3.1 はじめに

進化計算コンペティションは、進化計算学会[*1]の主催する最適化コンペティションです。毎年 10 月から 12 月にかけて最適化問題が公開され、参加者は規定の評価回数以内にできるだけ優れた近似最適解を発見することを競います。その結果は 12 月下旬の進化計算シンポジウムにおいて発表されます。

コンペティションの開催に向けて、運営委員はさまざまな準備を行います。本コンペティションは科学や産業の現場で扱われるリアルな実問題を出題するため、企業や研究機関と協力して候補となる実問題を収集します。実問題は機密情報や動作環境等の都合によりそのままでは出題困難なこともあるため、必要に応じて定式化しなおし、問題プログラムを改修もしくは再開発します。問題の説明文を執筆し Web で公開して参加者を集め、

[*1] `http://jpnsec.org`

競技期間中にはトラブル対応や参加者のサポートを行いながら、提出された解データを集計し、表彰や講評を行います。コンペティション開催後は、将来の実問題研究を促進するために問題プログラムと参加者の解データを一般公開し、保守します。

これらの仕事は、2017 年に大山聖氏（JAXA）が本コンペティションを発足して以来、2019 年まで彼をはじめとする有志によって行われてきました。2020 年 1 月に進化計算学会に実世界ベンチマーク問題分科会が設立され、今後の運営は分科会が引き継ぐことになりました。それに伴い、分科会ではコンペティションの運営を円滑化するために OptHub を開発しました。OptHub は、単にコンペティションの採点システムであるだけでなく、運営にまつわる先述のすべての工程を総合的にサポートするシステムになっています。さまざまな機能をもつ一方で、OptHub のシステム運用自体が負担にならないように、機能の大部分は既存の Web サービスに移譲する形で構築されています。運営委員は普段から使い慣れているサービスを利用する感覚で OptHub を運用することができます。

OptHub を用いて開催された進化計算コンペティション 2020[4] では、実問題の収集から問題公開までの一連の作業は、期間にして 2 か月以内[*2]、稼働時間にしてのべ 50 時間以内で完了しました。また、最大で 27,720 変数 7 目的という大規模問題を出題し、参加者の提出した解 60,000 件以上を OptHub 側で評価しました。これは、既存のオンラインシステムでホストされる最適化コンペティションを 1〜2 桁上回るスケールです。問題プログラムのソースコードとコンテナイメージはリポジトリで公開されており、解データはクエリ言語で取得することができます。

3.2 進化計算コンペティションへの期待と課題

分科会は、システム開発に先立って、進化計算コンペティションへの期待と現状の課題を把握するためにアンケートを行いました。アンケート結果をもとに、コンペティションの運営に必要な機能を整理し、既存のコンペティションシステムも参考にして、OptHub の開発要件をまとめました。以下、「アンケート」ではアンケートを紹介して要件をまとめ、「既存のコンペティションシステム」では既存のコンペティションシステムと要件を照らして OptHub 開発の動機を説明します。

アンケート

出題者の立場と競技者の立場のそれぞれからアンケートを実施しました。アンケートは 2020 年 4 月 30 日に進化計算学会のメーリングリストで配布しました。回答期間は 4 月 30 日から 5 月 15 日までとしました。

[*2] 進化計算コンペティション 2020 で出題した問題が KLab より提案されてから OptHub で公開されるまでに要した期間です。実問題は通年収集しており、2020 年はこの他にも 5 つの実問題を検討しました。

アンケート結果概要

本節では、アンケート結果を要約して、得られた知見をまとめます。質問文と回答の詳細は「アンケート結果詳細」を参照してください。

■コンペティションへの関心は高いが、敷居も高い 出題者アンケートは企業勤務者を中心に 7 件の回答があり（出題者 Q1）、競技者アンケートは大学教員を中心に 10 件の回答がありました（競技者 Q1）。多くの組織に実問題はありますが、出題できていません（出題者 Q7）。いずれの年のコンペティションも、10 人中 4〜6 人は問題文を読んでいますが、解を提出した人は 1〜2 人でした（競技者 Q2）。出題者・競技者ともに興味はあるものの敷居の高い様子がうかがえます。

■進化計算を使いこなすためのベストプラクティスの共有が不足しており、解データの共有が望まれている 進化計算への期待としては、すでに幅広いニーズをカバーする汎用的な手法の実装があり、解の精度や信頼性には満足な一方で、問題をどう定式化してどの手法で解くかの判断に専門知識が必要なことに不満が集まりました（出題者 Q3）。進化計算を使いこなすためのベストプラクティスの集積が求められています（出題者 Q4）。出題者と競技者ともに、サーバーで解を評価することやコンペティション後に解を公開することには多くの賛成がありました（出題者 Q11、競技者 Q7）。一方で、悪い成績を公開されたくないという意見もあったため（競技者 Q7）、オプトアウトを認める必要もあります。

■問題提供には、承諾取得・プログラム改修・計算資源の支援が必要 出題の承諾を得ることやプログラム改修が共通のネックとなっています（出題者 Q8）。進化計算の応用先は製品設計が主であり（出題者 Q2）、実問題は Windows 用に C/C++ で実装されていることが多いようです（出題者 Q13）。一方で、参加者の使用する OS やプログラミング言語は多岐に渡るため（競技者 Q10）、すべての OS と言語をサポートするコストが大きいと感じています（出題者 Q6）。大規模な計算資源や特殊なソフトウェアが要求されることもあるため（出題者 Q8）、スーパーコンピュータで評価したり、オンプレミスで評価するといった選択肢にも対応できるようにしておく必要がありそうです。

■参加には、インストールや計算機の支援が必要 本格的な実問題であるだけに、問題プログラムのインストール作業などの準備が大変であることや、計算機が不足するといった点がネックとなっています（競技者 Q4）。解をサーバーで評価する方式に変えることで、インストール作業は不要になり、参加者が高性能な計算機を用意する必要もなくなります。なお、参加する時間がないという意見も多かったのですが（競技者 Q4）、開催時期・期間は例年どおり（9〜11 月）で適切との声が多数でした（出題者 Q10、競技者 Q6）。

■代表的なソルバーでの最適化結果が望まれている 従来のコンペティションでも公開してきた情報に加えて、代表的なソルバーでの最適化結果も公開可能との回答が多くありました（出題者 Q9）。これは競技者側からも希望する声がありました（競技者 Q8）。

■評価回数制限と事前検討が望まれている　サーバーで解を評価するうえでどの程度試行錯誤を認めるかについては、評価回数に上限を設けつつも事前検討の機会を設ける方式に賛成が集まり、完全な 1 試行勝負を希望する人はいませんでした（出題者 Q11、競技者 Q7）。1 試行の評価回数を多めにするか少なめにするかについては意見が分かれました（出題者 Q11、競技者 Q7）。

■連続変数以外の問題も望まれている　産業現場には、連続最適化問題だけでなく、整数変数や混合変数の問題、不確実問題、実行時間制限もあります（出題者 Q10）。また、競技者もそのような出題を望んでいます（競技者 Q8）。

既存のコンペティションシステム

「アンケート結果概要」で述べたニーズをどのように満たすかを検討しました。もし進化計算コンペティションを既存のオンラインコンペティションシステムでホストできれば、運営委員の限られたリソースを実問題収集と作問支援に集中できます。また、そのシステムの既存ユーザーが流入することも期待できます。そこで、国内外に知名度が高く、最適化コンペティションの運用実績もある BBComp、AtCoder、Kaggle が進化計算コンペティションをホストできるかを検討しました。

BBComp

BBComp[6] は進化計算分野で最も実績のあるブラックボックス最適化コンペティションのオンラインシステムです。コンペティションは 2015 年から 2019 年まで毎年、国際会議 GECCO[*3]に合わせて開催されており、一部の年は CEC[*4]や EMO[*5]でも開催されました。

プレイヤーは専用の通信モジュールを使ってサーバーに解を送信し、サーバー側で目的関数値とスコアの計算が行われ、結果を受信します。これまでに送信したすべての解をもとにスコアが決まり、規定回数の送信で終了となります。送信する解は進化計算などのソルバーで作成してもいいですし、人手で作成することもできます。解を 1 つ送信するたびに結果が得られるため、それを見て途中で探索戦略を変えるなどといった Human-in-the-Loop な最適化ができる点が大きな特徴です。Human-in-the-Loop 方式は実務での最適化問題への取り組み方に近く、リアリティがあります。

しかし、この方式をオンラインで実現するために、BBComp の問題規模はひかえめになっています。過去のコンペティションと実問題の規模の比較を表 3.1 に示します。BBComp は、ほとんどの年で計算時間の短い人工問題を扱い、変数・目的・制約のいず

[*3] Genetic and Evolutionary Computation Conference (GECCO), https://gecco-2021.sigevo.org/.
[*4] IEEE Congress on Evolutionary Computation (CEC), https://cec2021.mini.pw.edu.pl/.
[*5] Evolutionary Multi-Criterion Optimization (EMO), https://emo2021.org/.

れもが少ない傾向があります。BBComp2017 EMO のみ実問題を扱っていますが[*6]、例年に輪をかけて小規模です。一方、進化計算コンペティション（ECComp）はより大規模な実問題を扱っています。統計的には、実問題の規模は進化計算コンペティションに近いようです。実問題として、最近 5 年の GECCO（[38] の表 4）とアンケート調査（[5] の Fig. 3）に掲載された合計 82 件を集計しました。表の値は、集計値の 95 パーセンタイル（82 件の集計結果から各列の最大・最小 2 件ずつを除いた 78 件の値がとる範囲）を示しています。78 件の実問題は 1 ～ 100,000 変数、1 ～ 8 目的、0 ～ 100,000 制約、10 ～ 100,000 評価[*7]の範囲に収まっています。実問題のリアリティを担保するためには、コンペティションシステムはこれらの上限にあたる規模の問題をホストできることが望ましいと考えます。

BBComp のソースコードの一部は Web サイトで公開されています。この内容を確認すると、扱える問題にさまざまな制限があることがわかります。解のデータ型の仕様上、変数は実数値ベクトルに限られており、制約を扱うことはできません。Hypervolume の実装は 2 目的と 3 目的しかサポートしていません。問題は、数式を独自のドメイン固有言語で記述するか、さもなければ C++ で実装する必要があります。後者はサーバープログラムと静的リンクする必要があるため、サーバーを止めなければ問題を追加することができませんし、第三者が提供する問題をホストする際にソフトウェアライセンスが競合する恐れがあります。問題プログラムのみを切り出して公開するためにも再実装が必要になります。コンペティションで提出された解を公開するための仕組みもありません。

AtCoder

AtCoder[7] は国内最大の競技プログラミングコンテストサイトです。8 万人以上のユーザーを抱え、毎週開催されるコンテストには国内だけでなく世界各国から 1 万人を超えるユーザーが参加しています。[*8]出題される問題の大半は最適化とは無関係ですが、近年、最適化コンテストも増加傾向にあります。これまでに、日立北大ラボ [11], [12], [13], [14], [15], [16]、Asprova[17], [18], [19], [20]、DISCO[21]、ヤマト運輸 [10] が実問題最適化コンテストを開催しています。

AtCoder のシステムは、オープンソースソフトウェアの DOMjudge[*9]に似ています。BBComp とは対照的に、AtCoder の最適化コンテストは基本的に Human-out-of-the-Loop 方式です。プレイヤーは解データではなく、ソルバーのソースコードを提出し、サーバー側でソルバーをバッチ実行した結果がスコアとして返却されます。実問題では目的関数値や制約関数値の計算に数時間から数日を要することもあるため（[5] の Fig. 6）、最適

[*6] 実問題の内容は `https://www.ini.rub.de/PEOPLE/glasmtbl/projects/bbcomp/#realworld` を参照。

[*7] 実問題アンケート調査では評価回数は無制限との回答が 20/45 件を占めましたが（[5] の Fig. 7）、ここでは制限のあるケースのみを集計しました。

[*8] AtCoder Beginner Contest 190 の順位表 [8] に掲載されたプレイヤー数です。

[*9] DOMjudge（`https://www.domjudge.org/`）は国際的なプログラミングコンテスト ICPC などに使われているオンラインジャッジシステムです。

▼表 3.1　実問題とコンペティションの問題規模

	問題数	変数	目的	制約	評価回数	参加者数
BBComp2015 CEC	1,000 問 × 1 部門	2 ∼ 64	1	0	400 ∼ 409,600	25
BBComp2015 GECCO	1,000 問 × 1 部門	2 ∼ 64	1	0	20 ∼ 6,400	28
BBComp2016 GECCO	1,000 問 × 5 部門	2 ∼ 64	1 ∼ 3	0	20 ∼ 409,600	49
BBComp2017 EMO	10 問 × 1 部門	4 ∼ 24	2	0	100 ∼ 2,000	10
BBComp2017 GECCO	1,000 問 × 5 部門	2 ∼ 64	1 ∼ 3	0	20 ∼ 409,600	50
BBComp2018 GECCO	1,000 問 × 5 部門	2 ∼ 64	1 ∼ 3	0	20 ∼ 409,600	45
BBComp2019 GECCO	1,000 問 × 5 部門	2 ∼ 64	1 ∼ 3	0	20 ∼ 409,600	28
ECComp2017 [1]	1 問 × 2 部門	222	1 ∼ 2	54	30,000	20
ECComp2018 [2]	1 問 × 2 部門	2	1 ∼ 3	2	30,000	15
ECComp2019 [3]	1 問 × 2 部門	32	1 ∼ 5	22	10,000	15
ECComp2020 [4]	16 問 × 2 部門	60 ∼ 27,720	1 ∼ 7	12	100 ∼ 800	30
実問題サーベイ [38], [5] の 95 パーセンタイル	78 問	1 ∼ 100,000	1 ∼ 8	0 ∼ 100,000	10 ∼ 100,000	--

化の途中で人間が介入する技術も重要となりますが、AtCoder で Human-in-the-Loop 方式の最適化コンテストが開催された実績はありません。

AtCoder では、提出したソースコードは Web サイト上で閲覧できるほか、有志による Web API[22] によって一覧を取得できます。しかし、あくまでも競技プログラミングのサイトであるため、最適化に特化した情報公開の仕組みはありません。特に、最適化問題のプログラムを他の出題者が再利用したり、ソルバーが探索過程で生成した解データを公開する仕組みがありません。また、さまざまなタイプのコンテストが開催されるため、最適化コンテストがそれ以外の大量のコンテストの中に埋もれてしまうという問題もあります。

Kaggle

Kaggle[23] は企業・研究機関・政府とデータサイエンティスト・機械学習エンジニアを結びつける国際的なプラットフォームです。世界各国の 100 万人を超えるユーザーを抱え、コンペティションには数百〜数千チームが挑戦し、出題者は実問題や賞金を提供する見返りとして入賞者の作成した機械学習コードやモデルを譲り受けることができます。日本企業からも Mercari[24] や Recruit[25] が出題しています。問題は機械学習のタスクが大半ですが、Kaggle 自身が毎年クリスマスに開催する通称サンタコンペ [26], [27], [28], [29], [29], [30], [31], [32], [33], [34], [35] や Google の Hash Code[36], [37] のような最適化コンペティションも開催されています。AtCoder と同様、最適化コンペティションはそれ以外の大量のコンペティションの中に埋もれてしまいがちですが、Kaggle ではタグ機能を使った検索で見つけ出すことができます。

Kaggle のコンペティションは、基本的には、プレイヤーが Kaggle 上のノートブックまたは自身の環境で求めた解を 1 つだけシステムに提出し、そのスコアが返却される方式です。AtCoder と違って解の作成には人手で介入する余地はありますが、BBComp ほど頻繁には解を提出できないルール（1 日に 5 回まで）になっています。出題された問題への過適合を防ぐため、提出締切後に異なる問題を使って最終的なスコアを評価する Private Leaderboard 方式がとられることも多いです。

出題者の立場からすると、コンペティションを Kaggle でホストすれば世界最大規模のユーザーにリーチできる反面、規模に見合った入念な準備が必要になります。参加チームを増やすためには、他のコンペティションに見劣りしない高額賞金も求められます。一方、競技者の立場では、コンペティション終了後にデータとコードを公開し、解法をノートブックとして共有する機能があります。データやコード、ノートブックは Web API でダウンロードすることもできます。しかし、これらの情報はユーザーが自発的にアップロードする必要があるうえに、英語で分かりやすいノートブックを用意するにはかなりの手間がかかります。総じて情報共有に必要な機能は揃っているものの、出題者にも競技者にも負担が大きく、進化計算コンペティションの敷居をさらに高めてしまう恐れがあります。

3.3 OptHub

「既存のコンペティションシステム」での検討から、大規模実問題を Human-in-the-Loop 方式でホストし、コンペティションを通して収集したプログラムやデータを再利用しやすい形で公開するためには新たなシステムが必要と判断しました。以下では、進化計算コンペティションをオンライン開催するために開発した新システム OptHub について説明します。

要件

本システムに求められる要件を、対象ユーザー、コンペティション方式、可用性、セキュリティ、予算、運用性といった観点から整理します。

対象ユーザー

当面は進化計算学会員を対象としますが、運用体制が安定した後は国内全体、将来的には国際展開も目指します。したがって、インターネット環境の整備されたすべての国から日本国内と同じ条件でコンペティションをプレイできるように設計します。特に、近年の進化計算コミュニティにおいて、中国の研究者の存在感は非常に大きくなっています。中国国内からも制限なく利用できるように配慮します。

コンペティション方式

実問題コンペティションのトレンドは変化しうるため、多様な方式をシステム改修なしにホストできるように設計します。表 3.1 に示した実問題の 95 パーセンタイルにあたる規模をホストできるようにします。連続最適化だけでなく、組合せ最適化や木構造の変数、さらには任意のオブジェクト構造の変数にも対応できるようにします。解の評価方式としては、解を 1 つずつ送信する BBComp 方式、ソルバーのソースコードを送信する AtCoder 方式、Public Leaderboard で試行錯誤し Private Leaderboard で最終スコアを評価する Kaggle 方式のいずれにも対応でき、将来登場しうる評価方式への拡張性も備えておきます。

可用性

システム全体を稼働させたまま、新しい問題や指標を追加できるように設計します。問題プログラムは毎年新規開発することになるので、不具合がつきものです。いかなる不具合が生じても、OptHub の他の部分にまで波及しないことが求められます。また、システム全体を稼働させたまま、不具合の生じた箇所のみをアップデートできる必要もあります。競技期間が約 3 か月に渡るため、OptHub をホストするクラウドにも障害が起こりえます。一方で、専任の運用担当者がおらず迅速な障害対応は望めないため、クラウドの障害時にもメンテナンスせずに稼働し続けることが求められます。具体的には、クラウドのゾーンレベルの障害時にもシステムのどの機能も完全に停止することがなく、障害解消とともに被害箇所が自動復旧するように設計します。

セキュリティ

オンラインコンペティションでは、パスワードやアクセスログなどの機密情報が生じるため、その管理方法は慎重に検討しなければなりません。運営委員は約 10 人で、毎年交代します。それに対して、運営委員を依頼できるようなアクティブな進化計算学会員は多

めに見積もっても 100 人程度です。したがって、もし運営委員が何らかの機密情報を抱えれば、何もトラブルがなくても数年でアクティブな学会員の大半に知れ渡ってしまうことになります。そのため、運営委員が機密情報に一切アクセスせずに運用できる仕組みを構築します。また、システムが攻撃されたり、悪用される恐れもあります。最悪の事態を想定して、管理者権限が奪取された場合にもシステムを停止し、復旧できるようにします。

予算

学会から実世界ベンチマーク問題分科会に配分される予算は年 10 万円程度です。そこからコンペティションの表彰費などを差し引いた金額でシステムを運用しなければなりません。システムの運用費は年間 5 万円以内に抑える必要があります。

運用性

運営委員には本業があるため、運営に費やす時間は、運営委員 1 人あたり年間 24 時間が限度です。この限られた時間で引継ぎ、作問、イベント運営、結果の論文化を行い、残ったわずかな時間でシステムを開発・運用しなければなりません。そのため、これから運営委員になる人がすでに習得している技術だけで開発・運用できる必要があります。クラウドのサービス変更や終了があった場合にも、影響範囲を別のクラウドに移すだけでサービスを継続できるように、代替性のあるサービスでシステムを構築します。また、OptHub を終了することになった場合にも、これまでに蓄積したプログラムやデータの公開は継続することができるように、これらの保管には無料の Web サービスを利用します。

システム構成

OptHub の用途は開発・作問・競技の 3 つに分けられます。これらに必要となる機能を次の 10 個のマイクロサービスで構成しました。

ソースコードリポジトリ
: コンペティションで用いる問題や指標のソースコードを保管します。OptHub システムのコードも保管します。

コンテナイメージレジストリ
: コンペティションで用いる問題や指標のコンテナイメージを保管します。

パッケージリポジトリ
: OptHub システムを構成するソフトウェアパッケージを保管します。

認証プロバイダ
: ユーザーアカウントを管理し、Web API を利用するためのアクセストークンを発行します。

データベース
: コンペティションの問題文や参加者が送信した解などのデータを保存します。

Web API

データベースの読み書きを行うための Web API を提供します。他のサービスはすべてこの API を通して機能します。

Web サイト

コンペティションの問題文を閲覧するための Web ページを提供します。また問題文を編集するためのオンラインエディタも提供します。

CLI クライアント

コマンドラインから OptHub を操作するツールです。開発・作問・競技のいずれにも使います。

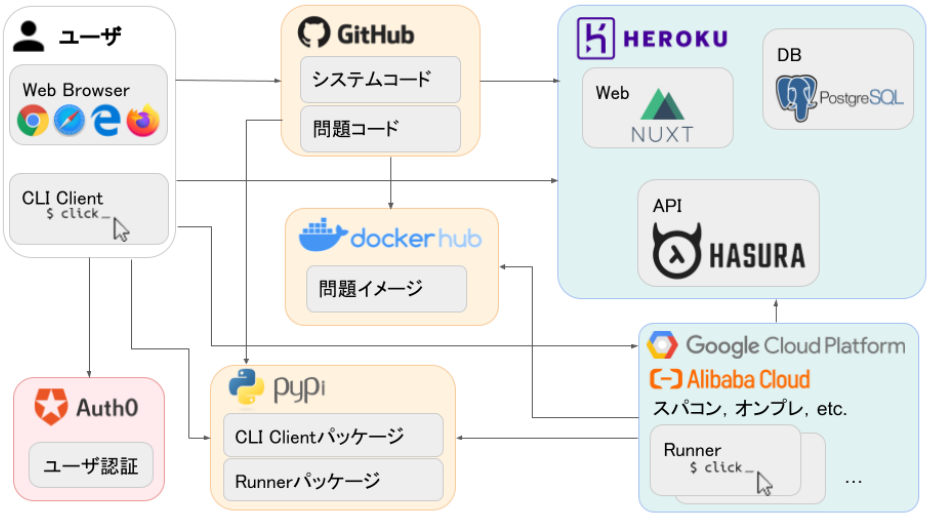
評価ランナー

問題のプログラムを実行し、目的関数と制約関数の値を計算します。

採点ランナー

指標のプログラムを実行し、スコアの値を計算します。

このうち、ソースコードリポジトリから認証プロバイダまでは既存の Web サービスを利用しています。データベース以降が独自開発したサービスです。OptHub のシステム構成図を図 3.1 に示します。以下では、これらのサービスをどのように組み合わせて OptHub の機能を実現しているかを説明します。



▲図 3.1　OptHub のシステム構成

ソースコードリポジトリ

問題や指標のソースコード、および OptHub のシステムのソースコードは、原則として GitHub のパブリックリポジトリで管理し、コミュニティベースで開発します。やむをえない事情があれば、リポジトリをプライベートとしたり、GitHub 以外のサービスを利用したり、そもそもソースコードを提供しないといった選択もできます。しかし、情報共有のためにも、可能なかぎり原則どおりに管理することを推奨しています。運営委員が提供するソフトウェアのソースコードは `https://github.com/opthub-org` にあります。

コンテナイメージレジストリ

問題や指標のコンテナイメージは、原則として DockerHub のパブリックリポジトリで管理し、ソースコードリポジトリの変更をトリガーとして自動ビルドします。やむをえない事情でそれが不可能な場合には、リポジトリをプライベートとしたり、DockerHub 以外のサービスを利用することもできます。しかし、情報共有のためにも、可能なかぎり原則どおりに管理することを推奨しています。運営委員が提供するイメージは `https://hub.docker.com/orgs/opthub` にあります。

よく使われる問題として Sphere を、指標として最良評価値と Hypervolume を公開しています。一般に、Hypervolume の計算量は目的数に対して指数関数的に増加するため、十分なチューニングが求められます。運営が公開している Hypervolume イメージは、多数目的において最高速度と定評のある PyGMO2 の実装をベースとし、更なる高速化のための前処理を加えたものです。これを利用することで、指標の計算量爆発を心配せずに多数目的問題を出題することができます。

パッケージリポジトリ

CLI クライアント、評価ランナー、採点ランナーは Python パッケージとして PyPI に登録しており、ユーザーは `pip install`コマンド 1 つでインストールできます。GitHub のこれらのソースコードリポジトリにバージョン番号のついたタグがプッシュされると、GitHub Actions によって自動的にパッケージが作成され、PyPI に登録されます。運営委員が提供するパッケージは `https://pypi.org/user/SIG-RBP/` にあります。

認証プロバイダ

ユーザーアカウントは Auth0 で管理し、OptHub の認証・認可は Auth0 から発行されるアクセストークンを用いて行っています。クレデンシャルはブラウザと Auth0 の間でのみ通信され、運営委員が開発したソフトウェア上を通過することはありません。パスワードハッシュは Auth0 社が管理しており、運営委員が知ることはできません。[*10] この

[*10] Auth0 の無料プランでは、パスワードハッシュを取得する手段はありません。詳しくは `https://auth0.com/docs/support/manage-subscriptions/export-data#user-passwords` を参照。

ように運営委員が秘密情報に一切アクセスできない設計とすることにより、情報漏洩リスクを低減するとともに、運営委員を安全に毎年交代できるようにしています。

データベース

データベースは Heroku の PostgreSQL サービスでホストしています。PaaS を利用することで運営委員の運用工数を抑えています。また、運営委員が機密情報を管理せずにすむように、データベースには公開可能な情報しか保存しないようにしています。[*11]

データベースの ER 図を図 3.2 に示します。図の各エンティティはデータベースのテーブルに対応しています。(PK)と書かれた属性はテーブルの Primary Key で、(FK)は Foreign Key です。

OptHub に登録したユーザーは User で表されます。データベースではユーザーのアカウント名（name）のみを管理しています。メールアドレスなどの非公開情報は Auth0 に保管されているためです。

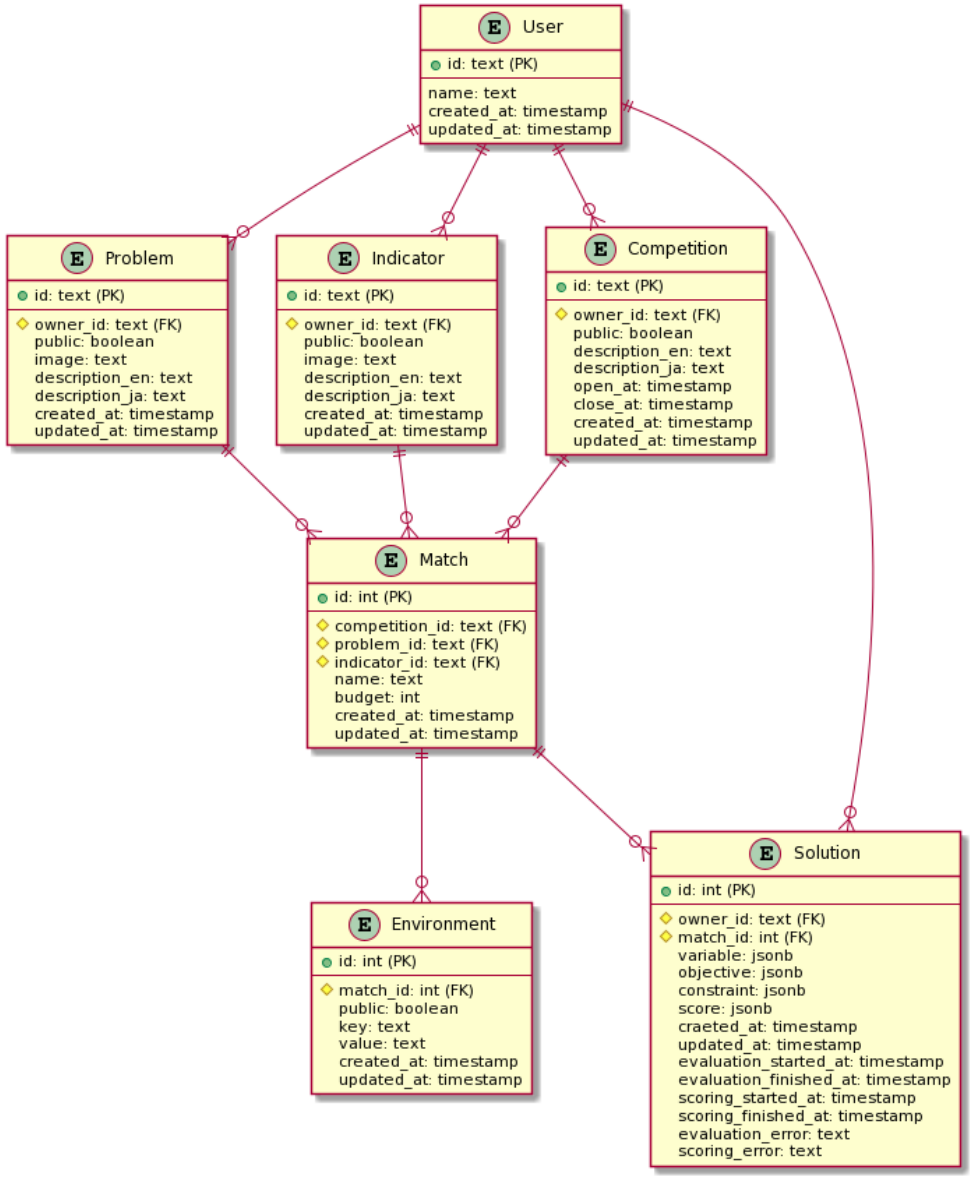
すべてのユーザーはコンペティションを開催することができます。そのためには、Competition を作成します。作成したユーザーがコンペティションの所有者（owner_id）となります。説明文（description_en, description_ja）を執筆している間は、公開フラグ（public）を偽にしておくことで、コンペティションを他人には見えないようにできます。準備ができたらこれを真にして公開します。公開されたコンペティションは、開始時刻（open_at）から終了時刻（close_at）までの間、解の提出を受け付けます。

コンペティションでは、1 つ以上の競技（Match）が出題されます。コンペティション（competition_id）の勝者は各競技のスコアに基づいて決まります。[*12]各競技には解くべき問題（problem_id）とスコア計算に使う指標（indicator_id）が定められており、競技内容を端的に表す名前（name）がついています。コンペティションに参加したユーザーは、所定の評価回数（budget）まで解を提出して、できるだけよいスコアを獲得することを目指します。

競技で使われる問題（Problem）も、ユーザー（owner_id）によって作成されます。コンペティションの開催者と問題の作成者は必ずしも同一ユーザーである必要はありません。公開フラグ（public）が真であれば、他人もこの問題を利用してコンペティションを開催することができます。そのため、説明文（description_en, description_ja）には、単に問題の説明を書くだけでなく、他人がコンペティションに使う場合も考えて利用方法を書きます。問題の実体は、目的関数値や制約関数値を計算するプログラムを格納した Docker イメージです。コンテナイメージレジストリに登録されたイメージのタグ（image）を指定することで、提出した解は指定されたコンテナで評価されます。指標（Indicator）も同様です。

[*11] 作問中のコンペティションを非公開としたり、競技期間中に他人の解を非公開としたりはしていますが、コンペティションの終了後にはすべての情報が公開されます。

[*12] 各競技のスコアからどのようにコンペティションの勝者を決めるかはシステムでは規定されていません。コンペティションの開催者にゆだねられています。



▲図 3.2　データベースの ER 図

多くの問題や指標には、調節可能なパラメータがあります。パラメータはコンテナの環境変数として与えられます。Environmentはある競技（match_id）において問題や指標のパラメータを設定するための環境変数です。公開フラグ（public）が真の場合はコンペティション参加者にパラメータ値が公開され、偽の場合は非公開となります。キー（key）にはパラメータの環境変数名を指定し、値（value）にはパラメータ値を指定します。

競技に提出された解は Solutionで表されます。ユーザー（owner_id）は競技（match_id）に対して変数（variable）を提出します。変数は問題によって評価され、結果は目的（objective）と制約（constraint）に格納されます。評価にかかった時間は評価開始時刻（evaluation_started_at）と評価終了時刻（evaluation_finished_at）からわかります。評価を終えた解は、指標によって採点され、結果はスコア（score）に格納されます。採点にかかった時間は採点開始日時（scoring_started_at）と採点終了日時（scoring_finished_at）からわかります。評価や採点においてエラーが発生した場合は、エラーメッセージが評価エラー（evaluation_error）や採点エラー（scoring_error）に記録されます。

Web API

Web API は GraphQL によってデータベースの読み書きや集計を行う機能を提供します。エンドポイントは https://opthub-api.herokuapp.com/v1/graphql です。GraphQL エンジンには Hasura を採用し、データベーススキーマから自動的に Web API を生成しています。これにより、一般に高い専門性が要求される GraphQL API の設計やリゾルバの実装をノーコードで済ませ、持ち回りの運営委員でも保守運用できるようにしました。

Hasura では、データベース操作のアクセス制御も行っています。アクセス権は、ロールと権限の組み合わせで表 3.2 のように管理されます。

▼表 3.2　アクセス制御

Role	Insert	Select	Update	Delete	Aggregate
anonymous	none	public	none	none	public
user	all	owner	owner	owner	owner
admin	all	all	all	all	all

ロールは、Web API 呼び出し時の HTTP リクエストヘッダーに応じて決まります。

anonymous

ユーザー認証されていないアクセスです。Web API へのリクエスト時にアクセストークンを渡さなかった場合には、このロールになります。

user

一般ユーザーとして認証されたアクセスです。Auth0 に一般ユーザーとしてログインしたときに発行されるアクセストークンを Web API へのリクエスト時に渡すことで、このロールになります。

admin

管理者ユーザーとして認証されたアクセスです。Auth0 に管理者ユーザーとしてログインしたときに発行されるアクセストークンを Web API へのリクエスト時に渡すことで、このロールになります。

権限には次の種類があります。

none
: いかなるレコードも操作できません。

public
: 公開レコードのみを操作できます。

owner
: 公開レコードおよび自身が所有者であるレコードのみを操作できます。

all
: すべてのレコードを操作できます。

権限のないレコードは操作を受け付けないだけでなく、存在自体が不可視になります。たとえば、user ロールで他人の解を削除しようとすると、「この解を削除する権限がありません」ではなく「解が見つかりません」というエラーになります。anonymous ロールで解の数をカウントした場合、公開されている解の件数が返されます。user ロールで解の数をカウントした場合、公開されている解に自身の非公開の解も加えた件数が返されます。

公開レコードと非公開レコードの定義はテーブルによって異なります。コンペティション、問題、指標は所有者の意思で公開・非公開を決めます。これらのレコードにはそのための公開フラグがあり、この値が真であれば公開レコード、さもなければ非公開レコードとして扱われます。

競技の公開・非公開はコンペティションに準じますが、加えて競技期間より前には内容を秘匿する必要があります。そのため、競技には公開フラグはなく、その公開・非公開は所属するコンペティションによって決まります。コンペティションの公開フラグが真かつ開始時刻以降ならば、競技は公開レコードとして扱われます。それ以外の場合は非公開レコードとして扱われます。

環境変数は、基本的に競技開始と同時に公開しますが、一方で競技期間中は秘匿したい環境変数もあります。秘匿した環境変数も、競技終了後にはタネ明かしのために公開しています。そのため、環境変数には公開フラグがありますが、その公開・非公開は所属するコンペティションにも依存します。[*13]次のいずれかの条件を満たすならば、環境変数は公開レコードとして扱われます。それ以外の場合は非公開レコードとして扱われます。

- コンペティションの公開フラグが真、かつ開始時刻以降、かつ環境変数の公開フラグが真
- コンペティションの公開フラグが真、かつ終了時刻以降

解は競技終了後に自動的に公開されます。そのため、解には公開フラグはなく、その公開・非公開は所属するコンペティションによって決まります。[*14]コンペティションの公開

[*13] 環境変数は 1 つの競技に所属し、競技は 1 つのコンペティションに所属します。ここでは、この 2 つの所属関係を辿って「環境変数が所属するコンペティション」とよんでいます。

[*14] 解も直接的には競技に所属しますが、環境変数と同様に「解が所属するコンペティション」とよんでいま

フラグが真かつ終了時刻を過ぎると、解は公開レコードとして扱われます。それ以外の場合は非公開レコードとして扱われます。加えて、競技終了後には、解の提出を受け付けないようにする必要もあります。そのため、user ロールによる解の Insert は、コンペティションの開始時刻以降かつ終了時刻以前のみ許可する特別な権限を設定してあります。

このように、レコードの公開・非公開を一定のルールに基づいて自動的に判定することで、コンペティションの開始時や終了時に運営委員の立ち合いが不要となり、情報公開のためにプレイヤーに作業を強いる必要もなくしています。

Web サイト

Web サイトは `https://ec-comp.jpnsec.org/ja/` にあります。OptHub に登録されているユーザー、コンペティション、問題、指標を見ることができます。また、ログインすることで、コンペティション、問題、指標を作成・編集できるようになります。チュートリアルコンペティション[*15]をプレイすれば、ひととおりの使い方が分かるようになっています。

Web サイトは Nuxt.js を用いて Progressive Web Application（PWA）として実装し、Heroku でホストしています。ソースコードは GitHub で管理し、main ブランチの更新をトリガーとして自動的に Heroku にデプロイされます。本番環境へのデプロイ前に、ステージング環境にて OWASP ZAP による脆弱性スキャンを行っています。

万が一 Heroku がダウンしたとしても、参加者は PWA のオフラインキャッシュによって問題文を読むことができます。Heroku の復旧後、コンテナのオートリスタートによって Web サイトも自動的に復旧します。そのため、短時間の障害であれば、運営委員が障害対応に追われることはありません。

CLI クライアント

コマンドラインから OptHub を操作するためのツールです。出題、競技、管理のいずれにも使用します。インストールすると、`opt`というコマンドが追加され、いくつかのサブコマンドによって OptHub を操作します。サブコマンドは用途に応じた 3 層構造になっています。GraphQL のクエリを直接発行できる低水準コマンド（`opt gql`）、データベースのテーブル操作に対応する CRUD コマンド（`create, list, update, delete`）、コンペティションの登録や解の送信などを行う高水準コマンド（`organize, submit, status`）です。出題者や競技者は基本的には高水準コマンドのみで必要な作業を行えます。管理者はさまざまなトラブルを解決するために CRUD や低水準コマンドを利用します。

CLI クライアントの初回利用時にはユーザー登録・ログインが求められます。ユーザー登録・ログインはブラウザ経由で行います。ユーザーが入力したメールアドレスやパスワードは、ブラウザと Auth0 の間でのみやりとりされ、CLI クライアント上を通過する

す。

[*15] `https://ec-comp.jpnsec.org/ja/competitions/tutorial`

ことはありません。CLI クライアント上でユーザー登録・ログイン処理を行わない理由は、少ない開発コストでセキュリティを担保するためです。運営委員の限られた時間では、自身の開発したソフトウェアやその依存ライブラリを十分に監査することはできません。コーディングミスやサプライチェーン攻撃などによる情報漏洩を防ぐため、クレデンシャルが CLI クライアント上を通過しない設計にしています。

評価ランナーと採点ランナー

評価ランナーは解の評価を行う常駐プロセスです。定期的に解をポーリングし、未評価の解が見つかると、その競技の問題イメージを取得してコンテナを起動し、解の評価を行います。採点ランナーは解の採点を行う常駐プロセスです。定期的に解をポーリングし、評価済かつ未採点の解が見つかると、その競技の指標イメージを取得してコンテナを起動し、解の採点を行います。ランナーを 2 つに分けた理由は、評価と採点では計算負荷が異なるため、それぞれ異なるスケーリングが必要になるためです。

これらのランナーは、Python と Docker が動く環境ならばどこでもホストできます。実問題の要求スペックはまちまちなため、安価な FaaS、IaaS のカスタム環境、高性能なスーパーコンピュータ、ポート開放が禁止されたオンプレミスのマシンまでさまざまな計算機環境で動作するようにしています。計算ノードは出題内容に応じて選ぶことになりますが、通年稼働させるノードには Google Cloud Platform と Alibaba Cloud の無料 IaaS インスタンスを使用しています。異なる会社の異なるリージョンでホストすることで、自然災害や大規模停電はもちろん、運用ミスやサイバー攻撃による全社的な障害に陥っても停止しないようにしています。一般論としてマルチクラウドは割高になりがちですが、本システムでは各社の無料枠を合算することで、1 社よりもむしろ低コストに抑えています。

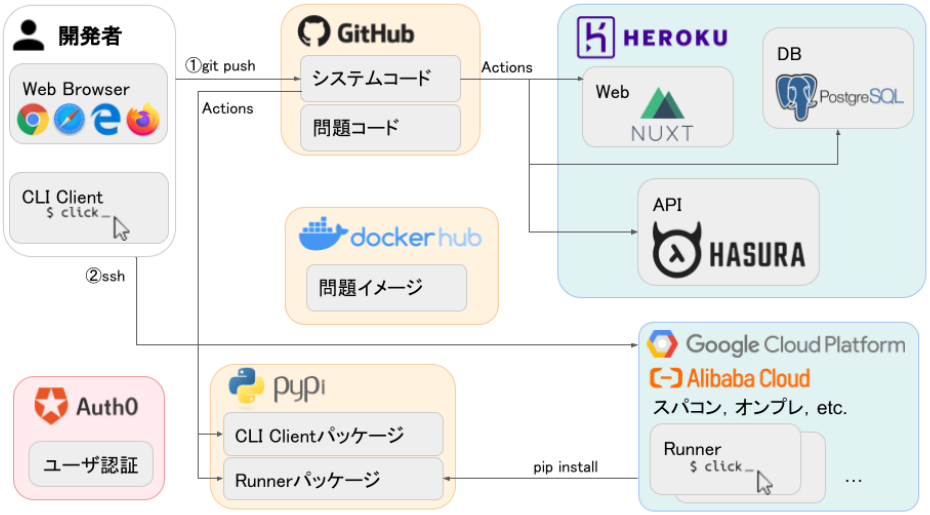
利用の流れ

最後に、これらのサービスを組み合わせて実際の運用がどのような流れで行われるかを説明します。運営委員によるシステムの開発、出題者によるコンペティションの作問、コンペティション参加者による競技の流れについて順に紹介します。

開発の流れ

開発の流れを図 3.3 に示します。OptHub のソースコードはすべて GitHub に保管されています。リポジトリは Web サイト[*16]、Web API[*17]、CLI クライアント[*18]、評価ランナー[*19]、採点ランナー[*20]の 5 つに分かれています。

[*16] https://github.com/opthub-org/opthub
[*17] https://github.com/opthub-org/opthub-api
[*18] https://github.com/opthub-org/opthub-client-cli
[*19] https://github.com/opthub-org/opthub-evaluator
[*20] https://github.com/opthub-org/opthub-scorer



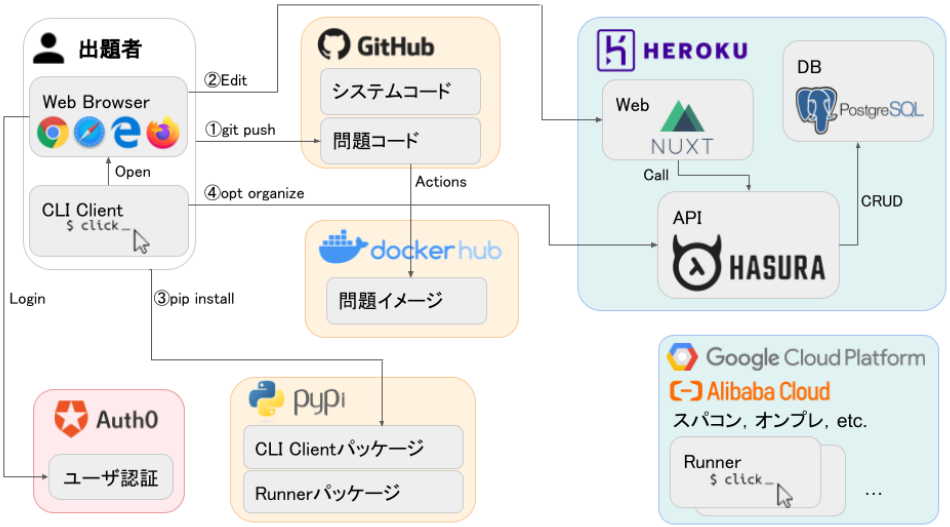
▲図 3.3　OptHub のユースケース：開発の流れ

①開発者はこれらのリポジトリをクローンして開発し、変更点を GitHub に Push します。GitHub の main ブランチが更新されると、コードは GitHub Actions によって自動的にデプロイされます。Web サイト、Web API、データベースは Heroku にデプロイされます。CLI クライアント、評価ランナー、採点ランナーはパッケージ化され、PyPI に登録されます。

②開催されるコンペティションの内容に応じた計算ノードを用意します。コンペティションで使用する問題や指標の計算負荷を見積もり、基本的には複数社のクラウドに適切な IaaS インスタンス等を作成してホストします。負荷が大きい場合には、大学等のスーパーコンピュータを借りることもあります。知的財産やライセンス、ソフトウェア実装等の都合上、問題や指標を出題者の組織外に出すことができないケースもありえます。その場合にはオンプレミスのマシンを計算ノードとすることもできます。計算ノードに SSH でログインし、PyPI に登録された評価ランナーと採点ランナーをインストールします。評価ランナーと採点ランナーを必要な数だけ起動することで、評価と採点を並列化できます。

作問の流れ

作問の流れを図 3.4 に示します。



▲図 3.4　OptHub のユースケース：作問の流れ

①出題者は GitHub の問題テンプレートを参考にして問題プログラムを開発し、ソースコードを GitHub に登録します。問題は Docker コンテナ内で実行されるプログラムであり[*21]、標準入力から JSON 形式で解を読みこみ、目的関数値や制約関数値を計算し、結果を標準出力に JSON 形式で書き出します。GitHub の main ブランチが更新されると、Docker イメージがビルドされ、DockerHub のパブリックリポジトリに push されます。[*22]もし指標も追加するならば、同様にしてコンテナイメージを公開します。

②ブラウザから OptHub の Web サイトにアクセスし、ユーザー登録・ログインします。ログインすると、Auth0 から Web API のアクセストークンが発行されます。これにより、Web API を通してデータベースに書き込めるようになり、Web サイト上で問題や指標などが作成できるようになります。Web サイトで問題を新規作成し、先ほど登録したコンテナイメージのタグを登録します。問題の説明文も記述します。Web サイトで作成された問題は Web API を通してデータベースに登録されます。指標も同様です。最後に、コンペティションを登録します。コンペティションに 1 つ以上の競技を追加し、それぞれの競技で用いる問題と指標、環境変数を指定します。なお、コンペティションには他

[*21] コンテナ化に適さないケースのために、非コンテナ環境で問題プログラムを実行する方法もあります。
[*22] ここでは OptHub で推奨される方法を述べました。公開が難しい場合は、コンテナイメージをプライベートレジストリに置くこともできます。

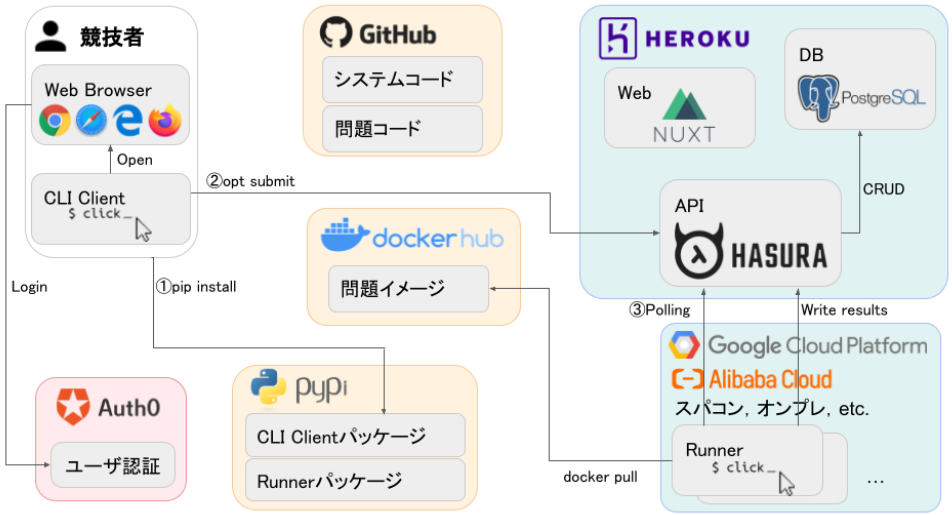
人が登録した問題と指標も利用できます。これにより、自分自身で問題や指標を用意できない場合でも、他人の問題と指標を利用して簡単にコンペティションを開催することができます。

③また、以上の作業は CLI クライアントを通して行うこともできます。多数の競技を登録するときは CLI クライアントが便利です。まず、自身のマシンで `pip install`コマンドを実行して、CLI クライアントを PyPI からインストールします。すると `opt`コマンドが利用できるようになります。

④ `opt organize`コマンドを実行すると、コンペティションを対話的に登録することができます。なお、初回起動時はブラウザを使ってユーザー登録・ログインが求められます。

競技の流れ

コンペティションの参加者が競技を行う際の流れを図 3.5 に示します。



▲図 3.5　OptHub のユースケース：競技の流れ

①参加者は自身のマシンで `pip install`コマンドを実行し、PyPI から CLI クライアントをインストールします。すると `opt`コマンドが利用できるようになります。`opt`コマンドをはじめて実行すると、ユーザー登録・ログインを促すメッセージが表示されます。メッセージにしたがって、Auth0 にユーザー登録・ログインします。ログインすると、Auth0 からアクセストークン（とリフレッシュトークン）が発行されます。CLI クライアントはこれを保存し、以降の Web API のアクセスに利用します。

②参加者は `opt submit`コマンドを使って解を送信します。CLI クライアントはアクセストークンを使用して Web API を呼び出し、解がデータベースに登録されます。CLI ク

ライアントは解の評価と採点が終わるまで Web API をポーリングします。

③評価ランナーは未評価の解をポーリングしています。解の登録を検知すると、その解を評価するために必要な問題イメージをコンテナレジストリから Pull して実行します。計算された評価値をデータベースに書き込みます。採点ランナーは評価済かつ未採点の解をポーリングしています。新しく評価された解を検知すると、その解を採点するために必要な指標イメージをコンテナレジストリから Pull して実行します。計算されたスコアをデータベースに書き込みます。CLI クライアントはスコアの書き込みを検知すると、ポーリングを終了し、評価値とスコアを表示します。

3.4 組織運営

組織運営もまた進化計算コンペティションを支える技術のひとつ、ということで分科会員の一年を紹介します。おおまかな年間スケジュールは表 3.3 のようになっています。分科会の設置期間は 10 月から翌年 9 月までの 1 年間で、毎年期限が来ると進化計算学会に期間更新を申請します。分科会員の交代も原則的にはこのタイミングで行われます。しかしコンペティションの運営を理解するうえでは、1 月から 12 月までを 1 サイクルと捉えた方が分かりやすいです。

▼表 3.3　分科会員の一年

役割	1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
幹事	出題者探し	➡	➡	➡	スポンサー探し	➡	➡	活動報告	デモ	広報	➡	表彰
作問	問題管理	➡	➡	作問	➡	➡	➡	➡	デモ	出題	問合せ対応	講評
論文	論文誌	➡	➡	➡	➡	国際会議	➡	➡	➡	➡	➡	➡
開発	開発	➡	➡	➡	計算機調達	➡	➡	➡	デモ	問合せ対応	➡	結果集計

分科会員は約 10 人で 4 つのチームに分かれています。幹事チームは 2 人で、主に全体のマネジメントや会計、対外活動を担当します。1 人は代表幹事で、出題者やスポンサー探し、広報、イベントの司会とアナウンス、学会への年間活動報告、分科会 ML や Slack の管理などを行います。もう 1 人は幹事で、代表幹事の補佐と会計、アンケートや表彰状の作成と発送などを行います。

作問チームは 3 人ですが、出題者にも 2 人程度加わっていただき、5 人程度でコンペ

ティションの問題を作成します。まず、幹事がこれまでに収集した実問題のストックの中から、出題する問題を検討します。開催日の決まったコンペティションに向けて 1 から作問することはリスクが高く、何より出題者に大きな負担をかけてしまいます。そこで、問題ストックの段階で、出題者にはコンペティションを意識しない形での論文化を進めていただくことを推奨しています。分科会 Slack には問題ストックごとのチャンネルがあり、そこで出題者と定期的に連絡を取り合いながら、論文の準備を進めます。論文が出版されたら出題候補とし、約半年かけて問題をコンペティション用に仕立て直します。出題者と毎月ミーティングを行って作問を進めます。出題者側で問題文ドラフトを作成し、問題プログラムを開発します。分科会側で問題文を推敲・英訳し、問題プログラムを Docker 化して OptHub で動作検証します。

論文チームは、昨年の作問チーム（出題者含む）がそのままシフトして務めます。コンペティションの結果をまとめて、国内外の論文誌や会議に投稿します。作問と論文は時間を要するため、直列に進めるとそれだけで丸 1 年かかってしまいます。チームを分けて、今年のコンペティションの作問と昨年のコンペティションの論文化を並行して進めています。

開発チームは 2 人で、OptHub のシステムを担当します。1 人は Web 担当で、Web サイトと Web API を開発・保守します。もう 1 人はクライアント担当で、CLI クライアントと評価・採点ランナーを開発・保守します。作問チームと連携して、今年のコンペティションをホストするために必要な計算機を調達します。コンペティションの期間中には、システム関係の問い合わせに対応します。コンペティションの結果を集計して、結果発表の資料を作ります。

1 年を通して、学会と連動したイベントが 2 つあります。例年 9 月に開催される進化計算学会研究会ではデモを行います。1 時間のセッション枠をいただいて、今年のコンペティションの問題を発表し、チュートリアルコンペティションをプレイするハンズオンを行います。例年 12 月に開催される進化計算シンポジウムでは結果発表を行います。3 時間のセッション枠をいただいて、今年のコンペティションの結果を発表し、表彰式や講評、来年に向けたディスカッションなどを行います。

こうして見ると 1 年を通してさまざまな仕事がありますが、誰もが無理なく続けられることを第一に運営しています。具体的には、1 人当たりの年間作業時間が必ず 24 時間以内となるように作業を分担しています。毎月 1 時間のオンラインミーティングに加えて、個人の作業時間が 12 時間以内の計算です。新規メンバーの就任時に、年間スケジュールと作業時間上限について説明し、オーバーしそうな場合は正直に申告してもらうルールにしています。

分科会員の任期は 2 年です。毎年約半数ずつメンバーを入れ替えることで、2 年目と新任がペアになってノウハウを引継ぎながら交代していきます。本人が希望すれば任期の延長もできます。開発チームは専任で、その他のチームは 1 年ごとに幹事→作問→論文→幹事→…とローテーションします。

基本的にボランティア活動ではありますが、熱意ある個人に不利益をもたらすことがな

いように心がけています。幹事・作問・論文チームは、ローテーションを通して全員が論文の出版業績を得ることができるようになっています。開発チームは、スポンサーとして所属組織を宣伝をする代わりに業務として分科会活動に取り組めるようにお願いしたり、それが難しい場合には謝金をお支払いしています。分科会の任期を終えたときに、誰もが何かしらプラスになったと感じてもらえることを願っています。

3.5 おわりに

本稿では、進化計算コンペティションの運営のために作成したシステム OptHub を紹介しました。進化計算学会員へのアンケートを通して、実問題最適化コンペティションをホストするために求められるシステム要件をまとめ、既存のコンペティションシステムを評価しました。検討の結果、大規模実問題を Human-in-the-Loop 方式でホストし、コンペティションを通して収集したプログラムやデータを再利用しやすい形で公開するためには新たなシステムが必要と判断し、OptHub を開発しました。OptHub は、多数のマイクロサービスを連携して作られたシステムです。システムの大部分には多くの人が使い慣れた既存の Web サービスを利用することで、独自開発を最小化し、学習コストや運用コストを抑えています。解の評価を処理する計算機環境は、出題される問題の性質と予算に応じてクラウドの FaaS から IaaS、オンプレミスやスーパーコンピュータまで柔軟に切り替えられるようになっています。

OptHub を利用することで、運営者・出題者・競技者が相互に入れ替わりながら、コミュニティ全体で進化計算コンペティションを活性化してゆける体制を目指しています。本稿によって多くの方に OptHub を知っていただくことで、進化計算コンペティションをより安定的に運営できる体制を作り、将来的には国際的なコンペティションのプラットフォームへと発展させていきたいと思っています。

最後に、進化計算学会実世界ベンチマーク問題分科会は会員を募集しています。進化計算コンペティションの運営に限らず、世の中の実問題を収集し、コミュニティの共有知として蓄積する活動に興味がある方を求めています。ご興味のある方は sig-rbp@googlegroups.com までご連絡ください。

謝辞

アンケートにご回答いただきました皆様、アンケートの集計を手伝っていただきました田邊遼司先生（横浜国立大学）、OptHub のテストプレイにご協力いただきました進化計算学会員の皆様、コンペティションを運営してくださいました進化計算学会実世界ベンチマーク問題分科会の皆様に感謝します。

アンケート結果詳細

「出題者アンケート」に出題者アンケートの結果を、「競技者アンケート」に競技者アンケートの結果を示します。

出題者アンケート

7 名から回答を得られました。14 個の質問項目と回答を次に示します。選択式の回答は数の多い順に並べ替えています。使用したアンケートフォームは `https://forms.gle/uDCVpZprcfU4yhzW8` を参照してください。

Q1. あなたの職業は？（単一選択）

- 企業勤務（民間研究所を含む） ··· 6
- 大学教員（公的研究機関を含む） ··· 1
- 学生〜PD ··· 0

Q2. あなたの身の回りでは進化計算を実務にどのように活用していますか？（自由記述）

- 製品開発
- 製品設計の一部に使用している
- 製品設計、パラメータ決定
- 製品設計の一部に対して活用している
- 設計ツールとして、シミュレーション精度改善（コリレーション、同定問題）
- 研究活動
- ほとんど使用していない

Q3. 進化計算は産業界の期待に応えられていると思いますか？（単一選択）

結果を表 3.4 に示します。

▼表 3.4　出題者 Q3. 進化計算は産業界の期待に応えられていると思いますか？（単一選択）

質問	期待を大きく上回る	期待以上	期待通り	期待以下	期待を大きく下回る
他の手法よりも良い解が見つかる	0	1	5	1	0
結果が信頼できる	0	1	3	3	0
自分が直面する問題に適した手法がある	0	0	5	2	0
使いたい手法の実装が用意されている	0	0	4	3	0
1 つの手法で幅広い問題が解ける	0	2	4	0	1
問題の定式化が簡単	0	2	0	2	3
結果の解釈が簡単	0	0	4	2	1
実用的な評価回数で満足な解が見つかる	0	0	2	4	1
進化計算の専門家でなくても使える	0	0	1	3	4
いつどの手法を使うべきか分かりやすい	0	0	0	5	4

Q4. 他に進化計算に期待することはありますか？（自由記述）

- DeepLearning にはできない ManuTech 分野への活用
- ベストプラクティスの集積
- 想定外にどうする（計算をスタートさせたが、ある時、環境が変わった。その時、それまでの計算を活かして、定式化を変えても、進化は進むのか？ その結果に意味はあるのか？）
- 人間の設計者の想定を超えた良解を発見してくれる。進化計算を経て得られたデータを解析することで対象の性質をよく知ることができる。
- 解析手法が得られる

Q5. コンペに期待することは何ですか？（複数選択）

- 研究開発のヒントになる ··· 6
- 産学の知識・人材交流 ··· 6
- 様々なアルゴリズムの実力を知れる ··· 5
- 提供した問題についての理解が深まる ··· 5
- 他人のアプローチが学べる ··· 4
- 実問題全般への興味を喚起できる ··· 4
- 実問題のリアリティを追求した問題設定 ··· 4
- 競技の公平性 ··· 3
- 自社の PR になる ··· 3
- 進化計算について学べる ··· 2
- 結果（順位）の納得感 ··· 2

- 競技の面白さを追求した問題設定 ··· 2
- 優秀な学生を見つけられる ··· 1
- その他：学生に実問題の難しさを伝えたい

Q6. 期待した結果は得られましたか？（得られそうですか？）その理由は何ですか？（自由記述）

- 現在は準備段階における評価回数に制限がなく、パラメータチューニングの良し悪しが結果に影響するので、準備段階も含め評価回数にカウントされると楽しいと思います。
- 実問題への興味が高まっている昨今では、期待した結果が得られると思われる。
- 得られた。多数のアルゴリズムを比較することで、アルゴリズム内のどの要素が効果的か判断できたため。
- わからない。コンペの後に、提供者、参加者、聴講者の感想が聞きたい。その感想をテキストマイニングしたいね。
- 実問題への興味を喚起できた点で◯。これまでのコンペはそれぞれの問題が対象に特徴的な課題要素を持っていて、対象ごとの工夫が必要だったから。

Q7. 身の回りにはコンペに提供できそうな問題がありますか？（単一選択）

- 提供できそうな問題はあるが、提供したことはない ··· 4
- 提供できそうな問題があり、提供したことがある ··· 2
- 提供できそうな問題がない ··· 1

Q8. 問題を提供するうえでのネックは？（複数選択）

- 知財などの理由から、提供の許諾を得ることが難しい ··· 6
- コンペに適した形にプログラムを改修するのが大変 ··· 5
- コンペに適した形に問題を定式化しなおすのが大変 ··· 3
- 時間がない（手間がかかりすぎる） ··· 3
- 問題を説明することが難しい ··· 3
- 計算機が足りない（評価プログラムが重い、評価回数が多い） ··· 1
- 問題を提供するメリットが小さい ··· 1
- コンペに興味がない ··· 0
- 提供できそうな問題を探すのが難しい ··· 0
- 問題を公開することで悪評が立つ懸念がある ··· 0
- 公開すべき情報が多すぎる ··· 0
- 開催時期が合わない ··· 0
- その他：会社のトップマネージメントの理解（これが一番かもしれない）

Q9. 問題を提供するとしたら、どんな情報をコンペ HP で公開できますか？（複数選択）

- 問題の背景（社会的ニーズや解くことの価値） ··· 4
- 代表的なソルバでの最適化結果 ··· 4
- 設計変数・目的関数・制約関数の物理的意味 ··· 3
- 目的関数や制約関数の式（シミュレーションの場合、微分方程式など） ··· 1
- 既存の設計解（実際の製品に相当する解） ··· 1
- 問題定義の記載された論文 ··· 1
- 問題を解いてみた結果が記載された論文 ··· 1
- コンパイル済み実行バイナリ ··· 1
- ソースコード ··· 1
- 専門家の典型的な解き方に関する説明 ··· 0
- その他：世間で公表されているテスト関数の中で、実問題に比較的近いものを選択。またはテスト関数を例として「産業界ならこうなりそう」という経験的コメントをする。

Q10. 適切な開催時期・期間は？（自由記述）

- コロナウイルス騒動があるので今年は忙しさの見込みが不明瞭
- 6 月〜11 月の間で 1 か月ほど
- 10 月〜12 月の間が、企業としては参加しやすい

Q11. 評価方法の変更について、賛成するものにチェックしてください。（複数選択）

- 解をサーバ側で評価すること ··· 5
- 総評価回数に上限を設けること（N 回評価で競技終了） ··· 4
- コンペ成績に影響しない事前問題検討の機会を作ること ··· 4
- 1 試行の評価回数を少なめにすること（リアリティ追求のため） ··· 4
- 1 試行の評価回数を多めにすること（試行錯誤の余地を残すため） ··· 3
- 評価した解とその評価値をすべてサーバに記録すること ··· 3
- 記録した解をコンペ後に Web 公開すること ··· 3
- 試行回数を 1 試行だけにすること ··· 0
- その他：計算リソースが潤沢で勝った場合だけでなく、限られたリソース（評価回数）で勝てるルールがあっても良いと思う。
- その他：こちらよりも Kaggle 形式もいいと思います。Kaggle は、工夫点やソースコードも公開されるので、それをもとに改善をしていける相乗効果が狙える。もちろん、論文性、特許性があるので、その部分のプロテクト（ルール化）は必要かと思いますが。

Q12. 提供できそうな問題をお持ちであれば、その性質にチェックしてください。（複数選択）

- 多数目的 ··· 3
- 多数変数 ··· 2
- 連続変数 ··· 2
- 連続・整数の混合変数 ··· 2
- 実行時間制約（総評価回数ではなく総評価時間で打切り） ··· 2
- 不確実問題（目的や制約にノイズがある） ··· 1
- 多数制約 ··· 1
- 整数変数 ··· 1
- バイナリ変数（ナップサック問題など） ··· 1
- カテゴリ変数（N 択変数の組合せ問題など） ··· 1
- 順列変数（巡回セールスマン問題など） ··· 0
- ツリー変数（遺伝的プログラミングなど） ··· 0
- より複雑な構造の変数（任意のオブジェクト） ··· 0
- 微分可能（導関数の式が与えられる） ··· 0
- 多因子最適化（複数の目的関数の単一目的最適解を求める） ··· 0
- 動的問題（目的や制約が時間変化する） ··· 0
- その他 ··· 0

Q13. 実問題のプログラムで使用する OS/ツール/言語にチェックしてください。（複数選択）

- Linux ··· 1
- MacOS ··· 1
- Windows ··· 5
- (Windows にチェックした方のみ) WSL / Cygwin / MinGW / etc. ··· 1
- Git / Mercurial / Subversion / CVS / etc. ··· 2
- Docker ··· 0
- Make / Maven / Gradle / Bazel / etc. ··· 1
- C / C++ ··· 4
- C# / その他の .net 系言語 ··· 0
- Fortran ··· 0
- Go ··· 0
- Java / Clojure / Kotlin / Scala ··· 1
- MATLAB / Octave / Scilab ··· 2
- Python ··· 5

- R ··· 1
- Ruby ··· 1
- その他：SQL

Q14. コンペ全般についてご意見があれば何でもお伝えください。（自由記述）

- 問題そのものを公表することについて、組織の承諾を得るのは難しそうなので、何か別の形をとれないか悩んでいる。
- 「実問題」としてどこに重きをおいて考えるか、に興味があります。設計者（人）の評価は式やプログラムに反映できない部分（経験則など）があり、そこをどうコンペに入れ込むか、が難しいと思っています。
- BBComp も同様ですが、コンペは、最良の f(x) を求めるものになっている。今は、時代が変わり、最適解集合をデータ分析（機械学習など）することが、スタンダードとなっている。今、進化計算に求められているのは、より良いデータセット取得のためのサンプリング手法としての進化計算である。f(x) で優越をつけるのではない、何かにシフトすべき時代になっていると思う。アンケートには、回答したが、f(x) の優越からの脱却を今後のコンペティションに期待する（企業からも問題を提供しやすい。解を得るだけだと価値が低いため）

競技者アンケート

10 名から回答を得られました。10 個の質問項目と回答を次に示します。選択式の回答は数の多い順に並べ替えています。使用したアンケートフォームは `https://forms.gle/9njx5zhm9Wk9TTdh8` を参照してください。

Q1. あなたの職業は？（単一選択）

- 大学教員（公的研究機関を含む） ··· 5
- 企業勤務（民間研究所を含む） ··· 4
- 学生〜PD ··· 1

学生からの回答は少なかったのですが、卒業のためと思われます。例年の競技者は学生が多いです。

Q2. どの問題に取り組んだことがありますか？（単一選択）

表 3.5 のようになりました。

▼表 3.5　コンペティションへの取り組み。

問題	最適解データをコンペに提出した	コンペ期間中に問題を解いた	コンペ期間外に問題を解いた	コンペ HP を読んだ	取り組んでいない
2017 単目的	1	0	0	4	5
2017 多目的	1	0	0	5	4
2018 単目的	2	0	0	2	6
2018 多目的	1	0	0	3	6
2019 単目的	2	0	1	3	4
2019 多目的	1	0	1	3	5

Q3. コンペに期待することは何ですか？（複数選択）

- 参加しやすさ ··· 8
- 実問題について学べる ··· 8
- 他人のアプローチが学べる ··· 6
- 自分のアルゴリズムを試せる ··· 6
- 様々なアルゴリズムの実力を知れる ··· 4
- 研究開発のヒントになる ··· 5
- 競技の公平性 ··· 3
- 結果（順位）の納得感 ··· 2
- 実問題としてのリアリティを追求した問題設定 ··· 3
- 競技としての面白さを追求した問題設定 ··· 2
- 産学の知識・人材交流 ··· 4
- 対外的な実績ができる（入賞など） ··· 7
- その他：進化計算分野への学生の足掛かり
- その他：学生のモティベーションが上がる

Q4. コンペに参加するうえでのネックは？（複数選択）

- 時間がない（手間がかかりすぎる） ··· 5
- 準備が大変（インストール作業など） ··· 4
- 計算機が足りない（評価プログラムが重い、評価回数が多い） ··· 3
- 悪い結果を残したくない ··· 1
- 入賞のメリットが小さい ··· 1
- コンペに興味がない ··· 0
- 解いてみたい問題が出題されない ··· 0
- 開催時期が合わない ··· 0
- 問題を理解することが難しい ··· 0

- その他：個人かつオンラインで参加できるかどうか
- その他：年度毎に結果がまとめられていくページが存在しない（毎年ばらばらに表彰されている）

Q5. コンペ問題に取り組むための予備知識（HP で公開する情報）として、何が必要だと思いますか？（複数選択）

- 設計変数・目的関数・制約関数の物理的意味 ··· 8
- 問題の背景（社会的ニーズや解くことの価値） ··· 7
- 代表的なソルバでの最適化結果 ··· 6
- 目的関数や制約関数の式（シミュレーションの場合、微分方程式など） ··· 5
- 既存の設計解（実際の製品に相当する解） ··· 5
- 専門家の典型的な解き方に関する説明 ··· 4
- 問題定義の記載された論文 ··· 3
- 問題を解いてみた結果が記載された論文 ··· 3
- コンパイル済み実行バイナリ ··· 3
- ソースコード ··· 3
- その他 ··· 0

Q6. 適切な開催時期・期間は？（自由記述）

- 6 月上旬公開、11 月下旬締め切りで 6 か月
- 6 月〜12 月の間で 1〜2 ヶ月ほど
- 8 月
- 12 月のシンポジウム併催でよい
- 今のままは結構よいです。
- 10 月から 12 月で 1 ヶ月ほど

Q7. 評価方法の変更について、賛成するものにチェックしてください。（複数選択）

- 解をサーバ側で評価すること ··· 6
- 総評価回数に上限を設けること（N 回評価で競技終了） ··· 6
- 記録した解をコンペ後に Web 公開すること ··· 6
- コンペ成績に影響しない事前問題検討の機会を作ること ··· 5
- 1 試行の評価回数を少なめにすること（リアリティ追求のため） ··· 3
- 1 試行の評価回数を多めにすること（試行錯誤の余地を残すため） ··· 3
- 評価した解とその評価値をすべてサーバに記録すること ··· 3
- 試行回数を 1 試行だけにすること ··· 1
- その他：コンペでは公平性。

Q8. 出題されたら参加したい問題にチェックしてください。（複数選択）

- 連続変数 ··· 9
- 不確実問題（目的や制約にノイズがある） ··· 8
- 連続・整数の混合変数 ··· 5
- 整数変数 ··· 4
- 多数目的 ··· 4
- 動的問題（目的や制約が時間変化する） ··· 4
- 多数変数 ··· 3
- 多数制約 ··· 3
- 多因子最適化（複数の目的関数の単一目的最適解を求める） ··· 3
- バイナリ変数（ナップサック問題など） ··· 3
- カテゴリ変数（N 択変数の組合せ問題など） ··· 3
- ツリー変数（遺伝的プログラミングなど） ··· 3
- 順列変数（巡回セールスマン問題など） ··· 2
- より複雑な構造の変数（任意のオブジェクト） ··· 2
- 実行時間制約（総評価回数ではなく総評価時間で打切り） ··· 2
- 微分可能（導関数の式が与えられる） ··· 0
- その他：評価回数が極端に少ない (<200)

Q9. コンペに使用する OS/ツール/言語にチェックしてください。（複数選択）

- Linux ··· 4
- MacOS ··· 5
- Windows ··· 4
- (Windows にチェックした方のみ) WSL / Cygwin / MinGW / etc. ··· 2
- Git / Mercurial / Subversion / CVS / etc. ··· 5
- Docker ··· 3
- Make / Maven / Gradle / Bazel / etc. ··· 2
- C / C++ ··· 2
- C# / その他の .net 系言語 ··· 0
- Fortran ··· 1
- Go ··· 0
- Java / Clojure / Kotlin / Scala ··· 2
- MATLAB / Octave / Scilab ··· 2
- Python ··· 10
- R ··· 3
- Ruby ··· 2
- その他 ··· 0

Q10. コンペ全般についてご意見があれば何でもお伝えください。（自由記述）

- 運営の方々の苦心に感謝いたします。
- 今までコンペの準備ありがとうございます。特に最近英語の説明も大変感謝します。

参考文献

[1] 進化計算学会, 宇宙航空研究開発機構. 進化計算コンペティション 2017.
http://is-csse-muroran.sakura.ne.jp/ec2017/EC2017compe.html

[2] 進化計算学会, 宇宙航空研究開発機構. 進化計算コンペティション 2018.
http://www.jpnsec.org/files/competition2018/EC-Symposium-2018-Competition.html

[3] 進化計算学会. 進化計算コンペティション 2019.
http://www.jpnsec.org/files/competition2019/EC-Symposium-2019-Competition.html

[4] 進化計算学会 実世界ベンチマーク問題分科会. 進化計算コンペティション 2020.
https://ec-comp.jpnsec.org/ja/competitions/eccomp2020

[5] Koen van der Blom, Timo M. Deist, Vanessa Volz, Mariapia Marchi, Yusuke Nojima, Boris Naujoks, Akira Oyama and Tea Tušar. Identifying Properties of Real-World Optimisation Problems through a Questionnaire. arXiv:2011.05547 [cs.NE], 2020.
https://arxiv.org/abs/2011.05547.

[6] Ilya Loshchilov and Tobias Glasmachers. BBComp: Black-Box Competition.
https://www.ini.rub.de/PEOPLE/glasmtbl/projects/bbcomp/

[7] AtCoder. AtCoder：競技プログラミングコンテストを開催する国内最大のサイト.
https://atcoder.jp

[8] AtCoder. 順位表 - AtCoder Beginner Contest 190.
https://atcoder.jp/contests/abc190/standings

[9] AtCoder. Introduction to Heuristics Contest
https://atcoder.jp/contests/intro-heuristics

[10] AtCoder. ヤマト運輸プログラミングコンテスト 2019.
https://atcoder.jp/contests/kuronekoyamato-contest2019

[11] AtCoder. Hokkaido Univ. & Hitachi 1st New-concept Computing Contest 2017.
https://atcoder.jp/contests/hokudai-hitachi2017-1

[12] AtCoder. Hokkaido Univ. & Hitachi 2nd New-concept Computing Contest 2017.
https://atcoder.jp/contests/hokudai-hitachi2017-2
[13] AtCoder. 北大・日立新概念コンピューティングコンテスト 2018.
https://atcoder.jp/contests/hokudai-hitachi2018
[14] AtCoder. Hitachi Hokudai Labo & Hokkaido University Contest 2019-1.
https://atcoder.jp/contests/hokudai-hitachi2019-1
[15] AtCoder. Hitachi Hokudai Labo & Hokkaido University} Contest 2019-2.
https://atcoder.jp/contests/hokudai-hitachi2019-2
[16] AtCoder. Hitachi Hokudai Lab. & Hokkaido University Contest 2020.
https://atcoder.jp/contests/hokudai-hitachi2020
[17] AtCoder. 2nd Asprova Programming Contest.
https://atcoder.jp/contests/asprocon2
[18] AtCoder. 第 3 回 Asprova プログラミングコンテスト.
https://atcoder.jp/contests/asprocon3
[19] AtCoder. 第 4 回 Asprova プログラミングコンテスト.
https://atcoder.jp/contests/asprocon4
[20] AtCoder. 第 5 回 Asprova プログラミングコンテスト.
https://atcoder.jp/contests/asprocon5
[21] AtCoder. DISCO presents ディスカバリーチャンネル コードコンテスト 2019 装置実装部門 本戦.
https://atcoder.jp/contests/ddcc2019-machine
[22] AtCoder Problems. AtCoder Problems API / Datasets.
https://github.com/kenkoooo/AtCoderProblems/blob/master/doc/api.md
[23] Kaggle. Kaggle Traveling Salesman Problem Competition.
https://www.kaggle.com
[24] Kaggle. Mercari Price Suggestion Challenge.
https://www.kaggle.com/c/mercari-price-suggestion-challenge
[25] Kaggle. Recruit Restaurant Visitor Forecasting.
https://www.kaggle.com/c/recruit-restaurant-visitor-forecasting
[26] Kaggle. Traveling Santa Problem.
https://www.kaggle.com/c/traveling-santa-problem
[27] Kaggle. Packing Santa's Sleigh.
https://www.kaggle.com/c/packing-santas-sleigh
[28] Kaggle. Helping Santa's Helpers.
https://www.kaggle.com/c/helping-santas-helpers
[29] Kaggle. Santa's Stolen Sleigh.
https://www.kaggle.com/c/santas-stolen-sleigh

[30] Kaggle. Santa's Uncertain Bags.
https://www.kaggle.com/c/santas-uncertain-bags
[31] Kaggle. Santa Gift Matching Challenge.
https://www.kaggle.com/c/santa-gift-matching
[32] Kaggle. Traveling Santa 2018 - Prime Paths.
https://www.kaggle.com/c/traveling-santa-2018-prime-paths
[33] Kaggle. Santa's Workshop Tour 2019.
https://www.kaggle.com/c/santa-workshop-tour-2019
[34] Kaggle. Santa 2019 - Revenge of the Accountants.
https://www.kaggle.com/c/santa-2019-revenge-of-the-accountants
[35] Kaggle. Santa 2020 - The Candy Cane Contest.
https://www.kaggle.com/c/santa-2020
[36] Kaggle. Hash Code Archive - Photo Slideshow Optimization.
https://www.kaggle.com/c/hashcode-photo-slideshow
[37] Kaggle. Hash Code 2021 - Traffic Signaling.
https://www.kaggle.com/c/hashcode-2021-oqr-extension
[38] 濱田 直希. 「実践と数理に根ざした多目的最適化ベンチマークの開発」成果報告書, 2019.
https://www.imi.kyushu-u.ac.jp/joint_research/detail/20190009

第 4 章

C#向け JSON デシリアライザ Hurisake.JsonDeserializer を作った

Takuya Hashimoto / @hasi_t

4.1 作ったもの

- repository: `https://github.com/hasipon/Hurisake.JsonDeserializer`
- NuGet: `https://www.nuget.org/packages/Hurisake.JsonDeserializer/`

Unity 2020.3.7 で動作確認しています。

Unlicense なので、自由に copy, modify, publish, use, compile, sell, or distribute しちゃってください。

使い方

`Hurisake.JsonDeserializer.Deseriaze(json)` でデシリアライズできます。 `json` は `string` または `byte[]` とします。

たとえば、 `[{"hoge":42}]` をデシリアライズして 42 を取り出すには `(int)a[0]["hoge"]` でできます。

やりたかったこと

Unity で使えて、スキーマ定義しなくても使える系の JSON デシリアライザを自作しようと思って書きました。

天下一 Game Battle Contest の API サーバーで StackExchange.Redis を使っていた

ときに RedisKey と RedisValue が気に入ったので、影響を受けた設計になっています。

経緯

2017 年の話ですが、MiniJSON[*1]を使っていて、サーバーに対する負荷試験を C#で書いていたら MiniJSON がボトルネックになった、ということがありました。@methane さんによると、 System.IO.StringReader の Read や Peek が大量の futex システムコールを発生させていたのが原因でした。mono の実装が原因だったのかもしれません（未調査)。

負荷試験は MiniJSON の修正や攻撃台数を増やすことで無事終わったのですが、JSON デシリアライズが遅いというのはなんとかしたいと思い、更なる高速化に取り組んだ結果、 string インスタンスを作らないようにするのが一番効くという結論に至りました。可能な限り `byte[]` のまま処理して、Dictionary インスタンスの生成を避けたり、JSON の object の key はハッシュ値にして比較したりといった高速化を実装し、@methane さんによる修正が入った MiniJSON 実装と比較して 4 倍くらい高速になることは確認しました。Hurisake.JsonDeserializer には、それなりに違う形にはなっていますが、このときの経験を反映させています。

余談

ICFPC 2017 で 2 位になったのは、リスト 4.1[*2]のような感じで JSON 処理を自前で書いたのも、タイムアウトせずに最終ラウンドまで残れた一因ではないかと妄想していたりします。ちょうど上記の高速化をやってた時期だったので、調子に乗って実装していた記憶があります。

▼リスト 4.1　ICFPC 2017 での提出コード（一部）

```
struct io_Json {
    char* buf;
    int64_t p, n;
    int64_t peek() {
        if (p >= n) return -1;
        return buf[p];
    }
    void read_ws() {
        for (;;) {
            auto c = peek();
            if (!is_ws(c)) break;
            ++p;
        }
    }
    bool is_ws(int64_t c) {
        return c == ' ' || c == '\t' || c == '\r' || c == '\n';
    }
};
```

[*1] https://gist.github.com/darktable/1411710

[*2] https://github.com/hasipon/icfpc2017/blob/master/AI3/hasi8.99/main.cpp

4.2 性能比較

neuecc/Utf8Json[*3] の `Utf8Json.JsonSerializer.Deserialize<dynamic>` と雑に比較しました。Deserialize の時間比較を 図 4.1 に、要素参照の時間比較を 図 4.2 に示します。単位はミリ秒、試行回数はそれぞれ 15 回です。計測プログラム（リスト 4.2)[*4] は 2 つのキーをもつ object 30 万要素の array をデシリアライズし、一方の値を合計するものです。

同じぐらいの時間でデシリアライズでき、 Hurisake.JsonValue は dynamic より高速な要素参照を実現できています。

計測環境

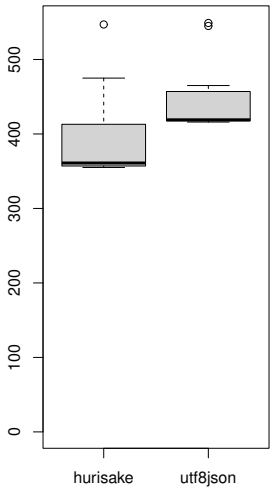
- Amazon EC2 t3.micro
- Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-059b6d3840b03d6dd (64-bit x86)
- dotnet 5.0.201

▼リスト 4.2　計測プログラム

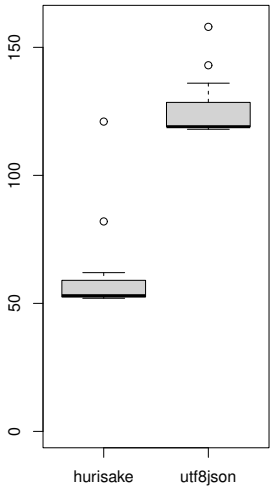
```
var data = new List<object>();
const int len = 300000;
long sum = 0;
var rnd = new System.Random();
for (int i = 0; i < len; ++i)
{
    var x = rnd.Next();
    sum += x;
    data.Add(new {hoge = x, piyo = "foo"});
}
var json = System.Text.Json.JsonSerializer.SerializeToUtf8Bytes(data);

var sw3 = new System.Diagnostics.Stopwatch();
var sw4 = new System.Diagnostics.Stopwatch();
sw3.Start();
var a = Hurisake.JsonDeserializer.Deserialize(json)!.Value;
// var a = Utf8Json.JsonSerializer.Deserialize<dynamic>(json);
sw3.Stop();
sw4.Start();
long sum2 = 0;
if (a.Count != len) throw new Exception();
for (int i = 0; i < a.Count; ++i)
{
    sum2 += (int)a[i]["hoge"];
}

if (sum != sum2) throw new Exception();
sw4.Stop();
Console.WriteLine(sw3.ElapsedMilliseconds);
Console.WriteLine(sw4.ElapsedMilliseconds);
```

[*3] https://github.com/neuecc/Utf8Json

[*4] https://gist.github.com/hasipon/071d1c92618aea26788b37746a0d21f8



▲図 4.1　Deserialize の時間比較



▲図 4.2　要素参照の時間比較

4.3 こぼれ話

Unity 対応

最初 Unity のことを考慮せずに書いてしまっていて、System.Collections.Immutable や ReadOnlySpan が Unity では使えなくて困りました。これらを使う方法もあるようですが、 Hurisake.JsonDeserializer は無くても動くように修正してあります。

Hurisake って何？

「天の原**ふりさけ**見れば春日なるみかさの山に出でし月かも――阿倍仲麻呂」から取ってます。小倉百人一首から「でし」を含む歌を選びました。深い意味は無いですが、検索しやすいようにする意図はあります。後付けですが、転勤して帰郷した今年の自分に合っている和歌な気もします。

4.4 おわりに

指摘・意見歓迎です。もしありましたら `https://github.com/hasipon/Hurisake.JsonDeserializer/issues` にお願いします。

第 5 章

オンライン対戦を支える独自シリアライズフォーマット

Daisuke Makiuchi / @makki_d

5.1 はじめに

近年のモバイルオンラインゲームでは、対戦や協力プレイといった同期通信が当たり前になっています。KLab でももちろんそのようなゲームをリリースしており、Photon[*1]のようなサードパーティのサービスを使うこともありますが、いくつかのタイトルでは独自の同期通信の仕組みを使っています。筆者はこの数年、この同期通信基盤の開発と運用に携わってきました。

KLab の多くのタイトルは Unity で制作していますが、この同期通信基盤では部屋管理とデータ中継のサーバーを Go 言語で実装しており、各クライアントから HTTP と WebSocket でこれらのサーバーに接続する構成を取っています。また、さまざまなプロジェクトで同じサーバーをそのまま使えるような汎用的な作りにしています。

この章では、KLab の同期通信基盤のために開発した独自のシリアライズフォーマットについて、その特徴や工夫した点を紹介したいと思います。

5.2 なぜ独自フォーマットが必要だったのか

ネットワークを介してデータを送信するには、何らかの方法でビット列に変換する必要があります。そして受信したビット列は、元のデータに復元しなければプログラムからは利用できません。クライアントは C#なので、値だけでなく型も送受信の前後で同じにならないと困ってしまいます。

C#同士だけであれば、C#の標準ライブラリの `System.Runtime.Serialization`を

[*1] `https://www.photonengine.com/`

使うこともできるかもしれません。しかし今回作っているものは、部屋を管理するためにGo 製のサーバーでもデータを読み取る必要があるため、Go でも読み取りやすい形式が必要でした。

世の中には C#でも Go でも、あるいは他の言語でも使えるような汎用フォーマットもあります。しかし、たとえば JSON や MessagePack[*2]では C#よりも型が少ないため送信元の型を完全に復元することができませんし、C#のときの型が Go からは分らなくなってしまいます。あるいは ProtocolBuffers[*3]のように共通の定義から事前にコード生成するものもありますが、利用するデータ型を追加するためにはクライアントだけでなくサーバーも合わせて更新する必要があります。これでは多くのプロジェクトで使える共通のサーバーを作るには不向きです。できるならクライアントだけで独自のデータ型を追加できるのが理想です。

このようなニッチな要求を満たすものはまず存在しないので作ることにしました。ここで紹介するシリアライザは GitHub にて公開しています[*4]。あわせてご覧ください[*5]。

5.3 独自フォーマットの特徴

C#のプリミティブ型に加え、独自定義型も特定インターフェイスを実装することでシリアライズできます。加えて、シリアライズ可能な型を要素にもつリストや配列、文字列キーの辞書型もサポートし、ネストもできます。

この辞書型は部屋のプロパティとしても利用しており、Go でも辞書（`map`）として扱うためにキーの型を固定する必要がありました。Photon でも部屋のプロパティ（`RoomInfo.CustomProperties`）は文字列キーの辞書を採用していますし、扱いやすさを優先して文字列キーとしました。

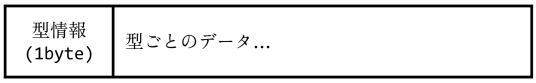
また Go でリストや辞書をデシリアライズするとき、各要素をバイト列のスライス（`[]byte`）のまま保持し、必要になるまでデシリアライズしない遅延デシリアライズを実現したほか、同じ型同士の単純な大小比較であればバイト列のまま比較できるようにしました。このメリットは後ほど紹介したいと思います。

クライアント側 C#の実装も、パフォーマンス対策として boxing の回避やオブジェクトの再利用ができるよう実装しています。その詳細はリポジトリの `SerialReader`、`SerialWriter`クラスをご覧いただくとして、ここではフォーマットの概要を説明します。

[*2] `https://msgpack.org/`
[*3] `https://developers.google.com/protocol-buffers`
[*4] `https://github.com/KLab/wsnet2-serializer`
[*5] 紙面に掲載したコード片は一部簡略化などの変更をしています

5.4 フォーマットの概要

基本的には、1byte の型情報とそれに続く型ごとのデータのバイト列で構成されます。扱える型は先述のとおり、C#のほとんどのプリミティブ型と、独自定義型、シリアライズ可能な型のリストや辞書です。



▲図 5.1　基本的な形

▼表 5.1　型情報と対応する C#の型一覧

0: Null	17: Obj (独自定義クラス)
1: False (`bool`)	18: List (`List<object>`)
2: True (`bool`)	19: Dict (`Dictionary<string, object>`)
3: SByte (`sbyte`)	20: Bools (`bool[]`)
4: Byte (`byte`)	21: SBytes (`sbyte[]`)
5: Char (`char`)	22: Bytes (`byte[]`)
6: Short (`short`)	23: Chars (`char[]`)
7: UShort (`ushort`)	24: Shorts (`short[]`)
8: Int (`int`)	25: UShorts (`ushort[]`)
9: UInt (`uint`)	26: Ints (`int[]`)
10: Long (`long`)	27: UInts (`uint[]`)
11: ULong (`ulong`)	28: Longs (`long[]`)
12: Float (`float`)	29: ULongs (`ulong[]`)
13: Double (`double`)	30: Floats (`float[]`)
14: Decimal (`decimal`)[*6]	31: Doubles (`double[]`)
15: Str8 (`string`)	32: Decimals (`decimal[]`)[*7]
16: Str16 (`string`)	

ここからはそれぞれの型について、その型ごとのシリアライズ方法を解説していきます。

Null 値と bool 型

さきほど、フォーマットの基本は 1byte の型とそれに続くデータと説明しましたが、いきなり例外的なものたちです。`bool`型は `true`または `false`ですが、それを型情報に加えて 1byte のデータで表すのはもったいないので、型情報を True と False の 2 種類に分

[*6] `decimal`型は定義していますが未実装です

け、データを持たない形としました。また、Null 値は型をもたない nullとして、これも 1byte の型情報のみで表現します。このようなやり方は MessagePack を参考にしました。

リスト 5.1 に bool 型のシリアライズとデシリアライズの C#実装を掲載します。boxing を避けるために、書き込み（Writeメソッド）は引数の型でメソッドオーバーロードし、読み出しは型ごとのメソッド（Read+型名）を定義しています。

▼リスト 5.1　bool 型のシリアライズ・デシリアライズ

```
public class SerialWriter
{
    int    pos; // 書き込み位置
    byte[] buf; // 書き込みバッファ
    (略)
    /// <summary>Bool 値を書き込む</summary>
    public void Write(bool v)
    {
        expand(1);                                      // バッファが足りなければ 1byte 拡張
        buf[pos] = (byte)(v ? Type.True : Type.False); // 型情報として True か False を書き込む
        pos++;
    }
    (略)
}

public class SerialReader
{
    (略)
    /// <summary>Bool 値として読み出す</summary>
    public bool ReadBool()
    {
        var t = checkType(Type.True, Type.False);
        return t == Type.True;
    }
    (略)
    /// <summary>先頭 1byte を読み、引数の Type だったらそれを返し、それ以外のときは例外送出</summary>
    Type checkType(Type want1, Type want2)
    {
        checkLength(1);               // 1byte 以上あるか確認
        var t = (Type)buf[pos];
        if (t != want1 && t != want2)
        {
            throw new SerializationException("invalid type");
        }
        pos++;
        return t;
    }
}
```

整数型、浮動小数点数型

整数型は 1byte の型情報に続けて、数値を BigEndian で書き込みます。MessagePack のように節約したフォーマットではなく、64bit の long型はそのまま 8byte で記録する単純な形です。このとき、符号なし整数はそのままですが、符号付き整数は下駄履き表現、つまり shortの場合 128を加えて、-128を 0、127を 255 となるようにして書き込みます。こうすることで、バイト列を先頭から単純に比較していくだけで、元の値の大小関係が分かるようになります。

▼リスト 5.2　符号付き short 型のシリアライズ・デシリアライズ

```
public class SerialWriter
{
    (略)
    /// <summary>Short 値を書き込む</summary>
    public void Write(short v)
    {
        expand(3);
        buf[pos] = (byte)Type.Short;
        pos++;
        var n = (int)v - (int)short.MinValue; // 下駄履き表現に変換
        Put16(n);
    }
    (略)
    /// <summary>16bit 値を BigEndian で書き込む</summary>
    public void Put16(int v)
    {
        buf[pos] = (byte)((v & 0xff00) >> 8);
        buf[pos+1] = (byte)(v & 0xff);
        pos += 2;
    }
    (略)
}

public class SerialReader
{
    (略)
    /// <summary>Short 値として読み出す</summary>
    public short ReadShort()
    {
        checkType(Type.Short);
        return (short)(Get16() + (int)short.MinValue); // 下駄履き表現から戻す
    }
    (略)
    /// <summary>16bit 値を読み出す</summary>
    public int Get16()
    {
        checkLength(2);
        var n = (int)buf[pos] << 8;
        n += (int)buf[pos+1];
        pos += 2;
        return n;
    }
    (略)
}
```

浮動小数点数でも IEEE 754 の表現からビット操作して、整数型と同じようにバイト列のまま大小比較できるようにしました。詳しくは筆者の blog 記事[*8]をご覧ください。

文字列型

文字列型は可変長なので、1byte の型情報に続けてデータ長を書いておきます。ゲームで通信しあう文字列は短いものが多いので、データ長は 1byte で表現したいところですが、チャットのような機能を作る場合は 255 文字では足りないかもしれません。そこで型情報を Str8 と Str16 の 2 つに分け、255byte 以下は前者で文字列長を 1byte、それ以上長いものは後者で文字列長を 2byte としました。

データのエンコーディングは UTF-8 とします。これは Go の文字列の内部エンコー

[*8] http://makiuchi-d.github.io/2020/12/09/float-comparable.ja.html

ディングが UTF-8 なので、バイト列からそのまま文字列にキャストできるようにするためです。

▼リスト 5.3　文字列型のシリアライズ・デシリアライズ

```
public class SerialWriter
{
    (略)
    /// <summary>文字列を書き込む</summary>
    public void Write(string v)
    {
        if (v == null)
        {
            Write(); // Type.Null 書き込み
            return;
        }

        var len = utf8.GetByteCount(v); // UTF-8 でのデータ長
        if (len <= byte.MaxValue)
        {
            expand(len+2);
            buf[pos] = (byte)Type.Str8;
            pos++;
            Put8(len); // 1byte でデータ長を記録
        }
        else if (len <= ushort.MaxValue)
        {
            expand(len+3);
            buf[pos] = (byte)Type.Str16;
            pos++;
            Put16(len); // 2byte でデータ長を記録
        }
        else
        {
            throw new SerializationException("too long");
        }

        utf8.GetBytes(v, 0, v.Length, buf, pos); // UTF-8 として書き込み
        pos += len;
    }
    (略)
}

public class SerialReader
{
    (略)
    /// <summary>文字列として読み出す</summary>
    public string ReadString()
    {
        var t = checkType(Type.Str8, Type.Str16, Type.Null);
        if (t == Type.Null)
        {
            return null;
        }
        // データ長は Str8 なら 1byte, Str16 なら 2byte
        var len = (t == Type.Str8) ? Get8() : Get16();
        var str = utf8.GetString(buf, pos, len);
        pos += len;
        return str;
    }
    (略)
}
```

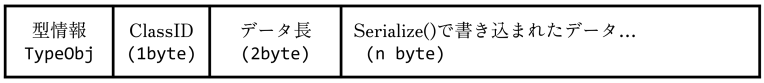
独自定義クラス

独自定義クラスをシリアライズできるようにするには、IWSNet2Serializableインターフェイスを実装し、WSNet2Serializer.Registerメソッドで ClassID を事前に登録する必要があります。この ClassID とクラスの対応関係は通信するすべてのクライアントで一致している必要があります。クライアントは基本的に同じソースからビルドするはずなので、一致させるのは容易でしょう。

▼リスト 5.4　IWSNet2Serializable インターフェイスと Register メソッド

```
public interface IWSNet2Serializable
{
    void Serialize(SerialWriter writer);
    void Deserialize(SerialReader reader, int size);
}

public class WSNet2Serializer
{
    public delegate object ReadFunc(SerialReader reader, object recycle);

    static Hashtable registeredTypes = new Hashtable(); // 型->ClassID のマッピング
    static ReadFunc[] readFuncs = new ReadFunc[256];
    (略)
    /// <summary>独自定義クラスを登録</summary>
    public static void Register<T>(byte classID) where T : class, IWSNet2Serializable, new()
    {
        var t = typeof(T);
        registeredTypes[t] = classID;

        // SerialReader.ReadObject<T>() は型 T がわからないと呼べない
        // ClassID だけから呼び出せるように無名関数を保持しておく
        readFuncs[classID] = (reader, obj) => reader.ReadObject<T>(obj as T);
    }
    (略)
}
```

シリアライズ後のデータは図 5.2 のような形になります。ClassID が 1byte なため登録できるクラスは 256 種類に限られますが、普通のゲームであれば十分な数です。また、ClassID の後にデータサイズがあることで、中身をデシリアライズすることなくデータを切り出すことができます。これにより、データ部分をバイト列として切り出しておき、必要になってからデシリアライズする遅延デシリアライズができます。



▲図 5.2　独自定義クラスのシリアライズイメージ

ここでリスト 5.5 に独自定義クラスの例として、チェスの駒を表すクラスを定義してみます。

▼リスト 5.5　独自定義クラスの例

```
public class ChessPiece
{
    public PieceType Type;
    public int PositionX;
    public int PositionY;

    public void Serialize(SerialWriter writer)
    {
        writer.Write((byte)Type); // PieceType は 1byte で
        // 盤面は 8x8 なので座標も 1byte にまとめて
        writer.Write((byte)(PositionX * 8 + PositionY));
    }

    public void Deserialize(SerialReader reader, int size)
    {
        Type = (PieceType)reader.ReadByte();
        var pos = writer.ReadByte();
        PositionX = pos / 8;
        PositionY = pos % 8;
    }
}
```

Serializeメソッドでは SerialWriter経由でデータを書き込んでいくため、書き込み毎に型情報が付加される形でシリアライズされます。若干冗長な気もしますが、このおかげで Go でも独自定義クラスの中にどのような型の値が含まれているか読み取ることができます。

またチェスの盤面は 8 × 8 なので、X と Y の座標を 1byte にまとめることで通信量を減らしています。ゲームで使うオブジェクトでは、このようなゲーム仕様に基づく最適化ができたり、マスタデータの ID など一部のメンバだけ送れば済むようなケースがよくあります。このため、リフレクションを使った自動的なシリアライズなどは行わず、若干面倒かもしれませんが自分で書く形としました。Serializeと Deserializeで対応がとれていないと正しく動かなくなってしまいますが、ユニットテストで担保するとよいでしょう。

▼リスト 5.6　独自定義クラスのシリアライズ・デシリアライズ

```
public class SerialWriter
{
    (略)
    /// <summary>独自定義クラスのオブジェクトを書き込む</summary>
    public void Write<T>(T v) where T : class, IWSNet2Serializable
    {
        if (v == null)
        {
            Write(); // null のときは型なし Null を書き込むだけ
            return;
        }

        var t = v.GetType();
        var id = types[t];

        expand(4);
```

```
        buf[pos] = (byte)Type.Obj;
        buf[pos+1] = (byte)id;      // ClassID 書き込み
        pos += 4;                   // サイズを書き込む領域分進める
        var start = pos;

        v.Serialize(this); // 独自定義クラスの Serialize() 呼び出し

        // Serialize で書き込んだサイズを埋める
        var size = pos - start;
        buf[start-2] = (byte)((size & 0xff00) >> 8);
        buf[start-1] = (byte)(size & 0xff);
    }
    (略)
}

public class SerialReader
{
    (略)
    /// <summary>独自定義クラスとして読み出す</summary>
    public T ReadObject<T>(T recycle = default) where T : class, IWSNet2Serializable, new()
    {
        if (checkType(Type.Obj, Type.Null) == Type.Null)
        {
            return null;
        }

        var cid = classIDs[typeof(T)];
        var id = (byte)Get8();
        if (id != (byte)cid)
        {
            throw new SerializationException("class id mismatch");
        }

        var size = Get16();
        checkLength(size);

        var obj = recycle;
        if (obj == null) {
            obj = new T();
        }

        var start = pos;
        obj.Deserialize(this, size);
        pos = start + size;

        return obj;
    }
    (略)
}
```

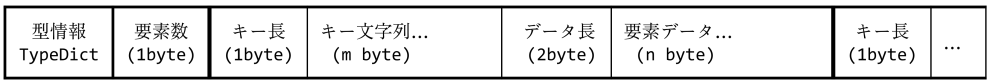
デシリアライズする ReadObject<T>メソッドでは、recycleというオブジェクトを引数として受け取ります。データ受信時にオブジェクトを新たに作るのではなく再利用することで、メモリアロケーションを減らすことができます。

リストと辞書

リスト型は型情報に続いて 1byte の要素数、その後にシリアライズした要素のデータ長とデータがが繰り返し配置される形でシリアライズされます。それぞれの要素データは、型情報とその型ごとのデータのバイト列からなる、シリアライズされたデータです。このような構造のため、何重にもネストすることができます。

▲図 5.3　リスト型のシリアライズイメージ

辞書型もリスト型と同じように、1byte の要素数と要素の繰り返しからなる形です。各要素は 1byte のキー長、キー文字列データ、シリアライズした要素のデータ長とデータが並びます。



▲図 5.4　辞書型のシリアライズイメージ

リストも辞書も、各要素データの長さがデータの前にあることで、要素の中身をデシリアライズすることなくバイト列として切り出すことができます。特に辞書型は部屋のプロパティとしても使われていて、データを切り出せることが Go での扱いやすさに繋がっています。これについては後で解説します。

プリミティブ型の配列

`int[]`のようなプリミティブ型の配列は、`int`がシリアライズ可能なのでリスト型としてもシリアライズできます。しかし、リスト型では要素ごとにサイズや型情報が入り効率がよくありません。なので、数値型や bool 型の配列は専用の型としてシリアライズできるようにしました。

型によって要素データのサイズは固定なので、リストのようにデータサイズを書き込まず、単純に値だけをシリアライズしたデータを並べます。さらに bool 型の配列は 1 ビット単位で効率的に格納します。

▲図 5.5　int 型配列のシリアライズイメージ

5.5　サーバーでの部屋のプロパティ

各部屋には、クライアントが自由に設定できるプロパティとして、文字列キーの辞書を用意しています。このプロパティは部屋にいる全クライアントに共有される他、部屋の検索やランダム入室の際のフィルタリングにも利用します。

この辞書は、Go のサーバーでは map[string][]byte型になっていて、辞書の各要素の値はバイト列（[]byte）のままデシリアライズせずに保持しています。

プロパティの値の変更

プロパティの値を変更するときは、リスト 5.7 のように変更するキーと値だけの辞書をクライアントから送ります。

▼リスト 5.7　クライアントから送る辞書型データ

```
var dict = new Dictionary<string, object>()
{
    {"Turn", 1},
    {"WhitePawn4", new ChessPiece(){ Type=PieceType.Porn, PositionX=3, PositionY=4 }},
};

room.ChangeRoomProperty(publicProps=dict);
```

Go の中継サーバーはこの辞書をリスト 5.8 のようにデシリアライズし、文字列キーと値データのバイト列を取り出します。

▼リスト 5.8　Go での辞書のデシリアライズ

```
type Dict map[string][]byte

func unmarshalDict(src []byte) (Dict, int, error) {
    count := get8(src[1:]) // 要素数
    l := 2
    dict := make(Dict)
    for i := 0; i < count; i++ {
        lk := get8(src[l:])     // キー長
        l += 1
        key := src[l : l+lk]    // キーデータ
        l += lk
        lv := get16(src[l:])    // データ長
        l += 2
        dict[string(key)] = src[l : l+lv] // データのバイト列
        l += lv
    }
    return dict, l, nil
}
```

このようにサーバー側での辞書のデシリアライズでは、各要素の値のデータをバイト列のスライスとして切り出しています。Go のスライスは元の配列の参照になっているため、メモリのコピーが発生せず高速です。そしてこのスライスをそのまま部屋のプロパティとして保持します。

5.6 部屋のフィルタリング

部屋検索では部屋のプロパティを参照する柔軟なフィルタリングができるようにしました。フィルタリングの条件は、たとえば「レベル 10~15 の範囲かつ、赤チームまたは黄色チーム」はリスト 5.9 のような形[*9]で指定します。

▼リスト 5.9　フィルタリング条件のイメージ

```
[
    [
        {"Team", Equal, (byte)Team.Red},
        {"Level", GreaterOrEqual, 10},
        {"Level", LessOrEqual, 15}
    ],
    [
        {"Team", Equal, (byte)Team.Yellow},
        {"Level", GreaterOrEqual, 10},
        {"Level", LessOrEqual, 15}
    ]
]
```

このように条件を｛キー、演算子、値｝の二重配列として表し、内側の配列は AND 結合、外側は OR 結合にしています。どんなに複雑な条件指定も、分配法則やド・モルガンの法則で変換すれば必ずこの形に変形できます。

この形にしておくと、サーバー側はリスト 5.10 のようにシンプルなループでフィルタリングできます。

▼リスト 5.10　フィルター

```
type PropQuery struct {
     Key string
     Op  OpType // operation type (==, !=, <, <=, >, >=)
     Val []byte // value
}

func filter(rooms []*Room, queries [][]PropQuery) []*Room {
    filtered := make([]*Room, 0)

    for _, room := rooms {

        // OR 結合：一つでもマッチしている条件群があれば Room を追加
        for _, qs := queries {

            // AND 結合：全てマッチしていたらこの条件群はマッチ
            match := true
            for j := range queries[i] {
                if !queries[i][j].match(room.Property[queries[i][j].Key]) {
                    match = false
                    break
                }
            }

            // マッチしていたので結果に追加
            if match {
```

[*9] 実際には、HTTP の Body に MessagePack 形式で他のパラメータとともにこのようなフィルタ条件を入れて送っています

```
                filtered = append(filtered, room)
                break
            }
        }
    }
    return filtered
}
```

値の比較はこれまで説明してきたとおり、バイト列をそのままバイト単位で比較します。数値型の場合、型が一致していれば大小関係もバイト列のまま比較できます。このようなサーバー側の実装だけでは、数値以外でも大小関係の比較を指定できてしまう形式なのですが、そこはクライアント側のフィルタ条件生成クラスで大小比較は数値型だけになるように担保しています。

▼リスト 5.11　マッチするか判定

```
func (q *PropQuery) match(val []byte) bool {
    ret := bytes.Compare(val, q.Val) // バイト列のまま比較
    switch q.Op {
    case OpEqual:
        return ret == 0
    case OpNot:
        return ret != 0
    case OpLessThan:
        return ret < 0
    case OpLessOrEqual:
        return ret <= 0
    case OpGreaterThan:
        return ret > 0
    case OpGreaterOrEqual:
        return ret >= 0
}
```

5.7　さいごに

この章では、KLab の同期通信基盤の独自のシリアライズフォーマットを紹介しました。汎用性を犠牲にして自分たちの用途に合わせているため、そのまま使える場面は少ないと思いますが、細かい工夫点やテクニックが何かの参考になれば幸いです。

また、シリアライザだけでなくこの同期通信基盤そのものについても、今後何らかの発表をしていきたいと思っておりますのでご期待下さい。

第 6 章

モデム通信しかサポートしていない昔のネット対戦ゲームをブロードバンド・光回線で行うには

Tomoaki Fude

これは、1998~2003 年頃に発売されたモデム通信でしかプレイできない（PPP を使用しないピアツーピアな）完全同期型ネット対戦ゲームを、2021 年現在主流となっているブロードバンド回線・光回線等でもネット対戦は技術的に可能であることを主張する記事です。また、そこに至る試行錯誤を記録したものです。

6.1 はじめに

1998 年~2006 年頃のお話です。その時期に発売された家庭用ゲーム機であるドリームキャストや PlayStation 2（以下 PS2）、ゲームキューブではネット対戦が可能なゲームソフトが発売されていました。しかし、PS2 を例に挙げると 2002 年 1 月くらいまでは LAN ケーブルではネット接続ができませんでした。電話回線とモジュラーケーブルとモデムを使ってネット接続をしていたのです。

とくに、2021 年現在で 20 代までの方は、モデムという言葉に馴染みがない方がほとんどではないでしょうか。20 年以上前の、ネットに接続するには電話回線が主流だったときのお話ですが、アナログな電話回線を使ってデジタルなデータのやり取りをするのにモデムというものを使っていました。

その後ブロードバンドが普及しはじめ、2002 年 1 月頃に通販やレンタルでブロードバンド対応のアダプタ（BBUnit）の利用が可能となります。つまり 2002 年頃ようやく PS2 で LAN ケーブルが利用できるようになり、それ以降でブロードバンド対応のネット対戦ゲームが続々と発売されたのです。

ブロードバンド対応のネット対戦ゲームは LAN ケーブルを使うため、対戦に必要な

サーバーのサービスが終了していない限り、2021 年現在もネット対戦が可能です。一方、1998 年~2003 年頃に発売された「モデム通信にのみ対応したネット対戦ゲーム」は電話回線を用いるか、PPP を用いるゲームソフトの場合は後述する RAS サーバーを用いなければ 2021 年現在もネット対戦を行うことはできないはずです。（さらに、インターネットに接続する必要があるモデム通信ゲームの場合で RAS サーバーを使わない場合は、プロバイダが提供するアクセスポイント（電話番号）にモデムから電話をかけて PPP で IP アドレスを取得する必要があります。アクセスポイントは 2021 年現在もまだ存在します。通話料がかかりますのでご使用の際はご注意ください。）

しかも、今のご時世では固定電話回線を持っていないお家も多いため、そういったご家庭では昔のモデムによる（PPP を使用しない）ネット対戦ゲームはプレイできません。... と思っていたのですが、電話回線を持っていなくてもモデム対戦を実現する方法がありました。本記事では、そんな昔ながらのモデム通信で PPP を使用しない、ピアツーピアなネット対戦ゲームを現在主流のインターネット回線に載せる手法について述べます。なお、これから述べる手法のうち前半はある一人の先駆者による発見・検証過程をヒアリングして記載しました。執筆者は先駆者と情報交換を行い、後半の手法である「PC に USB モデムを挿して通信する手法」についての検証と実装を担当しました。

モデムという単語について

本記事での「モデム」は主に、デジタルなデータを扱う PC・ゲーム機からアナログ回線にデータを載せるためにデジタル信号をアナログ信号に変換（またはその逆の変換）をする機器という意味で利用します。つまり PC に接続して利用する USB モデムや、PS2 やドリームキャスト、ゲームキューブに接続するモデムのことを指します。

CATV の回線を利用されている方は同軸ケーブルから LAN ケーブルへ信号変換するケーブルモデム（CATV モデム）を連想してしまうかと思いますが、本記事ではその機器のことではありません。また、集合住宅等でインターネットを利用するために既存のアナログ電話回線を流用して各部屋にインターネット接続を提供する VDSL で用いられる VDSL モデムのことでもありません。

動作確認に使用したゲームソフト

動作確認で使用したゲームソフトは 2001 年に株式会社フロム・ソフトウェアより発売された PS2 用ソフトのアーマード・コア 2 アナザーエイジです（以降 AC2AA と表記します）。

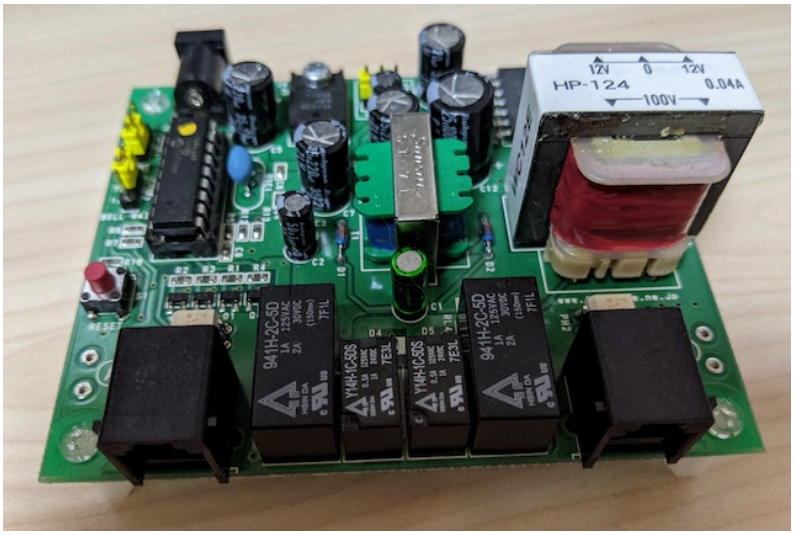
こちらのゲームはストーリーモードとは別にモデム対戦モードがあます。通信方法としては、アナログ電話回線を用いて対戦したいお友達の電話番号に PS2 のゲーム内から電話をかけて、お友達がゲーム内で電話をとるとモデム通信の接続が確立されてネット対戦ができるといったものです。

ネット対戦と書きましたが、お友達同士の電話回線をつなげているだけなので、インターネットは使いません。プロバイダ契約も不要です。ただ、回線としては通常の電話をしているのと変わらないので、テレホーダイ等の定額通話プランを使わない限り通話料が従量課金で発生します。また、このネット対戦をしている際は電話回線を占有してしまうため、電話が使えません。外から電話がかかってきても話し中になってしまうと思います。（※テレホーダイは 23 時から翌日 8 時まで指定した電話番号への通話料が定額となる 2021 年現在も現役のサービスです。）

6.2　アナログなデータをどうやって相互に通信するか

PS2 などのゲーム機からモデムを通して送られるデータはアナログな信号（音声）です。特に、サーバーを用いないピアツーピア接続の対戦ゲームであれば、モデムを接続したゲーム機を 2 台用意してあげれば、そのゲーム機は内線電話とみなしてよいはずです。言い換えれば、通常のアナログ電話を 1 対 1 で通話できる機材（交換機）を流用すればゲーム機でも通信対戦ができるはずです。これは実際のアナログ電話の交換機を使えば実現できそうです。

アナログ電話が主流なオフィスにある内線電話の PBX があれば、内線電話番号に応じた場所にモデムを接続したゲーム機を配置すれば内線電話番号で通信対戦ができるはずですが、一般の家庭には PBX なんて大層なものはありません。そこで一般家庭でも手が届く金額の範囲で、通話相手は固定ですが内線電話をかけられる商品が存在します。



▲図 6.1　トライステート社の擬似交換機

トライステート社の擬似交換機です。（図 6.1） モジュラージャックの口が 2 つついて

おり、アナログ電話を 2 つ挿せば、その電話同士が内線として使えます（図 6.2）（※ NTT の電話線などには決して接続してはいけません．電気通信事業法第 52 条第 2 項の規定に基づく）

▲図 6.2　擬似交換機を用いた内線電話

これを用いてモデムを接続した PS2 同士で通信対戦が可能なことが先駆者によって判明しました。しかし、このままでは通常の電話回線を使うのと何ら変わりありません。内線のため遠くのお友達と対戦もできませんし、問題解決になっていません。

6.3　アナログなデータをどうやって LAN ケーブルに載せるか

擬似交換機内で中継されるアナログなデータ（=音声）を LAN ケーブルで中継できるものがあればこの点はクリアできそうです。何かよいソリューションがあればよいのですが...。これも先駆者の発案・検証作業で VoIP を用いればよいことが判明しました。

VoIP をざっくり説明すると、人の声などの音声を時分割して、時間ごとに区切られた音声を符号化・音声圧縮しパケットにして IP 網を経由し、通話相手側でパケットから音声を取り出す技術です。ただ、通話品質はジッタ・パケロス等の影響を大きく受けてしまいます。回線の品質が悪い状況だと相手の話し声が途切れ途切れになってしまうでしょう。

モデムのデータ通信を VoIP に載せるとなると、パケロス等の影響でデータが欠けてしまうことになるかと思います。ただ、当時は他によい方法が思いつかなかったため、VoIP

でモデム対戦を実現する方向で検証が進められました。検証に使用した VoIP ルータは 2006 年 9 月にヤマハ株式会社より発売された RT58i です（図 6.3）。他にも日立の VoIP ゲートウェイや、一般には馴染みのない企業が官公庁や鉄道などのインフラ用に向けて制作したものや、Cisco のものなど多数の候補がありました。これらについては価格の調査などを行っており、RT58i での接続性が悪い場合の候補として残してありました。

▲図 6.3　VoIP ルータを用いた内線電話

この RT58i という製品にはカスケード接続機能というものがあります。RT58i の筐体にはモジュラージャックが 2 つしかなく、通常は 2 台までのアナログ電話しか接続することはできません。そこで、同じ RT58i をもうひとつ用意して、2 台の RT58i を LAN ケーブルで接続することで、1 台では 2 つの内線電話までしか接続できなかったものが、4 つまでのアナログ電話を管理することができるようになります。これがカスケード接続機能です。2 台の RT58i を接続する際に、どちらの RT58i を親機とするか、子機とするかの選択が必要となります。通常はブロードバンド回線や ISDN 回線を接続している方の RT58i を親機にしますが、今回はインターネットには接続せずにローカルで用いるため、どちらを選んでも構いません。LAN ケーブルで RT58i 同士を接続すると、リスト 6.1 のような構成をとることが可能となります。

▼リスト 6.1　RT58i のカスケード接続における配線イメージ図

```
[アナログ電話]----[RT58i(親機)]
                      |
                      |
                (LAN ケーブル)
                      |
                      |
[アナログ電話]----[RT58i(子機)]
```

カスケード接続をする際は、2 台の RT58i の間を LAN ケーブルで接続する、ということになっていましたが、これを VPN の LAN 内に配置することで、インターネット越しに RT58i のカスケード接続を実現します。つまり、VPN 内で VoIP を用いた内線通話環境ができ上がる訳です（リスト 6.2)。VPN には SoftEtherVPN を用いました。

▼リスト 6.2　インターネット越しで RT58i のカスケード接続を VPN で実現

```
[PS2 モデム]----[RT58i(親機)]----[PC(SoftEtherVPN)]
                                        |
                                        |
                                  (Internet)
                                        |
                                        |
[PS2 モデム]----[RT58i(子機)]----[PC(SoftEtherVPN)]
```

インターネット越しの内線電話が構築できたため、アナログ電話を配置していたところを PS2 モデムに置き換えます。これで PS2 モデムでの（ピアツーピアでの対戦ゲームでは）対戦が可能となります。

初めは通信エラーが頻発したそうですが、圧縮方式など何百とおりもの VoIP 設定（VoIP コーデックを変えてみる等）の組み合わせを試し、最終的にはかなり安定してネット対戦する状況を作りあげていました。2020 年 11 月頃において調整された VoIP 設定では、モデム対戦を開始してから 15 分程度でゲーム内の通信エラーが発生し始める問題があるようでした。再戦などを選択するメニュー画面を挟まずに、通しで対戦時間無制限をプレイした場合はロード時間も含めて 20 分程度でエラーが出始めます。また、試合が終わった後のメニュー画面では再戦やリプレイなどの選択ができるのですが、再戦選択時のロード画面やリプレイ再生中にエラーが発生しやすいです。なお、この 2020 年 11 月頃の設定では対戦可能な方との ping 値は 20ms 程度までが限界でした。

その後、2021 年 3 月頃以降では新たな基準で VoIP 設定の調整がなされます。キー入力のレスポンスと、ゲーム内の通信の安定性は同時に最良の状態を維持することは難しく、キー入力のレスポンスを良くすれば通信の安定性が悪くなってしまい、反対にゲーム内通信の安定性を確保していくとかなり安定度を増すことができますが、今度はキー入力のレスポンスが悪くなってしまいます。そこで、1 試合が終了する程度の間はゲーム内で通信エラーを発生させないように通信の安定性を確保し、できるだけキー入力のレスポンスを良くするといった基準で VoIP 設定が調整されました。

6.4 よりよい通信品質の回線を求めて

（※前節までは先駆者による検証の記録（対戦方法の発見・検証・VoIP 設定の基準についてをヒアリングして執筆者が記載）であり、本節以降から、とくに明記されていない限り、執筆者が担当した検証・試行錯誤についての記載となります。）

2020 年 11 月当時、VoIP 方式ではある程度まではできたものの、数分間プレイしてからは通信エラーの発生は避けられず、おそらくですがパケロスやジッタなどの影響をどうしても抑制できない部分がありました。後からわかったことですが、VoIP を使って電話以外のデータをやり取りする技術を FoIP（Fax over IP）というそうです。FoIP という FAX みなし音声通信には問題がいくつかあり、これら問題を解決するのは容易なことではありません。[*1] 新たな別の方式を模索する必要があると執筆者は感じました。

そこで、先駆者と昔のモデム通信に関する情報を漁った結果、RAS サーバーという方法があることが分かりました。[*2] RAS サーバーとは、今回のケースで使われている用語としての説明になりますが、次のとおりです。これは、ブロードバンドの普及しはじめにおいて、定額制ブロードバンドを契約しているにもかかわらず、ゲーム機本体がモデムしか備えておらず（従量課金性の）ダイヤルアップ接続しか対応していないため、どうにかしてモデム通信を定額制であるブロードバンド経由でインターネットに接続する方法がないか、といったときに編み出されたテクニックなようです。方法としては、モデムしか備えていないゲーム機を、モデムを備えた（ブロードバンドに接続した）PC に PPP でダイヤルアップ接続して IP アドレスを取得し、その PC 経由で定額制ブロードバンドでインターネットに接続するといったものです。

しかし、動作確認で利用していた AC2AA では PPP の機能はありません。ただ、PC に USB モデムを接続して PS2 モデムと対話するといったアイデアは使えるのではないかという考えはありました。この時点（2020 年 11 月末）で社内でよい案がないか、社内勉強会である ALM で募集したところ、興味を持ってくれた方から「モデムでアナログ回線を終端するという発想は有効に思えます。モデムの通信は PC からはシリアル通信として見えるので、それを相手に転送できればいいはず。」というアドバイスをいただきました。（他にもたくさんの素晴らしいアイデアをいただきました）なんとか PS2 モデムと、PC に接続した USB モデム間でデータのやり取りができればよいのですが...。

[*1] 画像電子学会誌, 36 巻（2007）1 号, VoIP 網における FAX みなし音声通信：
https://www.jstage.jst.go.jp/article/iieej/36/1/36_1_17/_article/-char/ja/

[*2] ゲームキューブで常時接続：http://kamishiro.gozaru.jp/pso/trial/001.html

6.5 ゲーム機のモデムと PC に接続した USB モデムとでデータをやり取りするには

PS2 と PC の USB モデムの間で内線電話をかけてあげればよいのではないかと考えました。電話番号の付与には VoIP ルータが使えます。ただしここでは VoIP としては使いません。LINE ポートが 2 つある製品の RT58i は内線電話のみとしても利用も可能でした。（もしかすると、内線電話が利用可能なターミナルアダプタでも代用可能かもしれません。）つまり、リスト 6.3 のような配線をします。

▼リスト 6.3　VoIP ルータ (RT58i) を用いた内線電話の配線

```
[PS2 モデム]----[VoIP ルータ（VoIP は使わず内線通話を使う）]----[USB モデム（PC）]
```

この配線にして、PS2 から PC に内線電話をかけます。そうすると、PC のモデムのシリアルコンソールに RING というメッセージが表示されました。

6.6 RING とはなにか

執筆している最中の今から振り返って思うと、当時、初見で RING とはなんのことだったのかまったく理解できていなかったなと思います。（執筆者はダイヤルアップ接続の世代ではなく ADSL からネットデビューした世代です。）結論をいうと PC に接続した USB モデムが、「今、電話かかってきてますよ」と教えてくれているのです。モデムではなく、アナログ電話で例えるとベルの音です。

かかってきた電話にモデムで応答するにはどうすればよいでしょうか？ まずここがサッパリわかりませんでした。執筆者側で調査をすすめるよりも先に、RING については先駆者の調査により 'ATA' で応答できることが判明しました。'ATA' をモデムのシリアルコンソールで叩くとモデムがかかってきた電話に対して電話をとった状態になるのです。これは、AT コマンドの一種で、モデムを操作するには AT コマンドを使って電話をとったり電話をかけたり、モデムの設定を変えたりできます。

ともかく、'ATA' コマンドを使って PS2 からの電話に応答することができました。応答すると、モデムのシリアルコンソールには 'CONNECT' といった表示がでました。モデムにはリザルトコードというものがあり、'CONNECT 33600' という表示がなされた場合は 33600bps で接続したという意味になります。（56K モデムの製品でも、スペック上は受信最大速度が 56k で送信最大速度は 33.6k となっているはずです。）

さて、PS2 からの電話に応答できるとバイナリデータが送られてきます。おそらくこれを対向の PS2 に中継すれば、PC を中継してモデム対戦ができるのではないかと考えました。これまでのネットワーク構成を振り返ってみましょう。リスト 6.4 が擬似交換機を用いた内線電話でのモデム対戦です。リスト 6.5 が VoIP を用いたモデム対戦です。

▼リスト 6.4　擬似交換機を用いたモデム対戦

```
[PS2 モデム]----[擬似交換機]----[PS2 モデム]
```

▼リスト 6.5　VoIP ルータを用いたモデム対戦

```
[PS2 モデム]----[(インターネットを経由した)VoIP]----[PS2 モデム]
```

これまで試したこの構成を、リスト 6.6 のようにしたいです。

▼リスト 6.6　PC の USB モデムを用いたモデム対戦

```
[PS2 モデム]-----------(USB モデム 1)[PC](USB モデム 2)-----------[PS2 モデム]
```

しかし、リスト 6.6 のようにモデム同士を直接接続してデータをやり取りすることはできません。リスト 6.7 のように、電話番号の割り当てとダイヤルトーン等を提供するために RT58i をモデムの間に挟むことにします。

▼リスト 6.7　PC の USB モデムを用いたモデム対戦

```
[PS2 モデム]---[RT58i]---(USB モデム 1)[PC](USB モデム 2)---[RT58i]---[PS2 モデム]
```

6.7　ゲーム機に PC から電話をかけるにはどうすればよいか

PS2 から PC に受け取ったデータを対向の PS2 に中継するにはどうすればよいでしょうか？ 動作確認に利用したゲームでは電話を受け取る役と電話をかける役に分かれて電話のやりとりをしていました。前の節で動作確認したのは電話をかける役の PS2 と PC 間でのお話です（リスト 6.8)。ということは対向の PS2 は電話を受け取る役をするべきでしょう。(対向の PS2 が電話をかける役でも大丈夫かは動作確認していません)

ということで、PC から対向の PS2 に対して内線電話をかけてみることにしました（リスト 6.9)。モデムで電話をかけるには AT コマンドの'ATD' を用います。ATD の後ろに電話番号を書けば OK です。PC から PS2 に電話をかけると CONNECT の表示の後、バイナリデータが送られてきました。これで、お互いの PS2 から PC にバイナリデータが受け取れた状況となります。あとは、このそれぞれの PS2 から送信されるバイナリデータを PC に接続した USB モデムで中継してあげれば良さそうです。

▼リスト 6.8　電話をかける役の PS2 を PC の USB モデムで終端するには

```
[PS2(電話をかける役)]----[RT58i]-----(USB モデム)[PC]
```

▼リスト 6.9　電話を受け取る役の PS2 を PC の USB モデムで終端するには

```
                    [PC](USB モデム)----[RT58i]----[PS2(電話を受け取る役)]
```

6.8　モデムを操作しつつモデムのデータをもうひとつのモデムに中継するには

USB モデムは PC ではシリアルデバイスとして扱えます。シリアルデバイスを簡単に扱え、サンプルコードも多く見かけた pySerial を用いることにしました。つまり Python を用いることになります。1 台の PC に USB モデムを 2 個挿して、それぞれの USB モデムを pySerial で AT コマンドを用いて操作します。次の手順のうち 1 番は手動でゲーム内の操作を行います。2~4 は Python スクリプトで処理を書きます。

1. （手動）ゲーム内で PS2 から PC に電話をかける操作をします。また、同じタイミングで対向の PS2 においては、電話がかかってきたら自動で電話をとるモードに入る操作をします。
2. PS2 から PC に電話がかかってきたら受話器をとるといった USB モデム 1 の操作を pySerial を用いて AT コマンドでモデムに指示します。うまくいけば CONNECT します。
3. PC の USB モデム 2 から PS2 に電話をかけるといったモデム操作を pySerial を用いて AT コマンドでモデムに指示します。うまくいけば CONNECT します。
4. 2 と 3 でそれぞれ PS2 と CONNECT したシリアルデバイスをつかって、それぞれのシリアルデバイスにデータをフォワードする処理を書きます.

フォワードする処理は、「USB モデム 1 からデータを読み取って USB モデム 2 に書き込む処理」と、「USB モデム 2 からデータを読み取って USB モデム 1 にデータを書き込む処理」をして実現しました。これで 2 台の PS2 間を PC が中継する形で問題なく通信対戦することができました。

しかし、まだ 1 台の PC の中で閉じています（リスト 6.10）。インターネットを介して対戦しているわけではありません。これを 2 台の PC にしてその間を TCP でやりとりするようにしましょう（リスト 6.11）。次の図では、PS2 と USB モデム間は RT58i をモジュラーケーブルで間に挟む必要がありますが、図が見づらくなってしまうため記載を省略しています。

▼リスト 6.10　1 台の PC に USB モデムを 2 つ挿してモデムのデータを中継するイメージ図

```
[PS2(電話をかける役)]----(USB モデム 1)[PC](USB モデム 2)----[PS2(電話を受け取る役)]
```

これが、リスト 6.11 のようになるイメージです。

▼リスト 6.11　2 台の PC を用いて LAN ケーブルでモデムのデータを中継するイメージ図

```
[PS2(電話をかける役)]-----(USB モデム 1)[PC]
                                    |
                              (LAN ケーブル)
                                    |
                                [PC](USB モデム 2)----[PS2(電話を受け取る役)]
```

はじめは LAN ケーブルを使わなくても 1 台の PC で localhost の宛先で通信するようにプログラムを書いていけばよいでしょう。localhost で動作に問題がないことが確認できれば、2 台の PC 構成にして LAN ケーブル等を使用して動作確認をします。

6.9　2 台の PC で TCP によるソケット通信

2020 年 12 月末に UDP でのソケット通信によりモデムデータ中継プログラムが動作しました。ただ、ローカルである LAN 内ではパケロス等もほぼ発生せず、安定して動作しましたが、インターネット越しに動作確認すると頻繁にゲーム内で通信エラーが発生してしまう問題がありました。UDP ではなく TCP で組む必要がありそうです。2021 年 3 月 ~9 月は検証作業が停滞してしまいましたが、TCP でちゃんと動作するものが 2021 年 10 月に完成しました。

あとはこの自作の中継プログラムを用いた際に、回線状況がどの程度悪い環境にゲーム側が耐えられるかが気になります。遠方のお友達とインターネット越しに対戦するとなると、どの程度の ping 値までは安定して対戦できるのかも測定したいです。

6.10　どの程度までの品質の回線に耐えられるか

KLab には PackDrop というパケットの遅延・パケットロスを再現する箱物のデバイスがあります。リポジトリは github[*3]にあり、また、詳細な解説記事は DSAS ブログに

[*3] PACKDROP の github リポジトリ： https://github.com/pandax381/PACKDROP

記事があります。[*4] PackDrop には LAN ケーブルを 2 つ刺す口があり、たとえば 2 台の PC の間に PackDrop を LAN ケーブルで繋ぐことで、その間の通信を指定した遅延・パケットロス率で通信テストをすることが可能となります。（リスト 6.12）

▼リスト 6.12　PackDrop を用いたネットワーク遅延・パケロス発生時の配線例

```
[PC 等]--------[PackDrop(遅延・パケロスを意図的に発生)]---------[PC 等]
```

工夫すれば WiFi でもネットワーク遅延を再現することができ、無線アクセスポイントとルータやスイッチの間でこの PackDrop を挟み込むことで、スマートフォンアプリのネットワーク遅延・パケロス試験が行えます. しかし、PackDrop は会社にあるものなので自宅で試すことができません。過去の MakerFaire で基板が配られていたのですが、自分は持っておらず、自作するしかありません。

そう思ったのですが、PackDrop は内部で tc コマンドを発行して遅延・パケロスを再現しており、その遅延具合とパケロス具合を「つまみでぐりぐり設定する」インターフェースを提供することで、エンジニアでない方でも分かりやすく使えるような工夫がされているものです。そのため、tc コマンドが使える Linux マシンと NIC を 2 つ用意すれば、PackDrop が手元になくても遅延・パケロスの試験が行えます。今回は localhost に対して TCP でデータをやり取りする中継プログラムを使い、tc コマンドで localhost に対して遅延・パケロスをかけてどの程度の環境に耐えられるか軽く確認を行いました。Linux マシンに USB モデムを 2 つ挿し、localhost に対して tc コマンドで遅延・パケロスをかけるためこの場合は NIC は 2 つ無くても大丈夫です（リスト 6.13)。

▼リスト 6.13　モデムデータ中継プログラムにおける遅延・パケロス試験の配線イメージ図

```
                    [PS2]
                      |
                      |
                (USB モデム 1)
[Linux マシン (localhost に tc による遅延・パケロスをかける)]
                (USB モデム 2)
                      |
                      |
                    [PS2]
```

動作確認の結果、遅延設定は入れずにパケロス設定を入れたタイミングでゲーム画面が両者同時に体感で 0.3 秒ほど固まる事象が確認できました。

[*4] PACKDROP の DSAS ブログ記事: http://dsas.blog.klab.org/archives/raspi-netem1.html

AC2AA は完全同期型のネット対戦ゲームなのか？

完全同期型（キー入力同期方式）とは、たとえば A と B というそれぞれがネット越しに接続されたゲームにおいて、1 フレーム毎にお互いにパケットを送信して、A のキー入力がなされてもすぐに A のゲーム画面に反映せずに、先に B へキー入力データを直近のパケット送信タイミングで送信します。データを受け取った B は、A へ「データが届いたよ」と（直近のパケット送信タイミングで）返事をします。A は B からの返事を受け取ってはじめて A 側でキー入力を A 側の画面に反映するといったものです。パケットロスが発生すると、ゲームの処理や描画処理を一時停止し、相手からの次のパケット到着でリカバリを行います。過去の動作試験で、手元の PS2 のコントローラで操作した結果が対戦相手側 PS2 のゲーム画面に反映されてから手元のゲーム画面に反映される挙動を確認していることや、座標ワープが発生したことがなく、これらの状況をふまえて AC2AA は完全同期型のネット対戦ゲームであると執筆者は判断しました。また、先ほど述べた完全同期型というワードの説明については CEDEC2010 の「ネットワークゲームの仕組みとゲームデザイン」[*5]というセッションを参考にしています。

このパケロスの影響もなるべく小さくする方法はあると思うのですが、執筆の時期が来てしまい、詳しい調査はまだできていません。パケロスについては 0.3 秒ほど固まる問題は現時点では回避できていないのですが、遅延の具合によってはどの程度まで耐えられるか確認してみます。初めは「5ms の遅延→問題なし」の動作確認から行い、「10ms の遅延→問題なし」「30ms の遅延→操作に多少ラグは感じるがゲーム内通信エラーは発生しない」と徐々に遅延を大きくして "ゲーム内で通信エラーが発生" するまで遅延を大きくします。結果、60ms の遅延をかけると通信エラー発生が割と発生し、現時点での構成ではこれが安定してプレイできる限界と判断しました。つまり、通信対戦をしたいお友達同士が ping 値で 60ms 以上の場合、プレイできないということになります。

6.11　インターネットへ接続する自宅のネット回線について

普段なにげなく快適にインターネットの利用ができている方は意識することはないと思うのですが、時間帯によってインターネットへ繋がりにくいといった状況が発生する回線が存在します。集合住宅等でインターネットを利用するために既存のアナログ電話回線を流用して各部屋にインターネット接続を提供する VDSL には、マンション供用部までは光回線できているものの、そこからは各部屋への回線は既存の電話回線をつかってイン

[*5] CEDEC2010, 株式会社セガ, ネットワークゲームの仕組みとゲームデザイン：
https://cedil.cesa.or.jp/cedil_sessions/view/306

ターネットへのアクセスを提供しているケースがあります。既存の電話回線が使えるという点ではかなり画期的なのですが、同時にインターネットを利用する人数が増えると回線が混雑してしまうことがあります。ということで VDSL 回線と戸建てで使われるような FTTH の回線とで通信対戦の動作確認を行いました。24 時ごろに試したのですが、結果としてはは安定してプレイできず、ゲーム内で通信エラーが発生してしまいました。現状の中継プログラムでは VDSL 回線には非推奨と捉えたほうが良さそうです。

6.12 どの程度の快適さを犠牲にしてより遠方のお友達と対戦するか

現状の構成で、仮にインターネット回線のレイテンシが 0 秒だったとして、残りの箇所すべてを合算したレイテンシがどの程度あるのか見積もってみます。動作確認に利用しているゲームは完全同期型対戦ゲームなので、レイテンシは少なければ少ないほどよいはずです。

そこで、先駆者の発案・計測により EGBROWSER LIGHT（PS2 用 web ブラウザソフト）と Windows10 の PC 間で PPP 接続をして PS2 に IP アドレスを払い出し、ping 値が計測されました。計測には株式会社アイ・オー・データ機器より発売された PS2 用 USB モデム P2GATE を使用しました。また、PC の USB モデムにはプラネックスコミュニケーションズ株式会社より発売された PL-US56K2 という型番の製品を利用しました。結果は、速い場合は 97ms を記録していました。統計はとっていませんが、97ms~110ms の範囲をウロウロしていました。動作確認に利用しているゲームは 60fps なので、1 フレームはおよそ 16ms です。P2GATE と PL-US56K2 間の ping 値に 97ms かかっているということは、97/16=約 6 フレームの遅延です。手元の PS2 と PC 間でこれだけかかっていますが、お友達側の PS2 と PC 間でも同じだけレイテンシがかかると見積もってよいでしょう。つまり、インターネットのレイテンシが 0 秒だとして、全体のレイテンシは 6+6=12 フレームとなります。

次はインターネットのレイテンシも考慮して見積もってみましょう。近郊ならともかく、速すぎず遅すぎずのレイテンシとして 32ms かかるとします。これは 2 フレーム分ですね。このインターネットのレイテンシを含めて、60fps ゲームにおいて合計 14 フレームの遅延にプレイヤーが満足できるかどうかとなってきます。お友達が同じ市内に住んでいる等近場で対戦できるのであれば、場合によってはインターネットのレイテンシは 1ms になることもあると思います。ただ、その場合でも 12 フレームの遅延がかかります。

多少の遅延があってもゲームシステム上問題ないゲームであればこのラグは許容できますが、この結果をみたときは、モデムで完全同期型ネット対戦ゲームを自分が新たに作る場合は 60fps ゲームの場合は 18 フレーム程度の遅延が常時かかっていてもゲームプレイに差し支えないようにゲームシステム上で作っておきたいなと感じました。ゲームモードなどの低遅延描画モードがない液晶テレビをお使いのご家庭も考慮したいとなると、もっ

と猶予を持たせるべきかもしれません。(例:自分側のゲーム画面で敵の攻撃が開始した演出が描画されてから自分の操作キャラが回避行動をとって回避できるまでの猶予時間を最低でも 18 フレームはゲームシステム上確保する等)

6.13 VoIP ルータは必要なのか

現状、ゲーム機のモデムと PC に接続した USB モデム間で通信を確立させる、ただそれだけのために VoIP ルータ(RT58i)を間に挟んでいます。しかし、それだけのためにこの機材を使うというのは、割とかさばってしまいお家のスペース的に少し問題があります。なんとか、省スペース化が実現できればよいのですが...。というのも、アナログ電話の技術情報がインターネットにあまり転がっておらず、この点については大変頭をなやまされました。図書館等で古い書籍に頼る必要があるのではと諦めかけていたタイミングで貴重なインターネットの情報に出会いました。

「音響カプラを自作して見る実験」というブログ記事[*6]では、黒電話そのものを音響カプラとして利用するために、黒電話とモデム間をモジュラーケーブル等の 2 線で繋ぎ、そのケーブル自体に電池と抵抗を取り付けて、黒電話の受話器のマイクの電源としているようです。このブログ記事のおかげで、モデム間の通信の確立には電池と抵抗を使う方法があるのだと知ることができました。しかし、私が実現したいのは黒電話ではなくゲーム機のモデムと PC に接続した USB モデム間での通信の確立です。それに私は、3x3x3 の簡単な LED キューブを制作した経験はありますが、逆にいえばその程度の知識しか持ち合わせておらず、アナログ電話等の知識はまったく持ち合わせておりません。もし、誤った回路を作ってしまってモデムを壊してしまったら取り返しがつきません。そのため、こちらのブログ記事で書かれていた回路を作って試すことは保留としました。

その後、社内の雑談でモデムに詳しい方から VoIP ルータいらないのではというツッコミをやはり、いただきました。というのも 9V 電池で通信の確立ができるというのです。そして、ドリームキャストのモデムと USB モデム間を 9V 電圧で通信を確立している記事[*7]を教えていただきました。こちらのブログ記事には回路図が掲載されており、これを参考にすれば簡単に PS2 でも試せそうだと感じました。また、ドリームキャストで成功事例があるのだし、「PS2 でも、機材を壊す可能性は否定できないがうまくいくかもしれない」という考えが芽生えました。そして、秋月電子通商等で必要なパーツを買い集め、まずはブレッドボードで問題なく動作するか確認してみることにしました。ここでは、そんなモジュラーケーブルに電圧を印加する機材を "line voltage inducer" と呼ぶことにします。しかし、PS2 で試してみたところうまくいきません。回路の結線を間違えてしまったのかと思い、回路を見直してみましたが間違っているところはみあたりません。そこで、1 台の PC に 2 つ USB モデムを接続し、その USB モデム間をモジュラーケーブル

[*6] 音響カプラを自作してみる実験: https://mzex.wordpress.com/2014/10/18/407/

[*7] Raspberry Pi 経由でドリキャスをネット接続可能にする DreamPi 試してみた: https://pomegd.hatenablog.com/entry/2020/09/15/190000

と 9V line voltage inducer（ブレッドボードで制作したもの）を間にはさんで USB モデム 1 で ATD コマンド、USB モデム 2 で ATA コマンドを試しましたが、これもまたうまくいきません（リスト 6.14）。

▼リスト 6.14　9V 電池での Line Voltage Inducer によるモデム間通信の配線イメージ図

```
+------(USB モデム 1)[PC](USB モデム 2)---+
|                                        |
+-------[9V Line Voltage Inducer]------+
                  |
                  +----(9V 電池)
```

うまくいかなかったものの、リザルトコードには 'NO DIALTONE' という表示がありました。これまで擬似交換機や VoIP ルータを使ってきましたが、それはつまりダイアルトーンである「プー音」がある環境下でしか CONNECT をしたことがないということです。電池のみの場合はダイヤルトーンがないので、どうしたものか...。そこで、DIALTONE がない環境下で CONNECT する方法がないか調べることにしました。

ググった結果、ダイヤルトーンがなくても通信の確立ができるモードは AT コマンドで設定ができ、ATX3 という、ダイヤルトーンを無視するコマンドがあることがわかりました。これは、海外のモデムを日本で使用する際にダイヤルトーンが異なるため、そういった場合でもダイヤルトーンを無視して利用できるときのために設けられた機能なようです。日本のダイヤルトーンの音は、固定電話があるお家の受話器から聞こえるのでご存知の方も多いかと思います。一方、海外のダイヤルトーンには馴染みがない方が多いと思います。まったく聞き覚えがないということもなく、私は洋画で固定電話をする際にダイヤルトーンが流れるシーンで聞いたことがあったので、かろうじて聞き覚えがありました。YouTube 等で国内・海外のダイヤルトーンの音声を聞くことができます。

話を元に戻しましょう。モデムに対して AT コマンドで ATX3 を発行してあげてから、一方の USB モデムで ATD コマンドを、もう一方の USB モデムで ATA コマンドを叩きます。すると、USB モデム間では（ATX3 を設定することで）すんなりと CONNECT に成功しました！ CONNECT してからは、（netcat で 2 つのターミナルウィンドウ間でテキストチャットができるように）2 つのモデム間でテキストチャットができます。お次は PS2 のモデムで試します。しかし残念ながら、PS2 ではモデム間の通信の確立ができませんでした。PS2 のモデムでは AT コマンドをユーザーが発行することは、動作確認で利用したゲームでは不可能です。動作確認で利用したゲームにおいては、ダイヤルトーンがなければダメみたいです。ただ、どうしても諦めきれなかった私は、本当の原因がなんなのか仮説を立てました。また、CONNECT に必要な前提条件としてモデムの設定に ATX3 が必要です。

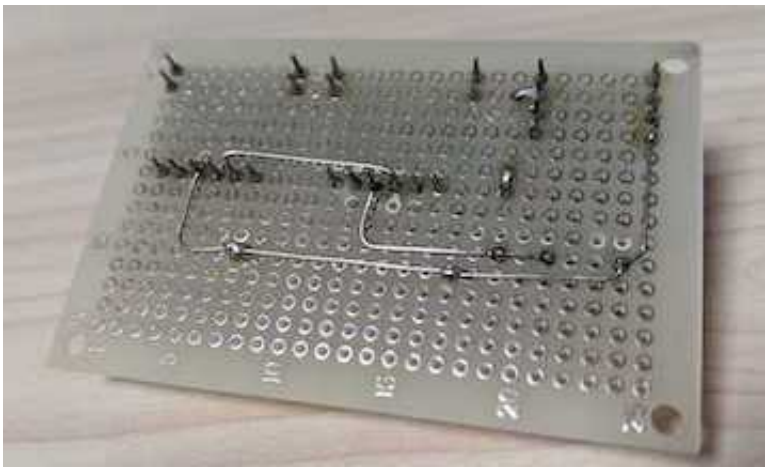
1.（仮説）本来の電話回線の電圧は 48V 程度であるため、9V では電圧が不足しているのではないか。
2.（事実）ゲーム内のモデム初期化設定にダイヤルトーンを無視する ATX3 コマンドが設定されていないという確証は取れていない。

藁にもすがる思いで、ダイヤルトーンがない 48V 印加の回路を用意して試そうと思いました。ただ、48V という電圧を用意することは難しそうです。そうこうしていると、先駆者から、「DreamPi 方面で PAL 版ドリームキャストにおいては電圧が足らないらしくて繋がらなく、ヤケクソになって 9V 電池を 2 個直列につないで試したらいけた（意訳）」というブログ記事があるという情報を教えてもらいました。[*8] ただ、9V 電池を直列にする道具が手元にありませんでした...。偶然にも 19V まで任意の電圧が出せる直流安定化電源がお家に転がっていたのでこれを使うことにしました。ブレッドボードで動作確認したところ、なんと！ うまくいきました。AC2AA で 18V 電圧による CONNECT に成功したのは世界でこれが初かもしれません。

動作に問題がないのでユニバーサル基盤で量産もできますね。というわけで（図 6.4, 図 6.5）のように簡単ではありますが自作基盤を作成しました。DC ジャックには直列接続した 9V 電池 2 個を DC プラグで刺すことを想定しています。9V 電池を 2 個直列に接続するには、共立エレショップで販売されている「006Px2 本用電池スナップ」（型番:BS-M-4A）を利用しました。このスナップのケーブルと DC プラグのケーブルを半田付けし、熱収縮チューブで皮膜して接続しています（図 6.6）。

▲図 6.4　ユニバーサル基盤で作成した inducer

[*8] DreamPi 1.0 Released!, Kazade's Internet Address:
https://blog.kazade.co.uk/2015/12/dreampi-10-released.html



▲図 6.5　ユニバーサル基盤で作成した inducer（裏面）

▲図 6.6　DC プラグを取り付けた 18V 電源

ただ、この嵩張る VoIP ルータを小型の自作基盤に置き換えることができるのは、電話をかける役の PS2 のみです。（動作確認で利用したゲームソフトでは）電話をとる側の PS2 は交換機からのリングトーンを検知して電話をとるようなので、電話を受け取る側は嵩張る VoIP ルータを使う必要があります。

また、自作基盤を使うにはひとつハードルがあり、PS2 が電話をかけたタイミングを

狙って PC モデム側で ATA（受話器を取る AT コマンド）を発行する必要があります。というのも、リングトーンがないため PC モデムが RING ステータスを表示できないのです。リングトーンがない状況下で電話をとるには、ゲーム内で電話をかけはじめるタイミングをあらかじめ調べておく必要があります。動作確認に使用したゲームでは、電話をかけるボタンをおすと、そこから数秒後に一瞬画面が暗転するタイミングがあり、そこで電話をかけるようになっていました。タイミングの確認用に電話の受け取り役としてアナログ電話を配置するとベルの音が鳴るのでわかりやすいです。

6.14 NAT 越え

現状では電話を受け取る役のプレイヤーはポート開放をしてサーバー役となり、対戦相手のクライアントからの接続を受け付ける必要があります。ただ、おうちのルータのポート開放設定ができないお友達もいると思います。そこで NAT 越えを実装することにしました。

NAT 越え技術に有名なものとして STUN、TURN があります。TURN は常に中継用のサーバーが必要なため、TURN サーバーの場所によってはレイテンシが増大してしまう恐れがあります。（例: 国内で対戦したいのに TURN サーバーが海外にある等。この場合はたとえば日本から海外にパケットを送ってから海外から日本にパケットが届くという経路になってしまい、レイテンシがかなりかかってしまいます。）STUN サーバーは NAT 越えをするにあたり、通信開始の初回のみ中継サーバーを利用するだけで、NAT 越えに成功してからは中継サーバーを経由する必要はありません。レイテンシをなるべく少なくする意図で STUN による NAT 越えを実装することにしました。

STUN の NAT 越えで有名な技術として UDP holepunching があります。ただ、今回の中継プログラムで必要なものは TCP です。（UDP で対戦すると、高頻度でゲーム内で通信エラーが発生してしまいまともにプレイできません。）しかし、UDP のほうが実装が簡単らしく、実装した経験が自分には無かったので、勉強がてら UDP holepunching の実装をすることにしました。ただ、UDP holepunching も実装する際にハマりポイントがあり、かなり苦労はしました。お次は TCP holepunching を実装します。

6.15 TCP holepunching

こちらについては理論は学び終わり、実装も最低限できたのですが、自分の NAT 環境でしかうまく NAT 越えができない実装となってしまいました。この辺りは詳しく書いていきたいのですが、検証途中というのもあり、後日 KLab の技術ブログである「KLablog - Technology[*9]」にて情報を公開できればと考えています。

[*9] KLablog - Technology: `https://www.klab.com/jp/blog/tech/`

6.16　中継プログラムのソースについて

低遅延・パケロス発生頻度が少ない環境においては、安定してプレイできているので完成ともいえるのですが、他に改良を加えてゆきたい開発途中のものなので現時点では公開していません。冒頭で「主張する」と書いたのはソースを公開しておらず、証拠を提示できないためです。また、Python で実装しており、いま流行りの Rust 等で読者のみなさまに実装いただいた方が場合によってはより低レイテンシなものができるかもしれないため、そこに期待して、実装方針だけこの記事に書き留めました。

6.17　あとがき

本文ではあっさりとこなしたかのように書かれていますが、実際はもっと泥臭い試行錯誤が繰り広げられています。先駆者の方にて 2020 年 1 月頃から着手しはじめ、2 月には RT58i（VoIP ルータ）での通信対戦ができることが確認されました。ただ、VoIP 設定のパラメータによってラグが変わることが分かり、2 月からはずっと VoIP のパラメータ調整による、よりよい通信対戦の安定性が模索されました。よりよい設定になっているかの確認には、コントローラのキー入力がなされてから何フレーム後に画面に反映されたかを定量的に計測することで確認し、よりよい設定が模索されました。2 月からはもうずっとその試行錯誤がなされており、2020 年 11 月に執筆者である私が検証作業に参加しました。

私がメインで実装を進めていた PC の USB モデムによる中継方式については 2020 年 12 月に UDP で、不安定ながらも通信対戦が中継プログラムで動作しました。その後は数カ月単位で停滞していた時期（2021/03~2021/9 月）もありましたが 2021 年 10 月ごろにようやく TCP で安定して対戦できるものが仕上がりました。個人的には、ゲーム機のモデムと PC の USB モデム間の通信を確立しようと試みた時期が一番難易度が高かったです。これはモデムに関する知識がゼロだったためです。針に糸を通すような際どい可能性に希望を抱いて、「ダメかもしれないけどできるかもしれない」といった想いで諦めずしつこく何度も検証作業をすすめました。

いくつか試行錯誤の例をあげましょう。PS2 モデムと PC モデムとで通信を確立するためにモデムの通信速度などの設定をいくつか試したりしました。が、なかなかうまくいきませんでした。モンキーテスト感はありますが、PS2 から PC に電話がかかってきたら ATA をたたき、すぐ ATH0 するとなぜか一瞬だけ CONNECT してすぐにエラーになるという謎の挙動に遭遇しました。ただ、「PS2 と PC 間で CONNECT に成功した!」という事実に大喜びして、試行錯誤を続けるモチベになりました。他にも、普通のアナログ電話と PC モデムをつなげて、PC からアナログ電話に電話をかけて、アナログ電話の受話器をとると、ピーガガガの音で耳が少しやられるということも今となってはよい思い出です。

PC モデムの初期設定に設定する通信速度などについて、はじめは何を設定すればよいのかサッパリわかりませんでした。しかし、そうした行き詰まった時に「もしかしてアレはつかえるのではないか」というものも思い当たりました。それは、EGBROWSER LIGHT のモデム設定を参考にすればよいのでは、ということです。EGBROWSER LIGHT とは、米国の PlanetWeb 社が開発し、株式会社エルゴソフトが発売した PS2 で利用できる Web ブラウザ・メール送受信ソフトです。LIGHT 版はブロードバンドは利用できず、PS2 用の対応モデムのみ利用可能です。この EGBROWSER LIGHT は、モデムの設定画面をみても、どのような設定がなされているかは確認することができません。しかし、ある方法で確認することができると気づきました。モデム接続時に、実行した AT コマンドとそのコマンド結果をログ表示するモードがあり、このログにはモデムの設定初期化用の AT コマンドも一瞬ですが出力されます。オプション設定でモデム設定初期化用の AT コマンドがユーザーの入力で設定できるため、ここを利用して（モデム設定初期化用の AT コマンドではありませんが）現在のモデム設定を、ユーザーが入力した AT コマンドで確認することができます。AT コマンドのマニュアルを参照したところ 'AT+MS?' という AT コマンドで現在の変調方式や通信速度設定が確認できるようです。この AT コマンドをモデム設定初期化用のコマンドとして設定します。そしてモデム接続を開始し、画面スクロールされながら流れて表示されるログをスマホのカメラで動画撮影し、'AT+MS?' コマンドの結果をメモります。そうして、PC 側の USB モデムの設定初期化コマンドとして適用することでスムーズに PS2 モデムと通信を確立することができました。この発想を思いつくには、興味本位で EGBROWSER LIGHT を起動して、触ってみていろんな設定項目を眺めていたという、一見無駄に思える経験が後から思うと重要なことでした。また、AT コマンドによるモデムのボーレート設定に慣れている方はそんなことはせずにあっさりと設定をこなせるかとおもいますが、私は AT コマンド初心者だったため、AT コマンドの設定例がデバッグログで分かる EGBROWSER LIGHT は大変助かりました。

また、PC でモデムを扱う際に read() で少しでもブロッキングしてしまうと当然ゲーム側で通信エラー判定がなされてしまうため、ブロッキングしないように実装してあげる必要があります。また、モデムに対して AT コマンドを発行した後にレスポンスとして同じ文字列を返す動作モード（コマンドエコー）にしていると、その文字列を読み込む処理を組む必要があります。読み込む文字数指定をちゃんと調整してあげないと、そこでブロッキングしてしまったりしてしまいますし、逆にバッファに溜まっている文字を自動で読み込むようにすると対向の PS2 に中継しないといけないデータまで読み込んでデータを捨ててしまうということも起こってしまいます。モデムの初期化用 AT コマンド発行時に、エコー文字列を read してデータを破棄する処理を適切に組むのに 4 時間くらいかかってしまいました。今思うと、コマンドエコーは使わない方が良かったかもしれません。

また、pySerial をつかってプログラムによるモデム操作で電話をかける際に、相手（PS2）が電話をとらなかったら無限ループで電話をもう一度かけ直すといった処理を書きました。しかし、ここに私は予想していなかった仕様があり、電話をかけて 160 秒程

度電話に応答がなかったら（検証で利用した USB モデムにおいて、ATD コマンドでダイヤルして相手が応答せずに 53 秒程度経過すると NO ANSWER のリザルトコードが表示されるため即座に ATD コマンドでリダイヤルする、というのを繰り返し、合計の秒数が 160 秒程度相手が応答しなかった場合)、モデムは電話をかけるのを諦め、その後 1 分 10 秒程度は電話をかけてはならないという仕様があったのです。この電話をかけてはならない時間中に ATD コマンドで再び電話をかけると、モデムのリザルトコードとしては WAIT とその待機秒数（DELAYED time）が表示されます。その待機時間を経過しなければ、モデムで電話をかける ATD コマンドを何度発行しても、WAIT と残り秒数が返ってきてしまいます。この対応方法として、「待機時間が後何秒です」といったログを表示して、ユーザーに時間が経過するまで待ってもらい、'y' キーと Enter を押すことでリダイヤルする処理を実装しました。上記のように、モデムのリザルトコードの正確な読み取り処理や条件分岐を実装するのに 4 時間程度かかってしまいました。後から調べて分かりましたが、この制限についてはリダイヤル制限と呼ばれており、端末設備等規則の第十一条の第 3 号にて規定があります。

また、この中継プログラムを作り上げた後に、2001 年当時になるべく通信料金を抑えてネット対戦する方法がなかったのか考えて調べてみました。そのためには当時のインターネット環境について調べる必要がありました。（調べるまで ISDN がどういった回線だったのかすら理解していませんでした。）調べた結果、定額の ISDN を利用している場合でも特定のアクセスポイント（電話番号）にしか定額の扱いにならないため、友達の固定電話番号に電話して対戦する AC2AA では、定額でネット対戦をする方法はテレホーダイ以外にはなかったのではと考えています。

せっかくなので執筆者が目の当たりにしたインターネット回線の進化についても触れたいと思います。2004 年頃に自宅にパソコンがやってきて、インターネット接続もできるようになり、ネットにデビューしました。当時は ADSL を使っているということしか知りませんでした。ネットサーフィンをしていると映画の PV やフリーゲーム等をダウンロードする際に「ブロードバンドをお使いの方はこちら！」「ナローバンドの方はこちら！」といったダウンロードをリンクをよく見かけたものです。当時はブロードバンドやナローバンドがなんなのかよく理解せず、「よくわからんが ADSL はブロードバンドらしいぞ」という浅い理解でネットサーフィンをしていました。今になってようやく、ナローバンドが PSTN を用いたダイヤルアップ接続や ISDN のことを指していたんだと理解しました。そうして、当時自分が置かれていた環境が比較的によい環境（ナローバンドに対して）ということに驚きました。

しかし、4 年経った 2008 年頃には、FTTH である光回線が普及し、より高帯域が求められる動画配信サービスが流行します（逆だったかもしれません、より広帯域が求められる動画サービスが流行し、FTTH である光回線が普及したのかもしれません)。2004 年頃はかなりよい環境だった ADSL も、そうした動画配信サービスでは帯域が不足してしまい、動画を視聴していると 10 秒間隔で読み込み中のグルグル状態に陥ってしまったりしました。高画質しか提供していないストリーミングサイトの動画なんて見れたものではな

かったのです。2008 年ごろも執筆者はいまだ ADSL を使い続けていて、YouTube やニコニコ動画も動画の読み込みが始まったら、まず初めに停止ボタンを押して、5 分くらい先の動画までダウンロードデータが貯まるまで待って再生ボタンを押すといったことをしていました。ただ、5 分も貯めるのに数分かかったりするので我慢できずに再生ボタンを押して、結果、動画のよいところで読み込み中のグルグルになってしまったりもよくありました。あの頃は忍耐が必要だったなと思います。画像のダウンロードも時間がかかり、png はダウンロード中でもモザイク状で全体像で確認ができる（インターレース）けれども jpg は上から徐々に画像が読み込まれるのを数分間眺める忍耐が必要でした。無料で最新映画一本をみれるキャンペーンに当選し、「やったー！」と喜んだのもつかの間、高画質しか提供しておらずなくなく視聴を諦めたこともありました。

こうしたネット回線の歴史的進化をここに記すことで、なにを読者にお伝えたしたいのかを言いますと、今となってはネット回線側に問題があること意識することは昔に較べてほとんどないですが、ネット回線を利用するアプリケーションによっては、ネット回線側がボトルネックとなる可能性も頭の片隅に置いていたほうがよいのではないか、という点です。本記事で述べた完全同期対戦ゲームのモデムデータ中継プログラムを VDSL で安定して利用できなかったことは、その一例です。回線の安定性、通信速度などの品質と、その回線を使って送るデータの必要な帯域や、帯域が細くてもパケロス・ジッタが少ない場合は、たとえゲームプログラム側でケアされていない場合でも安定してネット対戦ができることもありますし、その逆（回線の品質が悪い状況でもゲームプログラム側でケアしてあげることで影響を可能な限り小さくするなど）もあります。そういったケースがあることを知っている方が増えていただければ幸いです。また、今回は LAN ケーブルの品質（断線していたり）等に注意を払う必要もありました。遅延やパケロスが頻発すると、ゲーム内で通信エラーが頻発しやすい中継プログラムなためここは大変重要です。動作確認しているメンバー内で、有線である LAN ケーブルを使っているにもかかわらず、割と高い頻度で画面が瞬間的に固まる事象が発生しました。瞬間的に画面が固まるのはパケロスが発生した時に見受けられるものです。（必ずパケロスが発生しているといえるものではないです。）先駆者からの情報提供で、その後の調査で LAN ケーブルから Wifi に切り替えて動作確認した方が安定してプレイできるといった状況であったと教えていただきました。おそらく LAN ケーブルが断線等しているということでした。その後、新しい LAN ケーブルで対戦を試みたところ正常に対戦ができたため、LAN ケーブルに問題があったことが判明しました。

また、話題が変わりますがオンラインコミュニケーションに力を入れていたドリームキャストにはモデムが標準装備されていたと知り衝撃を受けました。PS2 やゲームキューブではモデムは別売りとなっていて、モデムでのインターネット接続・ネット対戦に対応しているゲームを遊ぼうとなると、ソフトとは別でモデムを買う必要があるのです。いくつか中古で P2GATE（PS2 専用の USB モデム）を買い漁りましたが、ドリームキャストだったらこの徒労が生まれなかったと思うとドリームキャスト様様となっていました。また、2012 年ごろに P2GATE が大阪日本橋で 50 円で 100 個くらいワゴンセールをして

いたときに 2 個しか買わなかったのですが、タイムマシンがあれば当時にタイムスリップして全部買い取っていたことでしょう。

本記事を執筆しているときに気付いたのですが、1988 年に発売されたメガドライブも 1990 年にメガモデムという名でモデムが発売されていたんですね... 驚きました。もしかするとメガモデムでも本記事にて述べた中継プログラムでオンライン対戦ができるかもしれません。

ここまで読んでくださりありがとうございました。

執筆者・スタッフコメント

第 1 章 Masato Yamada / @thunder_hey

普段執筆することがなかったので新鮮な気持ちで書かせてもらいました。

第 2 章 Shunsuke Ito / @fgshun

コーヒーいれて。のんびり、まったり。

第 3 章 Naoki Hamada / @hmkz_

青ヶ島からリモートワークしたい。都内だしいけるかな？

第 4 章 Takuya Hashimoto / @hasi_t

ぬるめた 2 巻たのしみ

第 5 章 Daisuke Makiuchi / @makki_d

眼鏡っ娘が好きです

第 6 章 Tomoaki Fude

ほんとうのさいわいとは一体なんだろうか。私が生きてきた証をここに残せた気がします。

企画進行・イラスト・デザイン

Toshifumi Umezawa

開始遅かったのにみんなスケジュール守ってくれているおかげで進行が楽で感謝しかない...

kentaro gondo @gondo-k

表紙のキャラデザインと着彩、仕上げを携わらさせていただきました！ 新鮮でとても楽しい経験をありがとうございました！

yilin cai

表紙デザインに携わらせていただきました。とても良い経験になりましたので本当にありがとうございます。

既刊・電子版ダウンロード

https://www.klab.com/jp/blog/tech/2022/tbf12.html

KLab Tech Book Vol. 9

2022 年 1 月 22 日　技術書典 12 版（1.0）
著　者　KLab 技術書サークル
編　集　梅澤 寿史、牧内 大輔
発行所　KLab 技術書サークル
印刷所　日光企画

KLab Tech Book Vol.9