「ALGORI」システム仕様書

変更履歴

更新箇所	更新内容	更新日
全体	新規作成	2023/10/02
３．「UNO」開始までの作業手順	テキスト修正	2023/10/06
４．プレイヤープログラムの開発～提出までの手順	Dockerfile の作成手順に python 用の説明を追加	2023/10/06
全体	仕様書内で使用している文言を他ドキュメントと統一	2023/10/13
３．「UNO」開始までの作業手順	解凍先のパスを修正	2023/10/26
５．データ通信仕様について	special-logicの説明を修正	2023/10/26

目次

1．ALGORI 大会概要

（1）はじめに

当資料は **『ALGORI -UNO プログラミングコンテスト- 』大会** （以下、ALGORI 大会）において使用するシステム及び、参加者が開発するプレイヤープログラムを実装するための様々な機能を説明する仕様書となります。
ALGORI 大会は、ご存知のとおり「UNO」をテーマとしており参加者各位が優勝を目指すため、参加者独自のプログラムを作成して競い合います。
普段遊んでいる「UNO」を別の角度から楽しんでもらうため、弊社は ALGORI 大会を開催する環境として「ALGORI システム」を構築しました。
その構築した環境では 2 つのプログラムを使用して ALGORI 大会の運営を行っていきます。

- 「ディーラープログラム」
  弊社（以下、ALGORI 大会運営）が管理するプログラムで「UNO」の進行、および管理を行います。
  ※機能詳細については、「2．（3）.ディーラープログラム」を参照してください。

- 「プレイヤープログラム」
  大会に参加される皆様が作成および管理を行うプログラムです。
  エントリー後、公式 HP 経由でプレイヤープログラムの提出を行っていただくことにより他の参加者との対戦が可能となります。
  ※機能詳細については、「2．（4）.プレイヤープログラム」を参照してください。

（2）配布資料について

本大会では参加者の皆様に以下の資料を配布しています。

1. 大会ルール
   - UNOのルール及び本大会のルールを解説した資料です。「ALGORI 大会」公式 HP の「応募要項」画面内「大会ルール」よりダウンロードしてください。
2. システム仕様書（当資料）
   ALGORI 大会にご参加される皆様へ、以下の 4 点について必要な詳細情報、および実際の作業手順と接続例等を掲載しております。
   - 必要な情報の提供
     - ALGORI 大会のシステム
     - 「UNO」開始までの作業手順
     - データ通信仕様
     - その他機能
   - イベントについて
     当資料において明記している「イベント」とは、以下の定義となります。
     - ディーラープログラムがゲームを進行するために必要な行動

    - 「UNO」をプレイするためにプレイヤープログラムがとる行動
    - ALGORI 大会専用イベント「スペシャルロジック」

3. ゲームログ仕様書
   当資料と同階層にあるgame-log.pdfです。
   本システムでは、試合におけるプレイヤーの動作を1行動ごとに1つのJSONで記録し、MongoDBおよびログファイルに記録します。
   このゲームログ仕様書では、記録されたJSONの各フィールドの意味を解説し、試合の流れを把握したり、皆様が開発したプレイヤーの行動の分析に利用して頂けます。

2．ALGORI 大会のシステムについて

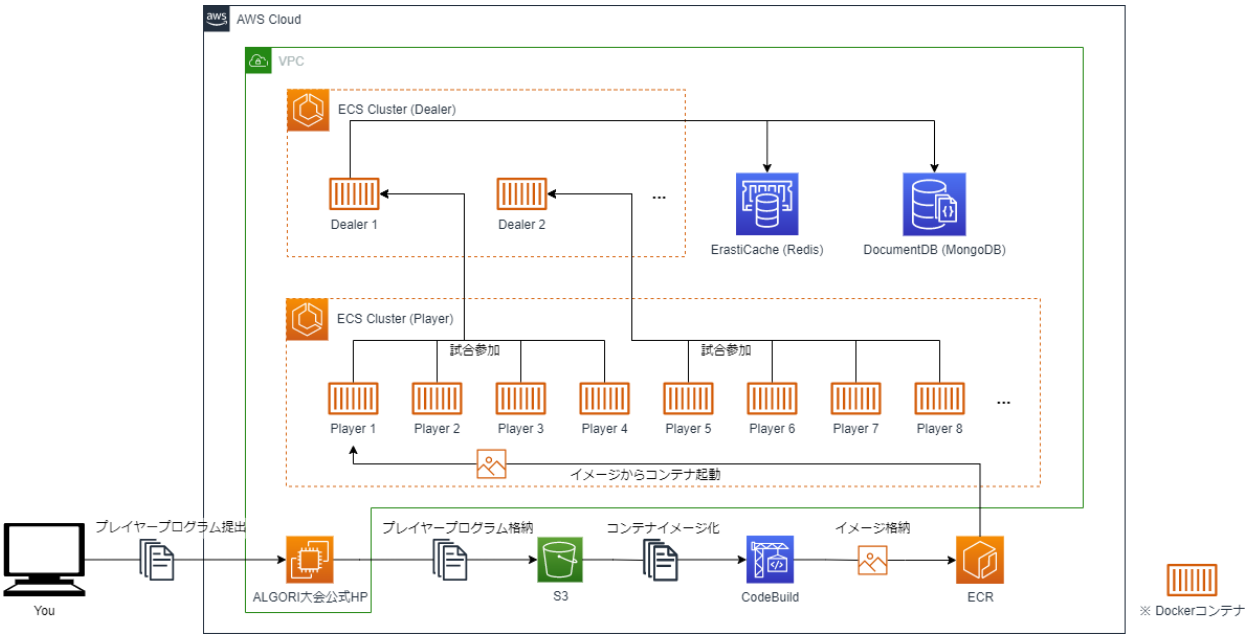
大会を開催する環境（ ALGORI システム）とディーラープログラム（以下、ディーラー）とプレイヤープログラム（以下、プレイヤー）のイメージは以下のとおりとなります。

（1）システム構成

ALGORI 大会では「 ALGORI システム」内にディーラーとプレイヤーを配置します。
ディーラーはALGORI運営側が仕様修正、または不具合解消毎にアップデートを行い、プレイヤーは大会参加者の戦略に沿ってカスタマイズを行ったプログラムとなります。
予選大会、決勝大会共に ALGORI システム内で行います。

大会環境

大会開催環境は Amazon Web Services を利用して構築しています。



※イメージ図内の説明

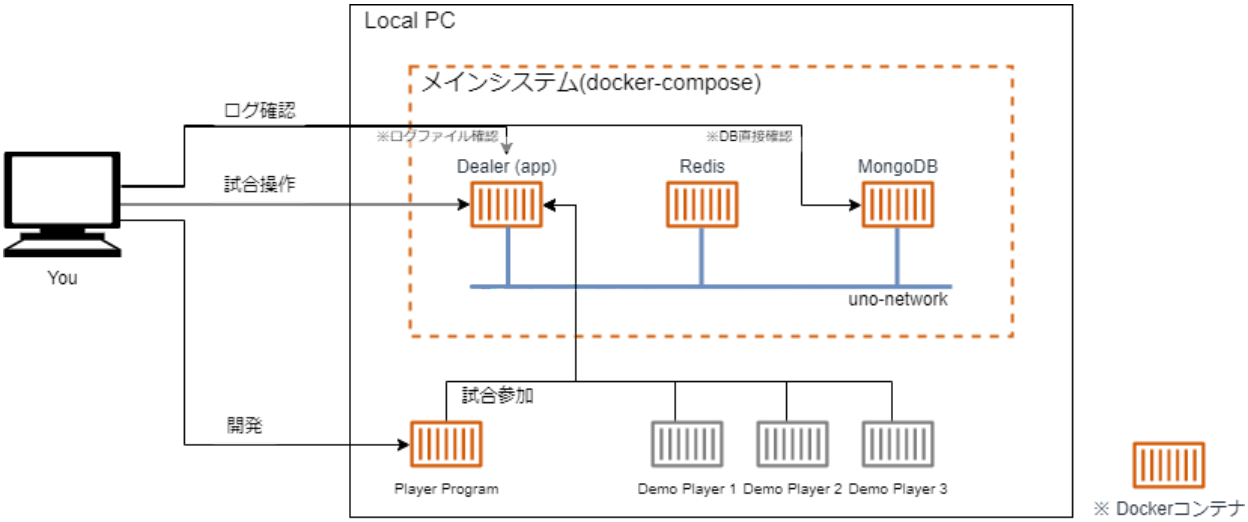
1. ディーラープログラム：「UNO」を進行および管理する。
2. プレイヤープログラム：大会参加者が作成するプログラム。

- 各リソース説明
  - **Amazon Elastic Container Service**
    ディーラープログラム、プレイヤープログラムをそれぞれ展開します。
  - **Amazon Fargate**
    上記のコンテナを管理するためのサービスです。
  - **Amazon ElastiCache**
    インメモリキャッシングサービスで、本システムでは Redis を利用しています。

Redis の詳細は後述の「6．その他機能について」を参照してください。

- **Amazon DocumentDB**
  フルマネージドドキュメントデータベースサービスで、本システムでは MongoDB を利用しています。
  MongoDB の詳細は後述の「6．その他機能について」を参照してください。

参加者のローカル環境



（2）データの通信について

ALGORI システムではリアルタイム通信の手段として WebSocket を用いております。

以下 1〜4 の内容は WebSocket における通信の流れであり、この流れに沿ってプログラムが実行されます。

1. クライアントとサーバが Socket 接続を確立
2. 受け取ったサーバがサーバサイドで処理の実行
3. サーバがクラアントに処理結果を逐次的に送信
4. 処理が終了したら Socket 接続を切断

（3）ディーラープログラム

ALGORI 大会における「UNO」 の対戦の進行と勝敗管理、および各プレイヤーが出したカード効果を制御するプログラムとなります。
主な機能としては、以下のとおりです。

1. 試合開始前の準備
   - プレイヤーの参加管理
   - 対戦回数の設定
   - 白いワイルドの設定

2. 試合中の制御

    - 設定した対戦回数が終了するまで「UNO」を進行
    - 各プレイヤーへ手札として 7 枚ずつカードを配布、余りのカードは山札とする
    - 各プレイヤーのカードを出す順番を決定
    - 山札にある一番上のカードを取りだし、場札として各プレイヤーへ周知
    - 各プレイヤーが出すカードを大会ルールに則り処理
        a. 場札へカードを出したプレイヤーが正当なプレイヤーであるか（カードを出す順番どおりであるか）
        b. 場札へ出したカードはプレイヤーが所持していたカードであるか（手札以外のところから出していないか）
        c. 場札に出したカードは誤っていないか（色違い、数字の相違、スキップ・ドロー 2 等のカード）
        d. カード効果、およびペナルティ発生有無の確認
        e. チャレンジの際の成功判定
        f. 「UNO」コール漏れの指摘時の対応
        g. スペシャルロジック発動と効果の対応

3. 試合終了後の処理

    - UNOのルールに沿った対戦終了判定
    - 対戦終了時の得点計算と得点累計

（4）プレイヤープログラム

参加者の皆様に開発していただくプログラムです。
開発するプレイヤープログラムの言語に指定はありません。ただし、後述するDocker形式での提出が必須となります。
また、開発キットにはデモプレイヤーのプログラムが含まれており、このデモプレイヤープログラムをカスタマイズして開発をしていただくことも可能です。

参加者同士が競い合うディーラープログラムを勝ち抜くため、個性的なアルゴリズムの構築を行いつつ、多種多様なカードをどの場面で使用するか（イベントとして発動させるか）などを検討し開発をお願いいたします。
プレイヤープログラムの開発〜提出までの手順は次章で説明します。

3. 「UNO」開始までの作業手順

この章では、デモプレイヤーを用いて試合を行う手順を解説します。以下の手順で作業を行ってください。
プレイヤープログラムの開発手順は次章で説明します。

1. 環境の構築
2. Dockerのインストール
3. ディーラーの起動
4. プレイヤーの起動
5. 「UNO」の開始、および終了

それぞれの作業については注意点や画面イメージを基に記載しております。

（1）環境の構築

1. 開発キットの確認
   ALGORI 公式 HP よりダウンロードした「開発キット」については以下の内容となります。

   i. 各種ドキュメント
      a. ALGORI 大会仕様書（本書）
      b. ゲームログ仕様書
   ii. ディーラープログラム
      a. ディーラープログラム
      b. 管理者ツール
      c. 開発ガイドライン
   iii. javascript 版デモプレイヤープログラム
   iv. Python 版デモプレイヤープログラム

```
# 開発キットのディレクトリ構造
.
├── demo-player_JS // javascript 版デモプレイヤープログラム
├── demo-player_python // Python 版デモプレイヤープログラム
├── document // 各種ドキュメント
└── uno-procon // ディーラープログラム
```

2. 環境構築にあたってのバージョン情報

   - 実行環境
     - Docker 4.13.x
   - データストア
     - MongoDB v5.0.x
     - Redis v7.0.x

- Socket 通信
  - Socket.io v2.4.x
- ディーラープログラム
  - Node.js v18.14 以上
  - tsc v4.3 以上
- デモプレイヤープログラム
  - Node.js v18.14 以上
  - Python v3.6 以上

※開発する言語・バージョンはこの限りではありませんが、記載しているバージョンでの開発を推奨します。
※バージョン情報に記載している内容（MongoDB 等）については後述「5．（3）補足説明」を参照してください。

（2）Docker のインストール

詳細は Docker 公式ページを参照しご自身の環境に合わせてインストールを行ってください。

（3）ディーラーの起動

1. ディーラープログラムの起動
   解凍した開発キットの uno-procon ディレクトリ内で、以下のコマンドを実行するとディーラープログラムが起動します。
   事前に docker（v4.13.x 推奨）が動作する環境を準備してください。

   - コマンドプロンプト/ターミナルの起動

     - Windows
       「Windows キー＋ R」→「プログラムを指定して実行」ウィンドウで
       「cmd」入力 →enter キー押下
     - Mac
       Finder で、「/アプリケーション/ユーティリティ」→「ターミナル」をダブルクリック

   - コマンド操作

```
# 移動 Windows
$ cd 解凍したフォルダのパス\development-kit.zip\development-kit\uno-procon
# 移動 Mac（Linux）
$ cd 解凍したフォルダのパス/development-kit/uno-procon
```

```
# ディーラープログラムのビルド
$ docker-compose build
```

```
# ディーラープログラムの起動
$ docker-compose up
```

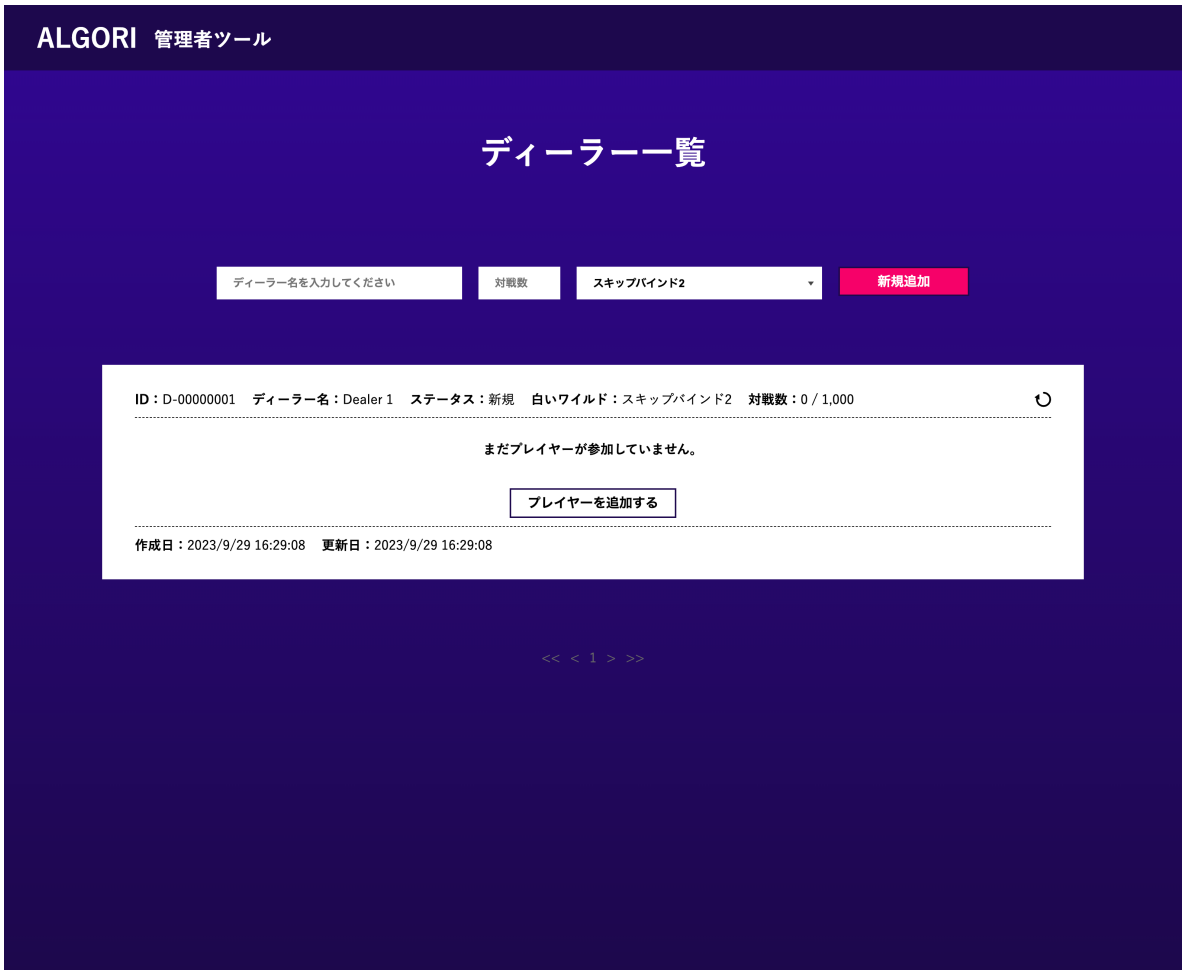
ディーラープログラムには、以下の機能が含まれます。

i. ディーラープログラム
ii. 管理者ツール
iii. 開発ガイドライン

※「管理者ツール」について詳細は「５．（１）管理者ツール」を参照してください。
※「開発ガイドライン」について詳細は「５．（２）開発ガイドライン」を参照してください。
※後述の「管理者ツール」と「開発ガイドライン」を起動させるため、ディーラープログラムの起動は忘れずに行ってください。

2. 管理者ツールによる試合の設定
ディーラープログラムを起動し、ブラウザで http://localhost:8080/api/v1/admin/web にアクセスすると管理者ツールが利用できます。
「管理者ツール」では、以下の設定を行い試合の準備を行います。
それぞれの項目毎に入力または選択後、「新規追加」ボタンをクリックすることで新しい試合が作成されます。

- ディーラー名
  試合中と同名のディーラー名で新たに試合を開始した場合、試合中のルーム名のステータスが終了となります。
  同一名称のディーラーが試合中の場合は、新しくデータを登録することはできません。
- 対戦数
  対戦数は任意の整数（但し、マイナス数字以外）で入力出来ます。
- 白いワイルドの効果
  今大会で採用される白いワイルドおよびその効果については、大会ルールを参照してください。



（4）プレイヤーの起動

1. プレイヤーの起動
   作成（更新）したプレイヤーを起動させます。
   ここでは、デモプレイヤープログラムの起動方法を説明します。
   まずは「javascript 版プレイヤープログラム」または「Python 版プレイヤープログラム」のいずれかを選択してください。

   それぞれ、解凍した開発キットに含まれる demo-player_JS または demo-player_python ディレクトリ内で操作を行います。

   javascript 版デモプレイヤー

```
# dockerイメージ作成
$ docker build -t javascript-demo-player ./
```

```
# Windows/Mac環境での実行
$ docker run javascript-demo-player "http://host.docker.internal:8080"
"Dealer 1" "Player 1"

# Linux環境での実行
$ docker run --add-host=host.docker.internal:host-gateway javascript-
demo-player "http://host.docker.internal:8080" "Dealer 1" "Player 1"
```

python 版デモプレイヤー

```
# dockerイメージ作成
$ docker build -t python-demo-player ./
```

```
# Windows/Mac環境での実行
$ docker run python-demo-player "http://host.docker.internal:8080"
"Dealer 1" "Player 1"

# Linux環境での実行
$ docker run --add-host=host.docker.internal:host-gateway python-demo-
player "http://host.docker.internal:8080" "Dealer 1" "Player 1"
```

※起動時の引数の注意点※

- "Dealer 1"：前頁「ディーラー一覧」画面内で設定した「ディーラー名」と同じディーラー名にしてください。
- "Player 1"：起動させるプレイヤー名にしてください。

2. 複数プレイヤーの起動
同じプレイヤーでもプレイヤー名を変更すれば複数のプレイヤーとして参加させることも出来ますので、参加するプレイヤーが 4 人になるようにプレイヤーを起動してください。

新しくコンソールを開き下記のコマンドを実行します。

javascript 版デモプレイヤー

```
# ディーラー名は同一の名前、プレイヤー名は異なる名前を指定します。
# 4 人それぞれ異なるプレイヤー名にして起動してください。

# Windows/Mac環境での実行
$ docker run javascript-demo-player "http://host.docker.internal:8080"
"Dealer 1" "Player 2"

# Linux環境での実行
$ docker run --add-host=host.docker.internal:host-gateway javascript-
demo-player "http://host.docker.internal:8080" "Dealer 1" "Player 2"
```

python 版デモプレイヤー

```
# ディーラー名は同一の名前、プレイヤー名は異なる名前を指定します。
# 4 人それぞれ異なるプレイヤー名にして起動してください。

# Windows/Mac環境での実行
$ docker run python-demo-player "http://host.docker.internal:8080"
"Dealer 1" "Player 2"

# Linux環境での実行
```

```
$ docker run --add-host=host.docker.internal:host-gateway python-demo-player "http://host.docker.internal:8080" "Dealer 1" "Player 2"
```

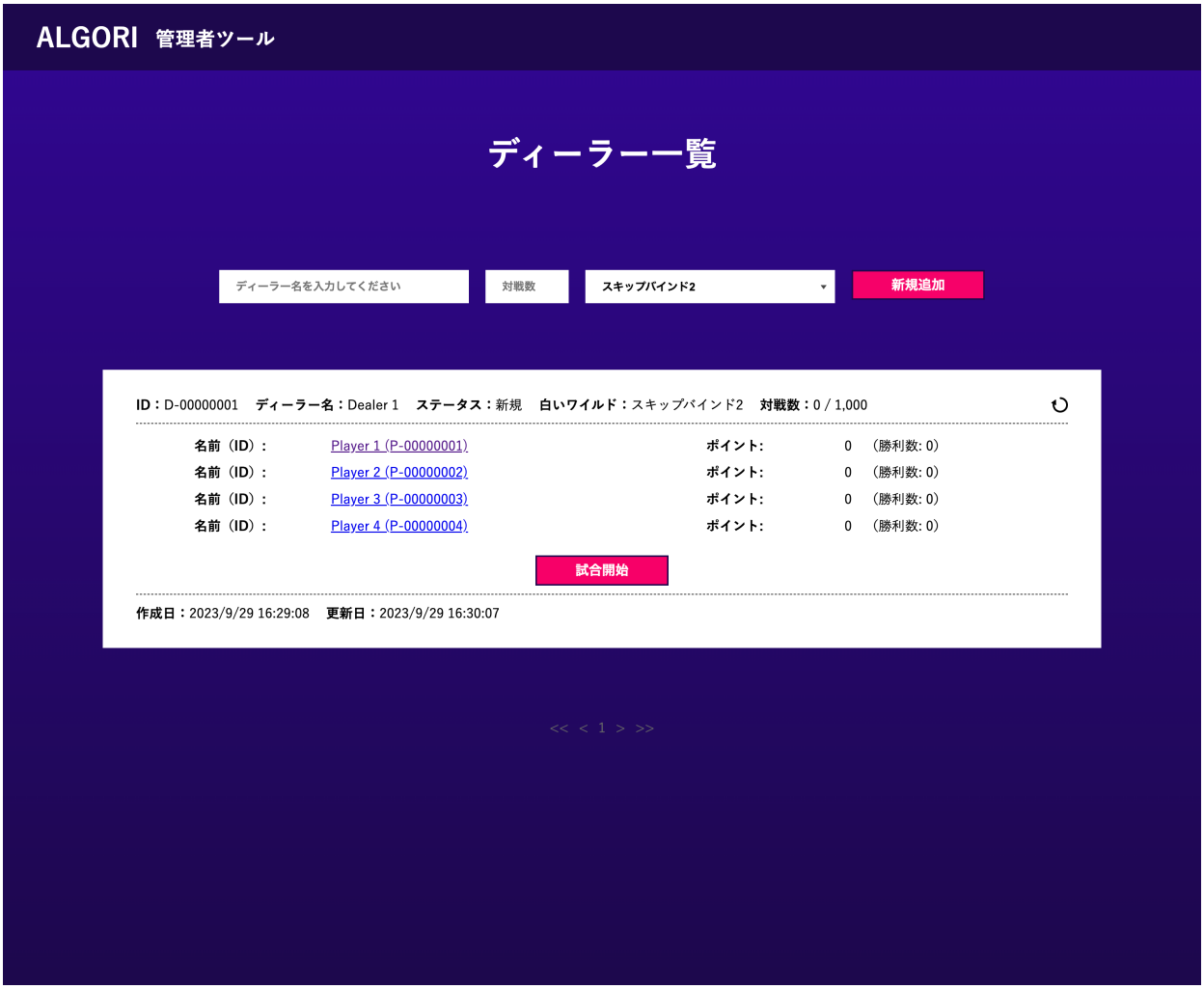
プレイヤーを追加したあとは、管理ツールをリロードしてプレイヤーが増えていること確認してください。

【補足事項】
管理ツールでディーラーを作成したあと、その試合の参加者が 4 人未満の場合は、「プレイヤーを追加する」というボタンが表示されます。
このボタンをクリックすることでデモプレイヤーを追加することができます。ただし、このデモプレイヤーのプログラムを変更することはできません。

（5）「UNO」の開始、および終了

「UNO」の開始

4 人分のプレイヤーを起動後、「管理者ツール」を再読込してください。
設定したディーラー名に対戦参加させたプレイヤーが表示されます。
「試合開始」ボタンをクリックすることで「UNO」が開始します。



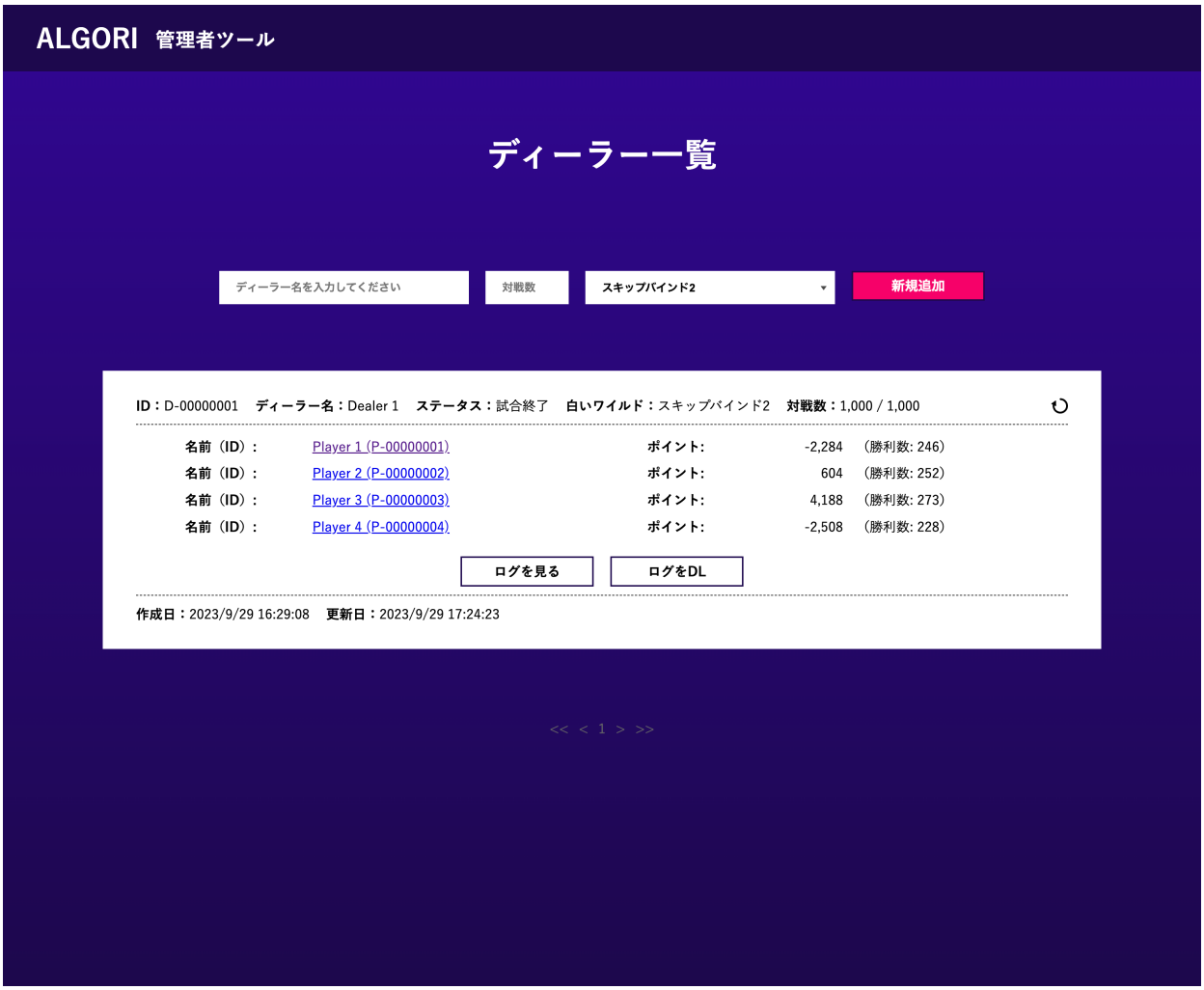
「UNO」の試合中

「試合開始」ボタンをクリックすることで「UNO」が開始します。
※このときはステータスが「試合中」と表示されます。

試合中はディーラープログラムのDockerをストップすることで「UNO」中断が可能です。**中断時の状態保存や途中からの再開は出来ないことをご承知おきください。**

「UNO」の終了

試合開始前に設定した対戦数を消化することによって「UNO」が終了します。
※このときのステータスは「試合終了」と表示されます。



【補足説明】

- ディーラープログラム
  試合終了した状態であっても、ディーラープログラムのプロセスは終了しておりません。引き続き新たに試合を行うことが可能です。
  ディーラープログラムのプロセスを終了したい場合は、ディーラープログラムのコマンドプロンプト上にて、「ctrl＋c」キーを押下することにより、終了することが出来ます。
  また、試合の途中でディーラープログラムを終了した場合は、試合に参加している各プレイヤーとの Socket 通信は切断され、各プレイヤーのプロセスは終了します。

- デモプレイヤー
  上述のとおり Socket 通信が切断されると、試合に参加しているプレイヤーのプロセスが終了するようになっています。

（6） ゲームログの確認手順

試合終了後、試合途中の経過や得点経過等を確認するにあたり以下の三つの方法があります。それぞれの確認方法について説明をしていきます。

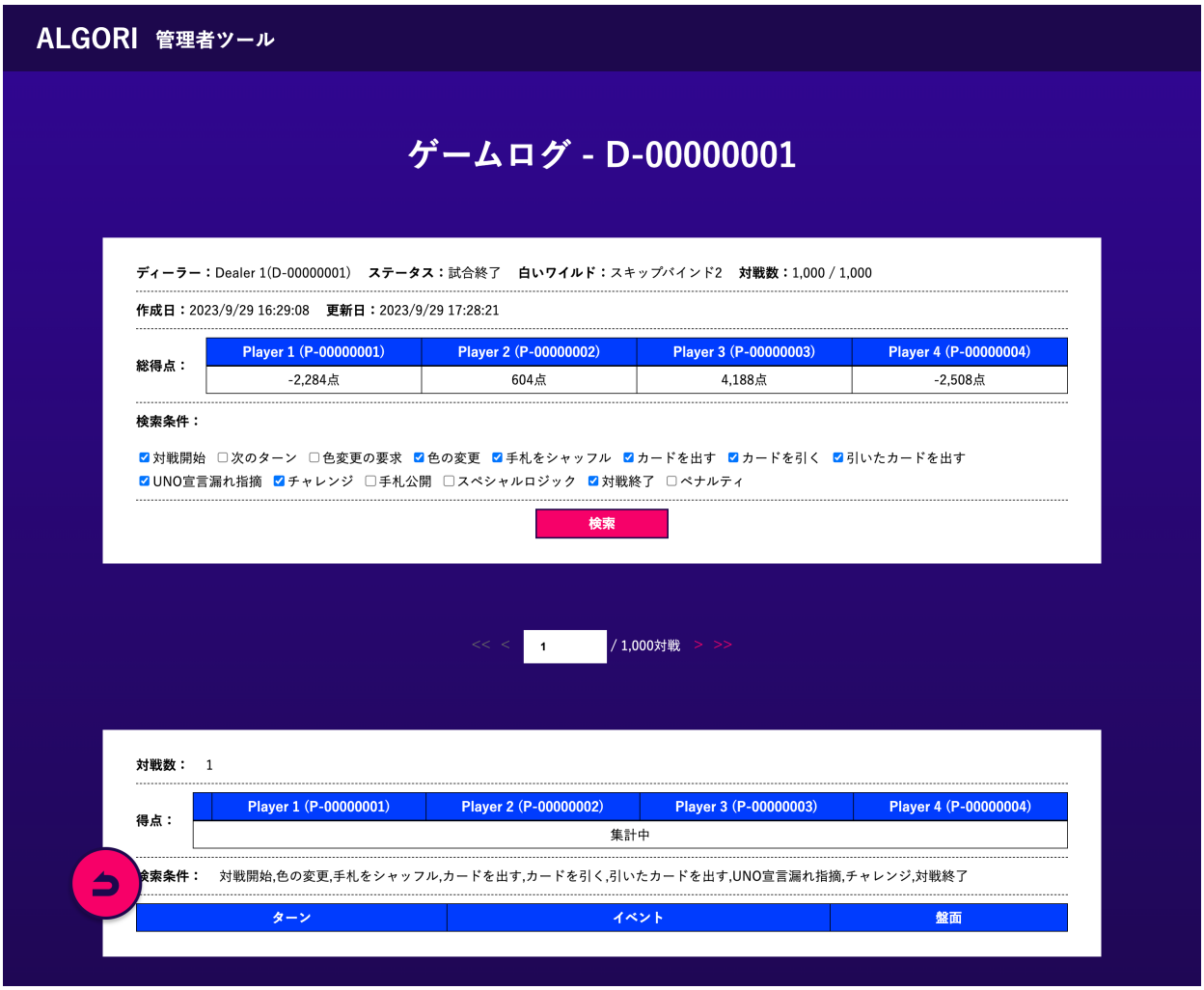
1. ゲームログ画面からの確認

2. 試合結果ログファイルからの確認
3. MongoDBのドキュメントを検索して確認

ログ情報の詳細は、別紙のゲームログ仕様書を参照してください。

ゲームログ画面からの確認

「管理者ツール」画面に表示されている試合終了後のディーラー欄内にある「ログを見る」ボタンを押下してください。
「ゲームログ」画面に遷移後、検索条件を指定して確認したい試合中の対戦数、またはスペシャルロジックが発生した対戦といった様々な条件で検索が可能です。
検索条件にチェック後、「検索」ボタンを押下してください。



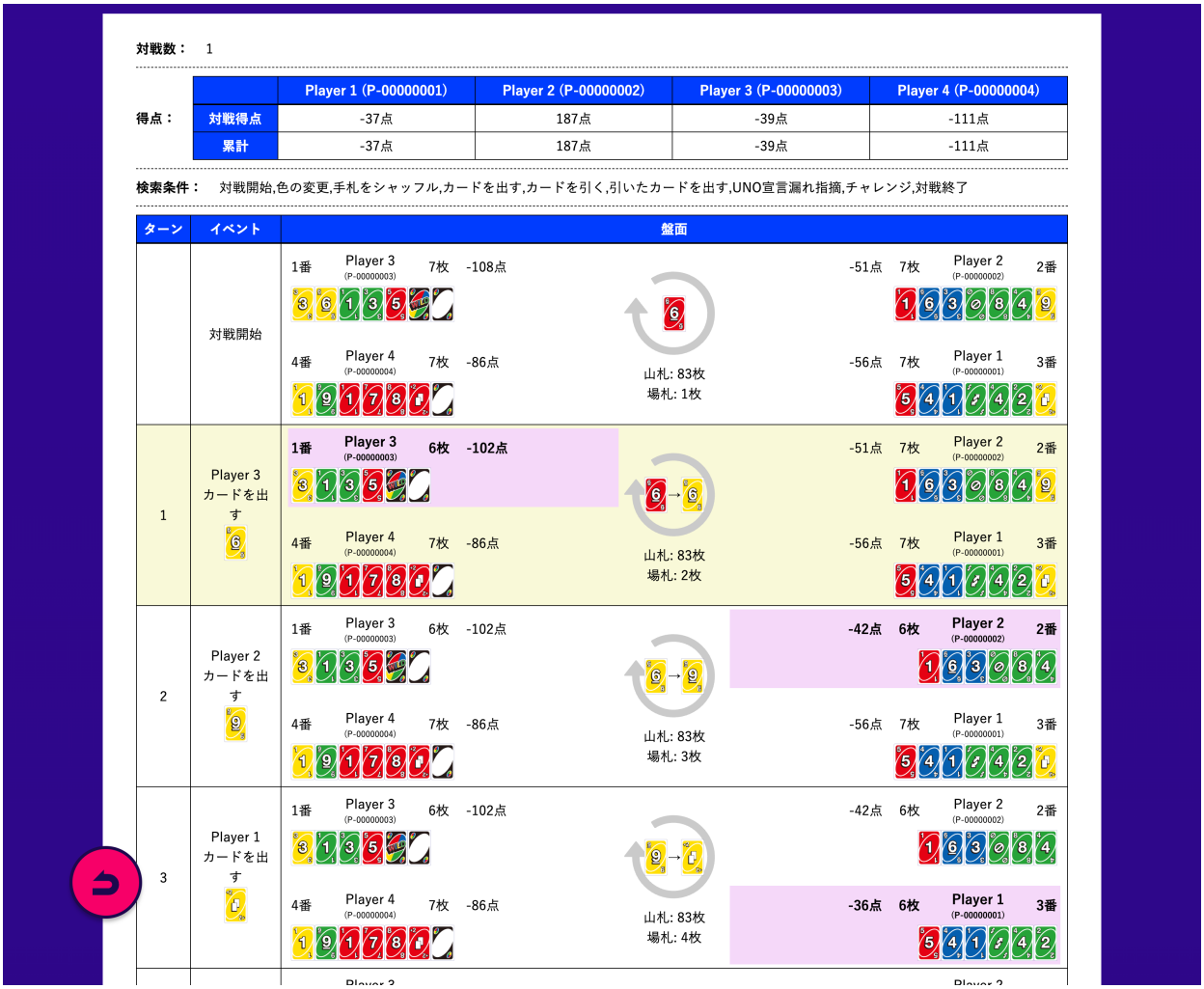
検索結果に該当する対戦中の内容が以下のように表示されます。
画面内の見方については、以下のとおりとなります。

1. ターン
   対戦数における順番を表します。
   1番目のプレイヤーが1ターン目、2番目のプレイヤーが2ターン目と表示されます。

2. イベント
   ディーラーまたはプレイヤーが起こしたイベントを表示します。
   （イベント情報については、後述「４．データ通信仕様」をご参照ください。）

3. 盤面

対戦の状況（場札の色と枚数、山札の残り枚数、各プレイヤーの手札、得点、順番、UNOコールの有無）が表示されます。



対戦数： 1

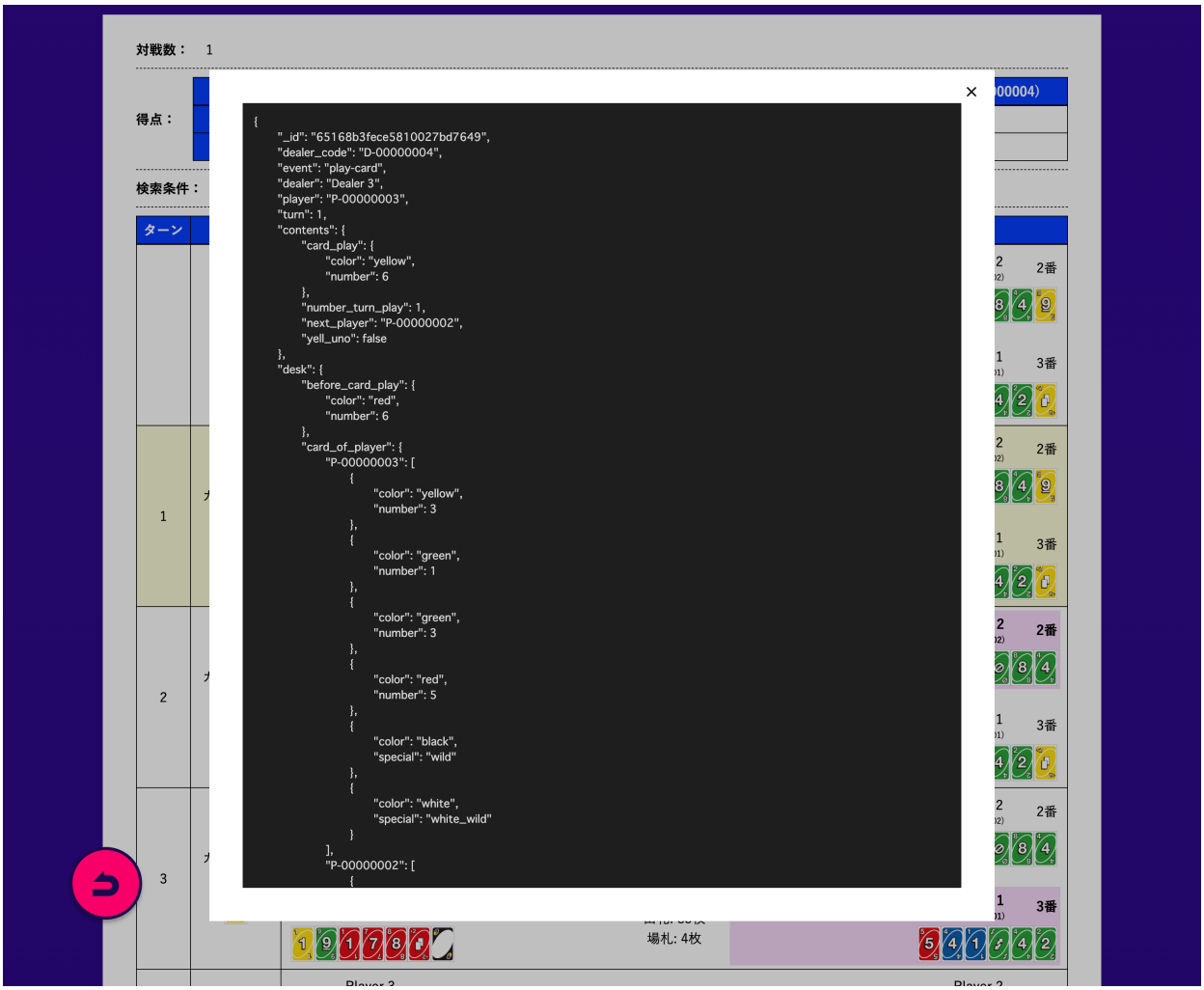
得点：

	Player 1 (P-00000001)	Player 2 (P-00000002)	Player 3 (P-00000003)	Player 4 (P-00000004)
対戦得点	-37点	187点	-39点	-111点
累計	-37点	187点	-39点	-111点

検索条件： 対戦開始,色の変更,手札をシャッフル,カードを出す,カードを引く,引いたカードを出す,UNO宣言漏れ指摘,チャレンジ,対戦終了

ターン	イベント	盤面
	対戦開始	1番 Player 3 (P-00000003) 7枚 -108点 4番 Player 4 (P-00000004) 7枚 -86点 山札: 83枚 場札: 1枚 -51点 7枚 Player 2 (P-00000002) 2番 -56点 7枚 Player 1 (P-00000001) 3番
1	Player 3 カードを出す	1番 Player 3 (P-00000003) 6枚 -102点 4番 Player 4 (P-00000004) 7枚 -86点 山札: 83枚 場札: 2枚 -51点 7枚 Player 2 (P-00000002) 2番 -56点 7枚 Player 1 (P-00000001) 3番
2	Player 2 カードを出す	1番 Player 3 (P-00000003) 6枚 -102点 4番 Player 4 (P-00000004) 7枚 -86点 山札: 83枚 場札: 3枚 -42点 6枚 Player 2 (P-00000002) 2番 -56点 7枚 Player 1 (P-00000001) 3番
3	Player 1 カードを出す	1番 Player 3 (P-00000003) 6枚 -102点 4番 Player 4 (P-00000004) 7枚 -86点 山札: 83枚 場札: 4枚 -42点 6枚 Player 2 (P-00000002) 2番 -36点 6枚 Player 1 (P-00000001) 3番

ゲームログ画面において、行をクリックすることで実際のディーラーのログを確認することが可能です。



※ゲームログ画面は対戦数ごとに表示するため、「接続」「試合終了」「切断」のログは表示されません。これらは、後述の試合結果ログファイルからの確認またはMongoDBのドキュメントを検索して確認の方法で確認してください。

試合結果ログファイルからの確認

「管理者ツール」画面に表示されている試合終了後のディーラー欄内にある「ログをDL」ボタンを押下してください。
試合の全てのログが出力されたファイルがダウンロードできるので、内容を確認することが可能です。

MongoDBのドキュメントを検索して確認

1. docker コンテナへのアクセス

```
# dockerコンテナ内にアクセス
$ docker exec -it uno-procon-mongodb-1 /bin/bash
```

docker コンテナが見つからない場合は、起動しているコンテナのコンテナ名を確認してください。

```
Error: No such container: {イメージ名}
```

```
# dockerコンテナ表示 右端のNAMESの項目がコンテナ名です。
$ docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED
STATUS          PORTS                                NAMES
2a4d87dd9fa8   uno-procon_app   "npm run start:local…"   27 minutes
ago   Up 27 minutes   0.0.0.0:8080->8080/tcp, 8081/tcp   uno-procon-
app-1
68b5f52d67fb   mongo:5.0        "docker-entrypoint.s…"   27 minutes
ago   Up 27 minutes   0.0.0.0:27017->27017/tcp           uno-procon-
mongodb-1
01447bae2b74   redis            "docker-entrypoint.s…"   27 minutes
ago   Up 27 minutes   0.0.0.0:6379->6379/tcp             uno-procon-
redis-1
```

2. mongo シェルにアクセス

```
# mongoシェルにアクセス
mongosh uno-local -u uno -p uno
```

3. ログの検索

```
# ディーラー名やプレイヤー名など必要な検索条件で検索します
uno-local> db.activities.find({ dealer: "Dealer 1" });
[
  {
    _id: ObjectId("635946ccfcb8ec0028435784"),
    dealer_code: "D-00000001",
    event: "join-room",
    dealer: "Dealer 1",
    player: "P-00000001",
    contents: {
      player_id: "P-00000001",
      player_name: "Player 1"
    },
    dateCreated: ISODate("2022-10-26T14:40:12.212Z"),
    dateUpdated: ISODate("2022-10-26T14:40:12.212Z")
  },
  {
    _id: ObjectId("635946d0fcb8ec0028435786"),
    dealer_code: "D-00000001",
    event: "join-room",
    dealer: "Dealer 1",
    player: "P-00000002",
    contents: {
      player_id: "P-00000002",
      player_name: "Player 2"
    },
    dateCreated: ISODate("2022-10-26T14:40:16.773Z"),
    dateUpdated: ISODate("2022-10-26T14:40:16.773Z")
  },
  {
    _id: ObjectId("635946edfcb8ec0028435788"),
```

```
    dealer_code: "D-00000001",
    event: "join-room",
    dealer: "Dealer 1",
    player: "P-00000003",
    contents: {
      player_id: "P-00000003",
      player_name: "Player 3"
    },
    dateCreated: ISODate("2022-10-26T14:40:45.520Z"),
    dateUpdated: ISODate("2022-10-26T14:40:45.520Z")
  },
  {
    _id: ObjectId("635946f2fcb8ec002843578a"),
    dealer_code: "D-00000001",
    event: "join-room",
    dealer: "Dealer 1",
    player: "P-00000004",
    contents: {
      player_id: "P-00000004",
      player_name: "Player 4"
    },
    dateCreated: ISODate("2022-10-26T14:40:50.739Z"),
    dateUpdated: ISODate("2022-10-26T14:40:50.739Z")
  }
]
```

4. Mongo シェルを終了する

```
# Mongoシェルを終了する
uno-local> exit
```

5. コンテナから抜ける

```
# コンテナから抜ける
exit
```

４．プレイヤープログラムの開発～提出までの手順

この章では実際に皆様に行っていただくプレイヤープログラムの開発について紹介します。
※イベントについては後述「５．データ通信仕様について」を参照してください。

（１）プレイヤープログラムの作成と更新

「開発キット」内に同封されている「デモプレイヤープログラム」を参考にして、参加される皆様の手によって"個性的な「プレイヤー」"の作成をお願いいたします。
同封されている開発ガイドラインに沿ってプレイヤープログラムの開発を行うことも可能です。
開発キットに同封されているのは「javascript版」と「Python版」のデモプレイヤープログラムとなりますが、「websocket」が使用可能な言語であれば、どのような言語でもプレイヤープログラムとして作成は可能となります。

（２）スペシャルロジックについて

スペシャルロジックとは、本コンテストの参加者がプログラムを構築する際に考案するロジックのうち、特に力を入れたものに名前を付けて表現できるものです。

スペシャルロジック内容は任意で設定が可能であり、どのタイミングで「スペシャルロジック」を発動させるかはプレイヤーを作成する皆様次第となります。

※実装および発動回数といった仕様については、大会ルールに記載しております。
弊社「ALGORI 大会」公式 HP の「応募要項」画面内「大会ルール」よりダウンロード後、「スペシャルロジックについて」欄をご参照ください。

（３）不明点や質問の問い合わせ

弊社「ALGORI大会」公式HPの「お問い合わせ」画面より、必須項目をご入力の上、ALGORI大会運営事務局へのお問い合わせをお願いします。

- お名前：エントリー時にご登録されたお名前でご入力をお願いします。
- メールアドレス：お名前と同様にエントリー時にご入力されたメールアドレスのご入力をお願いします。
- お問い合わせ内容：大会運営に関わるお問い合わせ、または開発に関わるご相談などをお問合せしたい内容のご入力をお願いします。
  ※お問い合わせ内容に対する返信につきましては、上述のメールアドレス宛への返信となりますのでご了承ください。

（４）プレイヤープログラムの提出について

1. プレイヤーのアップロードとは
   開発したプレイヤープログラムについては、弊社「ALGORI 大会公式 HP」にログイン

後、「参加者マイページ」内にある「プログラムの提出」からアップロードを行ってください。
アップロードを行わないと、ご自身で作成（更新）されたプレイヤーの ALGORI 大会へのエントリーが完了となりませんのでご注意ください。

2. アップロードまでの手順

   i. プレイヤーの稼働確認

   - アップロード提出前に必ず同梱されている「開発ガイドライン」に従って、作成したプレイヤーの動作確認を実施してください。
     ※尚、「開発ガイドライン」については後述「5．（2）開発ガイドライン」を参照してください。

   ii. プレイヤーのコンテナ化
   ※提出するために必須な作業となりますので忘れずに行ってください。

      a. Dockerfile の作成
      以下は javascript (Node.js) でアプリケーションを作成する例です。
      アプリケーションと同じディレクトリに Dockerfile というファイルを作成します。

      - プレイヤープログラムを javascript で作成した場合

```
# プレイヤーアプリケーションの起動方法が、下記のコマンドである場合を想定しています。
$ node player.js "http://localhost:8080/" "Dealer 1" "Player 1"
```

      Dockerfile の内容を以下のように記載します。
      必要に応じて CMD の部分の記載を修正してください。

```
FROM node:16
WORKDIR /app

COPY . .

RUN npm install

ENTRYPOINT [ "node", "player.js"]

CMD [ "http://localhost:8080/", "Dealer 1", "Player 1" ]
```

      - プレイヤープログラムを python で作成した場合

```
# プレイヤーアプリケーションの起動方法が、下記のコマンドである場合を想定しています。
```

```
$ python3 player.py "http://localhost:8080/" "Dealer 1"
"Player 1"
```

Dockerfile の内容を以下のように記載します。
必要に応じて CMD の部分の記載を修正してください。

```
FROM python:3.8.6
WORKDIR /app

COPY . .

RUN pip install --upgrade pip
RUN pip install -r requirements.txt

ENTRYPOINT [ "python", "demo-player.py"]

CMD [ "http://localhost:8080/", "Dealer 1", "Player 1" ]
```

※接続先のホスト名とディーラー名は試合ごとに異なるため、コマンドライン引数で指定できるように作成する必要があります。
※プレイヤー名は **他の参加者のプレイヤー名と同じものを使用することができません。** プレイヤー名はエントリー時に登録したチーム名が使用されますので、オリジナリティのあるチーム名での登録をお願いします。

b. Docker のビルド

```
# ビルド
$ docker build -t uno-procon-player .
```

c. ビルド後の実行確認

```
# 実行
$ docker run uno-procon-player "http://localhost:8080/"
"Dealer 1", "Player 1"
```

d. 提出前確認

- 上記手順 c でディーラープログラムと接続を行い、ローカル環境での対戦を実行および対戦が正常終了すること
- ファイルのサイズは、zip 形式で 1.0GB 以内とすること

e. 提出

- 参加者マイページ内からアップロードができます。
- アップロードの手順については、参加者マイページをご確認ください。

f. 注意点

- 余計なファイルがないこと
- Dockerfile がフォルダの直下にあること

3. アップロード後の動作確認とセキュリティ対策について
   アップロードしていただいたプレイヤーについては ALGORI 大会運営事務局でも動作確認とセキュリティチェックを実施します。
   禁止事項となる以下の事項に該当するプログラムは ALGORI 大会参加対象から除外します。

   i. 大会運営に支障をきたす目的のプログラム
   ii. 不正を狙っていると思われるプログラム
   iii. プレイヤーからディーラー以外へのアクセスを禁止（外部 API 等へのアクセス）

   ※上記以外で、動作確認中に全く動作しないプログラムについては ALGORI 大会事務局にて協議の上、除外（失格）とさせていただく可能性があることをご承知おきください。

4. Scoket 通信について
   ディーラーと Socket 通信を行えるように実装してください。
   ※通信仕様の詳細説明は後述の「5．データ通信仕様について」と「6．その他機能について」を参照してください。

5. データ通信仕様について

ALGORI 大会でのデータ通信においては、二つの定義があります。

- 「データタイプ」：プレイヤーがカードを出す・カードを取る・チャレンジをする、といった様々な行動を起こす際、手札にどのようなカードを所持しているか、場札にどのようなカードが出されているか、前の順番のプレイヤーがどのようなカードを出したか判断を行います。
- 「イベント」：プレイヤーが出したカード効果によって任意に発生させる「イベント」の管理、またはディーラーで管理する「イベント」の制御を行います。

（1）データタイプ

プレイヤーは、以下のデータタイプに従ってディーラーと通信を行い、「UNO」を進行します。

1. カードの色の種類

   プログラムで使用する既定値は以下です。

   ```
   export enum Color {
     RED = "red",
     YELLOW = "yellow",
     GREEN = "green",
     BLUE = "blue",
     BLACK = "black",
     WHITE = "white",
   }
   ```

2. 記号カードの種類

   プログラムで使用する既定値は以下です。
   各カードの効果については大会ルールブック を参照してください。

   ```
   export enum Special {
     SKIP = "skip",
     REVERSE = "reverse",
     DRAW_2 = "draw_2",
     WILD = "wild",
     WILD_DRAW_4 = "wild_draw_4",
     WILD_SHUFFLE = "wild_shuffle",
     WHITE_WILD = "white_wild",
   }
   ```

3. 白いワイルドと効果

   プログラムで使用する既定値は以下です。
   白いワイルドの効果と今年度に採用されてい る種類は大会ルールブックを参照してくだ

さい。

```
export enum WhiteWild {
  BIND_2 = "bind_2",
  SKIP_BIND_2 = 'skip_bind_2',
}
```

4. カードのフォーマット
   数字（0 〜 9）とカードの色の種類とカードの記号の種類の組み合わせを表します。

```
export interface Card {
  color: Color; // red, yellow, green, blue, black, white
  number?: number; // 0, 1, 2, ... 9
  special?: Special; // skip, reverse, draw_2, wild, wild_draw_4,
wild_shuffle, white_wild
}
```

   ※数字のカードには special のプロパティがありません。
   ※記号のカードには number のプロパティがありません。

   カードデータのサンプルは次項を参照してください。

5. その他：カードデータのサンプル

```
// Example Card Red 9
{
  "number": 9,
  "color": "red"
}
```

```
//Example Card Yellow Skip
{
  "color": "yellow",
  "special": "skip"
}
```

```
//Example Card Wild Draw 4
{
  "color": "black",
  "special": "wild_draw_4"
}
```

6. 山札からカードを引かされる理由

   - ドロー 2
     前のプレイヤーがドロー 2 を場札に出したため
   - ワイルドドロー 4
     前のプレイヤーがワイルドドロー 4 を場札に出したため

- バインド 2
  前のプレイヤーが白いワイルド（バインド 2）を場札に出したため
- なし
  カードを引かされる理由はなし

```
enum DrawReason {
  DRAW_2 = "draw_2",
  WILD_DRAW_4 = "wild_draw_4",
  BIND_2 = "bind_2",
  SKIP_BIND_2 = 'skip_bind_2',
  NOTHING = "nothing",
}
```

（2）イベント一覧

プレイヤーが任意に発生させるイベント（カード効果）、またはディーラーで管理するイベントについては以下のとおりとなります。

1. プレイヤーが任意に発生させるイベント
   - `join-room` : ゲームへの参加
   - `play-card` : 場札にカードを出す
   - `draw-card` : 山札からカードを引く
   - `play-draw-card` : 山札から引いたカードを場札に出す
   - `challenge` : チャレンジ
   - `pointed-not-say-uno` : UNO コールを忘れていることを指摘
   - `special-logic` : スペシャルロジック発動
2. ディーラーが管理するイベント
   - `receiver-card` : カードを手札に追加
   - `first-player` : 対戦開始
   - `color-of-wild` : 場札の色指定を要求
   - `update-color` : 場札の色を更新
   - `shuffle-wild` : 手札のカードをシャッフル
   - `next-player` : 次の順番のプレイヤーを通知
   - `public-card` : 手札の公開
   - `finish-turn` : 対戦終了の通知
   - `finish-game` : ゲーム終了の通知
   - `penalty` : ゲーム終了の通知

（3）イベントの紹介（プレイヤーが任意に発生させる）

1. `join-room`

   - 機能：ゲームへの参加
     ゲームに参加します。
     データ送信の通信に対して返却されるコールバックデータから自分のプレイヤーコード を取得します。
     以後、ご自身のプログラムで自分のプレイヤーコード を判定する場合に使用します。

   - ディーラーからの通知範囲：全クライアント

   - タイミング：ゲーム開始前

   - 通信データ

```
// Emit
{
  "room_name": "room AAA", // 参加ルーム名 - string
  "player": "Player 1" // プレイヤー名 - string
}
```

```
// callback
{
  "game_id": "P-00000001", // ゲームID - string
  "room_name": "room AAA", // 参加ルーム名 - string
  "player": "Player 1", // 新規参加のプレイヤー名 - string
  "your_id": "P-00000001", // プレイヤーコード - string
  "total_turn": 1000, // 総対戦数 - number
  "white_wild": "bind_2" // 白いワイルドの効果 - WhiteWild
}
```

```
// On
{
  "room_name": "room AAA", // 参加ルーム名 - string
  "player": "Player 1" // 新規参加のプレイヤー名 - string
}
```

   - エラー

メッセージ	エラー内容	ペナルティ対象
Room name is required.	リクエストデータに room_name プロパティがない	-

メッセージ	エラー内容	ペナルティ対象
Player name is required.	リクエストデータに player プロパティがない	-
Player name too long.	プレイヤー名が 20 文字を超えている	-
Dealer not found.	指定したディーラー名が見つからない	-
Status dealer invalid.	指定したディーラーの試合が終了している	-
Player name duplicate.	プレイヤー名が重複している	-
Dealer max player. You can not join room.	既に4人参加している試合には参加できない	-

2. `play-card`

   - 機能：場札にカードを出す
     場札に出されたカードが数字カードの時は、次のプレイヤーに順番が回ります。
     場札に出されたカードが記号カードの時は適切な効果を与え、次のプレイヤーに順番が回ります。
     場札に出されたカードが無効なカードだった場合、カードを出したプレイヤーにペナルティが与えられます。
   - タイミング：ゲーム対戦中
   - ディーラーからの通知範囲：全クライアント
   - その他
     ※ color_of_wild プロパティは false ワイルドまたはワイルドドロー 4 を出すときに必要となる要素です。
   - 通信データ

```
// Emit
{
  "card_play": {
    "number": 9,
    "color": "red"
  }, // 場札に出すカード - Card
  "yell_uno": true, // UNO宣言するか
  "color_of_wild": "blue" // 変更する色
}
```

```
// On
{
  "player": "P-00000002", // カードを出したプレイヤーコード - string
  "card_play": {
    "number": 9,
    "color": "red"
  }, // 場札に出されたカード - Card
  "yell_uno": true, // UNO宣言したか
  "color_of_wild": "blue" // 変更した色
}
```

- エラー

メッセージ	エラー内容	ペナルティ対象
Number of socket client join dealer lower two.	接続しているクライアントが足りない	-
Interrupts are restricted.	割り込み処理が禁止されている	-
Next player invalid.	自分の順番ではない	○
Can not play card.	カードを出すことができない	○
Card play is required.	リクエストデータに card_play プロパティがない	○
Param card play invalid.	リクエストデータに card_play.number および card_play.special がない	○
Special card play invalid.	リクエストデータの card_play.special の値が既定値と異なる	○
Color card play invalid.	リクエストデータの card_play.color の値が既定値と異なる	○
Number card play invalid.	リクエストデータの card_play.number の値が既定値と異なる	○
Yell uno is required.	リクエストデータに yell_uno プロパティがない、または、null である	○
Param yell uno invalie.	リクエストデータの yell_uno が boolean ではない	○

メッセージ	エラー内容	ペナルティ対象
Color wild is required.	リクエストデータに color_of_wild がない	○
Color wild invalid.	リクエストデータの color_of_wild が boolean ではない	○
Card play not exist of player.	所持していないカードを出した	○
Card play invalid with card before.	場に出すことができないカードを出した	○
Can not say uno and play card.	カードを出すことができない、または、手札の所持数が 2 枚のときではない	○

3. `draw-card`
    - 機能：山札からカードを引く
      山札からカードを引きます。
      ※このイベントは山札からカードを引いたことを全プレイヤーに通知するイベントです。引いたカードは `receiver-card` イベントにて、自分にだけ通知されます。
    - タイミング：ゲーム対戦中
    - ディーラーからの通知範囲：全クライアント
    - その他
        - ワイルドドロー 4 が場札に出された時、次のプレイヤーは山札から 4 枚のカードを引き、次のプレイヤーに順番が回る。
        - ドロー 2 が場札に出された時、次のプレイヤーは山札から 2 枚のカードを引き、次のプレイヤーに順番が回る。
        - 白いワイルド（バインド 2）が場札に出された時、次のプレイヤーは山札から 1 枚のカードを引き、次のプレイヤーに順番が回る。
        - 手札に有効なカードが無い時に、山札から 1 枚カードを引く。
          can_play_draw_card が false のときは、次のプレイヤーに順番が回る。
    - 通信データ

```
// Emit
{}
```

```
// callback
{
  "player": "P-00000002", // カードを引いたプレイヤーコード - string
  "is_draw": true, // 山札からカードを引いたか - boolean
  "can_play_draw_card": true, // 手札からカードを場札に出すことができるか
- boolean
  "draw_card": [
    {
      "number": 9,
      "color": "red"
    }
  ] // 山札から引いたカードリスト - Card[]
}
```

```
// On
{
  "player": "P-00000002", // カードを引いたプレイヤーコード - string
  "is_draw": true // 山札からカードを引いたか - boolean
}
```

- 通信データ内の追加説明
  is_draw: 手札の上限枚数以上である時など、山札からカードを引くことを強制されていない時に、山札からカードを引いた場合、この項目が false になります。

  can_play_draw_card: この項目が false の時、プレイヤーは引いたカードを出すことができず、次のプレイヤーに順番が移ります。
  ドロー 2 やワイルド ドロー 4、白いワイルド（バインド 2）が場札に出された時など、カードの効果を受けたことで山札からカードを引く場合に false になります。
  この項目が true の時は、`play-draw-card` イベントを呼び出します。

- エラー

メッセージ	エラー内容	ペナルティ対象
Number of socket client join dealer lower two.	接続しているクライアントが足りない	-
Interrupts are restricted.	割り込み処理が禁止されている	-
Next player invalid.	自分の順番ではない	○

4. `play-draw-card`

- 機能：山札から引いたカードを場札に出すかを決める
  イベント「`draw-card`」 イベントで引いたカードを場札に出すか、または場札に出さずターンを終えます。

※ is_play_card プロパティを false にすることでカードを出さない選択も可能です。
※ color_of_wild プロパティは false ワイルドまたはワイルドドロー 4 を出すときに必要となる要素です。

- ディーラーからの通知範囲：全クライアント
- タイミング：ゲーム対戦中
- 通信データ

```
// Emit
{
  "is_play_card": true, // 直前で引いたカードを場札に出すか - boolean
  "yell_uno": true,
  "color_of_wild": "blue" // 変更する色
}
```

```
// On
{
  "player": "P-00000002", // プレイヤーコード - string
  "is_play_card": true, // 直前で引いたカードが場札に出されたか - boolean
  "card_play": {
    "number": 9,
    "color": "red"
  }, // 場札に出されたカード - Card
  "yell_uno": true,
  "color_of_wild": "blue" // 変更した色
}
```

- エラー

メッセージ	エラー内容	ペナルティ対象
Number of socket client join dealer lower two.	接続しているクライアントが足りない	-
Interrupts are restricted.	割り込み処理が禁止されている	-
Next player invalid.	自分の順番ではない	○
Can not play draw card.	引いたカードが場に出せるカードではない	○

メッセージ	エラー内容	ペナルティ対象
Is play card is required.	リクエストデータに is_play_card プロパティがない、または、null である	○
Param is play card invalie.	リクエストデータの is_play_card が boolean ではない	○
Yell uno is required.	リクエストデータに yell_uno プロパティがない、または、null である	○
Param yell uno invalie.	リクエストデータの yell_uno が boolean ではない	○
Color wild is required.	リクエストデータに color_of_wild がない	○
Color wild invalid.	リクエストデータの color_of_wild が boolean ではない	○
Card play not exist of player.	所持していないカードを出した	○
Card play invalid with card before.	場に出すことができないカードを出した	○
Can not say uno and play card.	カードを出すことができない、または、手札の所持数が 2 枚のときではない	○

5. `challenge`

   - 機能：チャレンジ
     チャレンジを行った場合、ワイルドドロー 4 を出したプレイヤーの手札をディーラーがチェックし、チャレンジの成否を判断します。
     ワイルドドロー 4 を出し他プレイヤーの手札は、チャレンジしたプレイヤーにのみ `public-card` イベントで通知されます。
     チャレンジを行わなかった場合は、次のプレイヤーが通常通りプレイを行います。
     ※このイベントを実行できるのは、ワイルドドロー 4 を出したプレイヤーの次に順番が回ってくるプレイヤーです。
     ※ is_challenge プロパティを false にすることでチャンレンジを行わない選択も可能です。

   - ディーラーからの通知範囲：全クライアント

   - タイミング：ゲーム対戦中

- 通信データ

```
// Emit
{
  "is_challenge": true // チャレンジを行うか - boolean
}
```

```
// On
{
  "challenger": "P-00000002", // チャレンジしたプレイヤーコード - string
  "target": "P-00000001", // ワイルドドロー4を出したプレイヤーコード - string
  "is_challenge": true, // チャレンジを行ったか - boolean
  "is_challenge_success": true // チャレンジの成否- boolean | null
}
```

- エラー

メッセージ	エラー内容	ペナルティ対象
Number of socket client join dealer lower two.	接続しているクライアントが足りない	-
Interrupts are restricted.	割り込み処理が禁止されている	-
Next player invalid.	自分の順番ではない	○
Can not play card.	カードを出すことができない	○
Is challenge is required.	リクエストデータに is_challenge プロパティがない、または、null である	○
Param is challenge invalie.	リクエストデータに is_challenge が boolean ではない	○
Can not challenge.	チャレンジすることができない	○

- 通信データ内の追加説明
  is_challenge_success : is_challenge が false の場合、この項目はありません。

6. `pointed-not-say-uno`

- 機能：UNO コールを忘れていることを指摘

- ディーラーからの通知範囲：全クライアント
- タイミング：ゲーム対戦中
- 通信データ

```
// Emit
{
  "target": "P-00000002" // 指摘したいプレイヤーコード - string
}
```

```
// On
{
  "pointer": "P-00000001", // 指摘したプレイヤーコード - string
  "target": "P-00000002", // 指摘されたプレイヤーコード - string
  "have_say_uno": true // UNO 宣言があったか - boolean この項目が true のときはペナルティなし
}
```

- エラー

メッセージ	エラー内容	ペナルティ対象
Number of socket client join dealer lower two.	接続しているクライアントが足りない	-
Interrupts are restricted.	割り込み処理が禁止されている	-
Already penalized.	他のプレイヤーによって既に指摘されている	-
Next player invalid.	参加しているプレイヤーの ID ではない	○
Can not pointed not say uno.	指定したプレイヤーの手札所持数が 1 枚ではない	○
Out of target.	現在指摘できる対象のプレイヤーではない	-
Did not say uno.	UNO 宣言をしていない	○

※UNO 宣言をしていないときのペナルティにおいては、指摘された人がペナルティを受ける

7. `special-logic`

- 機能：スペシャルロジック発動
  プレイヤーが実行したスペシャルロジックを受け取る機能です。
- ディーラーからの通知範囲：なし
- タイミング：ゲーム対戦中
- 通信データ

```
// Emit
{
  "title": "スペシャルロジック名" // スペシャルロジック名 – string
}
```

```
// On
// なし
```

- エラー

メッセージ	エラー内容	ペナルティ対象
Number of socket client join dealer lower two.	接続しているクライアントが足りない	-
Interrupts are restricted.	割り込み処理が禁止されている	-
Param title invalid.	リクエストデータに title プロパティがない	○
Special logic title too long.	title が 32 文字を超えている	○

（4）イベントの紹介（ディーラーが管理）

1. `receiver-card`
   - 機能：カードを手札に追加
     新しく配布されたカードを手札に加えます。
   - タイミング：対戦開始前及び対戦中
     - 全プレイヤーに対し、対戦開始時に 7 枚配布される時。
     - 直前のプレイヤーがドロー 2 を出した後の `draw-card` が呼ばれ 2 枚配布される時。
     - 直前のプレイヤーがワイルドドロー 4 を出した後の `draw-card` が呼ばれ 4 枚配布される時。

- 直前のプレイヤーが白いワイルド（バインド 2）を出した後の `draw-card` が呼ばれた時。
- プレイヤーが無効なカードを出したときのペナルティとして 2 枚配布される時。
- チャレンジが成功した場合、ワイルドドロー 4 を出したプレイヤーは合計で 5 枚（場札に出したワイルドドロー 4 およびペナルティの 4 枚）のカードを配布される時。
- チャレンジが失敗した場合、チャレンジを行ったプレイヤーは合計 6 枚（ワイルドドロー 4 の効果による 4 枚およびチャレンジ失敗のペナルティの 2 枚）のカードを配布される時。
- UNO 宣言を忘れていることを指摘されたプレイヤーが 2 枚配布される時。

◦ 通信データ

```
// On
{
  "cards_receive": [
    {
      "number": 9,
      "color": "red"
    },
    {
      "number": 8,
      "color": "yellow"
    },
    {
      "special": "wild_draw_4",
      "color": "black"
    }
  ], // 配布されたカードリスト - Card[]
  "is_penalty": false // ペナルティとして配布されたか - boolean
}
```

```
// Emit
// なし
```

2. `first-player`

◦ 機能：対戦開始
すべてのプレイヤーに、対戦開始を通知します。

◦ ディーラーからの通知範囲：全クライアント

◦ タイミング：ゲーム対戦開始前

◦ 通信データ

```
// On
{
  "first_player": "P-00000001", // 最初の順番のプレイヤーコード - string
  "first_card": {
    "color": "yellow",
    "number": 8
  }, // 山札から引いた最初のカード - Card
  "play_order": ["P-00000001", "P-00000002", "P-00000003", "P-00000004"] // 順番 - string[]
}
```

```
// Emit
// なし
```

3. `color-of-wild`
   - 場札の色指定を要求
     シャッフルワイルドが場札に出された時に色の変更を行います。
     または、対戦開始時のカードがワイルドであった時に最初のプレイヤーが色の変更を行います。
   - ディーラーからの通知範囲：特定のクライアント
   - タイミング：ゲーム対戦中
     - 対戦開始時のカードがワイルドだった時
     - シャッフルワイルドが場札に出された時
   - 通信データ

```
// On
{}
```

```
// Emit
{
  "color_of_wild": "red" // 指定する色 - Color
}
```

4. `update-color`
   - 機能：場札の色を更新
     `color-of-wild`イベントにより、場札の色が指定されたことを通知します。
   - ディーラーからの通知範囲：全クライアント
   - タイミング：ゲーム対戦中

- 対戦開始時のカードがワイルドであり、最初のプレイヤーが`color-or-wild`を行った時
- シャッフルワイルドを出したプレイヤーが`color-or-wild`を行った時

- 通信データ

```
// On
{
  "color": "red" // 場札の色 - Color
}
```

```
// Emit
{}
```

5. `shuffle-wild`

- 機能：手札のカードをシャッフル
  全プレイヤーの手札を集め、シャッフルワイルドを出した次の順番の人から順に１枚ずつ配布されます。
- ディーラーからの通知範囲：特定のクライアント
- タイミング：ゲーム対戦中
  - シャッフルワイルドが場札に出された時
- 通信データ

```
// On
{
  "cards_receive": [
    {
      "number": 9,
      "color": "red"
    },
    {
      "number": 8,
      "color": "yellow"
    },
    {
      "special": "wild_draw_4",
      "color": "black"
    }
  ], // 配布されたカードリスト - Card[]
  "number_card_of_player": {
    "P-00000001": 4, // プレイヤー 1 の手札枚数 - number
    "P-00000002": 5, // プレイヤー 2 の手札枚数 - number
    "P-00000003": 4, // プレイヤー 3 の手札枚数 - number
    "P-00000004": 4 // プレイヤー 4 の手札枚数 - number
  }
}
```

```

```

```
// Emit
// なし
```

6. `next-player`

   - 機能：次の順番のプレイヤーを通知
     自分の順番のとき、ディーラーから通知されます。
     このイベントを受け取った時、プレイヤーは手札からカードを出したり、山札からカードを引いたりします。
   - ディーラーからの通知範囲：特定のクライアント
   - タイミング：ゲーム対戦中
     - 制限時間以内にイベントを行わなかった時
     - 対戦開始時の最初のカードがワイルドだった場合に最初のプレイヤーが色の指定を制限時間以内に宣言しなかった時
     - シャッフルワイルドにより場札の色の変更を制限時間以内に行わなかった時（`color-of-wild`）
     - 場札にワイルド、ワイルドドロー 4、シャッフルワイルド以外のカードが出された時（`play-card`, `play-draw-card`）
     - 山札からカードを引いて、プレイヤーがターンを失った時（`draw-card`）
     - チャレンジを行った時
       ※成否によって、次の順番が変わります。
   - 通信データ

```
// On
{
  "next_player": "P-00000001", // 次の順番のプレイヤー - string
  "before_player": "P-00000002", // 前の順番のプレイヤー - string
  "card_before": {
    "number": 9,
    "color": "red"
  }, // 場札のカード - Card
  "card_of_player": [
    {
      "color": "yellow",
      "special": "skip"
    },
    {
      "color": "yellow",
      "number": 8
    },
    {
      "color": "blue",
      "number": 0
    },
```

```
    {
      "color": "blue",
      "number": 9
    },
    {
      "color": "red",
      "number": 5
    },
    {
      "color": "blue",
      "number": 6
    }
  ], // 所持しているカード - Card
  "must_call_draw_card": true, // draw-card イベントを強制させるか - boolean
  "draw_reason": "bind_2", // カードを引かなければならない理由 - DrawReason
  "turn_right": true, // 右回りであるか - boolean
  "number_card_play": 14, // 今対戦中に場札に出されたカードの合計枚数 - number
  "number_turn_play": 14, // 今対戦中のターン数 - number
  "number_card_of_player": {
    "P-00000001": 4, // プレイヤー 1 の手札枚数 - number
    "P-00000002": 3, // プレイヤー 2 の手札枚数 - number
    "P-00000003": 6, // プレイヤー 3 の手札枚数 - number
    "P-00000004": 5 // プレイヤー 4 の手札枚数 - number
  }
}
```

```
// Emit
// なし
```

- 通信データ内の追加説明
  must_call_draw_card : この項目が true の時プレイヤーは場札に出せません。
  must_call_draw_card : デフォルト値は false。この項目が true の時、プレイヤーは `draw-card` イベントを呼び出す必要があります。

  turn_right : この項目が true の時プレイ順序は正順(A→B→C→D)、false の時は逆順（D→C→B→A）となります。

7. `public-card`

   - 機能：手札の公開
     チャレンジを行ったプレイヤーにのみ、ワイルドドロー 4 を出したプレイヤーの手札を公開します。
   - ディーラーからの通知範囲：特定のクライアント
   - タイミング：ゲーム対戦中

    - `challenge`を行った時

- 通信データ

```
// On
{
  "card_of_player": "P-00000002", // 手札を公開するプレイヤーコード - string
  "cards": [
    {
      "number": 9,
      "color": "red"
    },
    {
      "number": 8,
      "color": "yellow"
    },
    {
      "special": "wild_draw_4",
      "color": "black"
    }
  ] // 公開する手札リスト - Card[]
}
```

```
// Emit
// なし
```

8. `finish-turn`

- 機能：対戦終了の通知
  1対戦終了したことを通知します。
  プレイヤーは所持しているカードを廃棄します。

- ディーラーからの通知範囲：全クライアント

- タイミング：ゲーム対戦中

- 通信データ

```
// On
{
  "turn_no": 49, // ターン数 - number
  "winner": "P-00000002", // 今対戦の勝利プレイヤーコード - string
  "score": {
    "P-00000001": 26, // プレイヤー 1 の点数 - number
    "P-00000002": -15, // プレイヤー 2 の点数 - number
    "P-00000003": -6, // プレイヤー 3 の点数 - number
    "P-00000004": -5 // プレイヤー 4 の点数 - number
  }
}
```

```
// Emit
// なし
```

9. `finish-game`

   - 機能：ゲーム終了の通知
     ゲームが終了し、クライアントが切断されます。
   - ディーラーからの通知範囲：全クライアント
   - タイミング：ゲーム対戦中
   - 通信データ

```
// On
{
  "winner": "P-00000002", // ゲームの勝利プレイヤーコード - string
  "turn_win": 28, // 勝利プレイヤーの勝数 - number
  "order": {
    "P-00000001": 28, // プレイヤー 1 の勝数 - number
    "P-00000002": 27, // プレイヤー 2 の勝数 - number
    "P-00000003": 23, // プレイヤー 3 の勝数 - number
    "P-00000004": 33 // プレイヤー 4 の勝数 - number
  },
  "total_score": {
    "P-00000001": 560, // プレイヤー 1 の総得点 - number
    "P-00000002": -125, // プレイヤー 2 の総得点 - number
    "P-00000003": 65, // プレイヤー 3 の総得点 - number
    "P-00000004": -508 // プレイヤー 4 の総得点 - number
  }
}
```

```
// Emit
// なし
```

10. `penalty`

    - 機能：ペナルティの通知
      ペナルティが発生したことを通知します。
    - ディーラーからの通知範囲：全クライアント
    - タイミング：ゲーム対戦中
    - 通信データ

```
// On
{
  "player": "TestPlayer1", // プレイヤーコード - string
  "number_card_of_player": 2, // カード追加後の合計手札枚数 - number
```

```
  "error": "Param card play invalid." // ペナルティの原因 - string
}
```

```
// Emit
// なし
```

（5）ペナルティによる手札の増加数について

ペナルティにより山札からカードを引く枚数が、手札の上限枚数を超える場合は山札からカードを引く枚数が調整されます。
既に手札が上限枚数以上ある場合、ペナルティとして新たにカードを引くことはありません。
ただし、前のプレイヤーが出したカードの効果を受ける場合においては、手札の上限枚数を超えてカードを引きます。
ペナルティについては、「ALGORI 大会」公式 HP の「応募要項」画面内「大会ルール」よりダウンロード後、ご確認ください。

6．その他機能について

（1）管理者ツール

管理者ツールとは、管理者がゲームの操作を行うための GUI ツールです。
「UNO」対戦中の各プレイヤーの挙動に合わせ、カード効果の表示を行います。
当ツールはディーラープログラムを起動することで使用できるようになります。
http://localhost:8080/api/v1/admin/web

（2）開発ガイドライン

プレイヤー開発のサポートを行なうツールで、ディーラープログラムを起動することにより使用できるようになります。
http://localhost:3000/api/v1/test-tool
以下、1〜5 の機能について説明を記載しております。

1. 開発に必要な下準備
2. プレイヤーからディーラーへの通信内容をイベントごとにチェック
3. ディーラーからプレイヤーへの通信をイベントごとに発生させる
4. プレイヤープログラムに組み込むロジックを入れる箇所の紹介
5. 1 試合行うための手順

（3）補足説明

1. MongoDB
   JSON 形式のデータを保存するドキュメント指向型のデータベースです。
   ALGORI 大会ではディーラー情報、プレイヤー情報、ゲームログなどの恒久的なデータを保存するデータベースとして使用しています。
2. Redis
   メモリ上にデータを展開し、高速に処理できるキャッシュデータベースです。
   ALGORI 大会では、ゲームの盤面情報の保存に使用しています。
3. Socket
   サーバーとクライアントが特定のポートを通じてリアルタイムで通信する方式です。
   HTTP 通信はクライアントがリクエストすることでサーバがレスポンスを返しますが、Socket 通信はクライアントからもサーバからもリクエストが発生します。
4. クライアント
   一般的には、コンピュータ間のデータ通信の役割の一つで、データを提供される側の役割をするコンピュータまたはソフトウェアを指します。
   ALGORI 大会では、主に Socket 通信を行うプレイヤーを指しています。
5. Node.js
   javascirpt をサーバ上で駆動させる環境です。

ALGORI 大会では、ディーラープログラムや javascript 版デモプレイヤーを javascript で記述しています。それらのプログラムを実行するために利用しています。

6. Python
   組み込み開発、WEB アプリケーション、デスクトップアプリケーションなどで利用されているプログラミング言語です。
   ALGORI 大会では、Python 版デモプレイヤーで利用しています。
7. docker
   アプリケーションの構築、テスト、デプロイ等ができるソフトウェアプラットフォームです。

7. 終わりに

ALGORI 大会へのお問い合わせ、または開発時におけるご不明点や質問事項がございましたら、弊社「ALGORI大会」公式HPの「お問い合わせ」画面より、必須項目をご入力の上、ALGORI大会運営事務局へのお問い合わせをお願いします。

- お名前：エントリー時にご登録されたお名前でご入力をお願いします。
- メールアドレス：お名前と同様にエントリー時にご入力されたメールアドレスのご入力をお願いします。
- お問い合わせ内容：大会運営に関わるお問い合わせ、または開発に関わるご相談などをお問合せしたい内容のご入力をお願いします。
  ※お問い合わせ内容に対する返信につきましては、上述のメールアドレス宛への返信となりますのでご了承ください。

以上