

# 「ALGORI」システム仕様書

---

第1.0版

# 変更履歴

---

更新箇所	更新内容	更新日	備考
全体	新規作成	2022/12/05	

# 目次

---

- [変更履歴](#)
- [目次](#)
- [1. ALGORI大会概要](#)
  - [\(1\) はじめに](#)
  - [\(2\) 当仕様書について](#)
  - [\(3\) その他](#)
- [2. ALGORI大会のシステムについて](#)
  - [\(1\) システム構成](#)
    - [大会環境](#)
    - [参加者のローカル環境](#)
  - [\(2\) データの通信について](#)
  - [\(3\) ディーラープログラム](#)
  - [\(4\) プレイヤープログラム](#)
- [3. 「UNO」 開始までの作業手順](#)
  - [\(1\) 環境の構築](#)
  - [\(2\) Dockerのインストール](#)
  - [\(3\) メインシステムの起動](#)
  - [\(4\) プレイヤーの起動](#)
  - [\(5\) 「UNO」 の開始、および終了](#)
- [4. データ通信仕様について](#)
  - [\(1\) データタイプ](#)
  - [\(2\) イベント一覧](#)
  - [\(3\) イベントの紹介（プレイヤーが任意に発生させる）](#)
  - [\(4\) イベントの紹介（ディーラーが管理）](#)
- [5. その他機能について](#)
  - [\(1\) 管理者ツール](#)
  - [\(2\) 開発ガイドライン](#)
  - [\(3\) 補足説明](#)
- [6. 終わりに](#)

# 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) 当仕様書について

### 1. 必要な情報の提供

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。

- 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

### 2. 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) その他

1. 『ALGORI -UNO プログラミングコンテスト-』大会のルールについては弊社「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、ご確認くださいようお願いします。

## 2. ALGORI大会のシステムについて

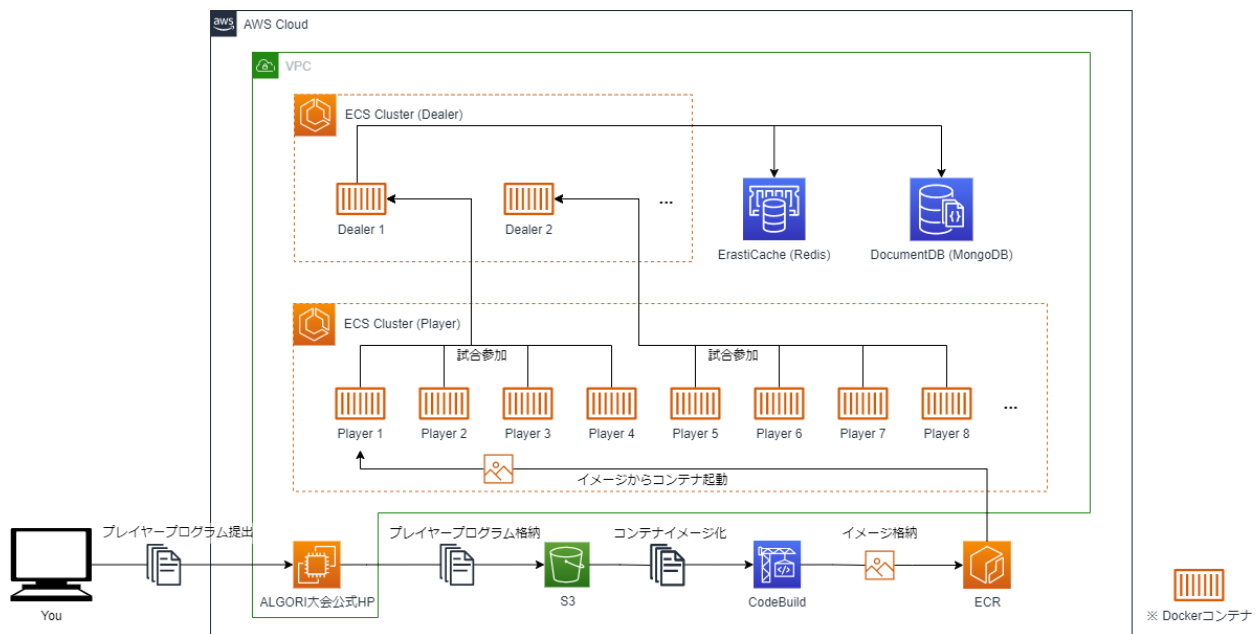
大会を開催する環境（ALGORI システム）とディーラープログラム（以下、ディーラー）とプレイヤープログラム（以下、プレイヤー）のイメージは以下のとおりとなります。

### （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運営側がカスタマイズを行い、プレイヤーは参加者側が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。

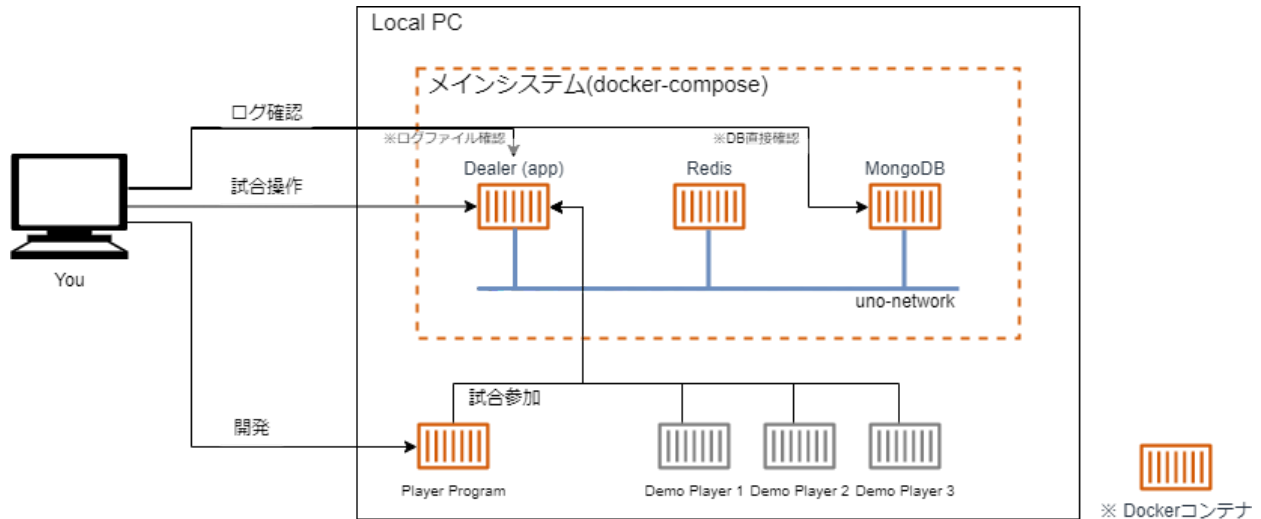
Redisの詳細は後述の「[5. その他機能について](#)」を参照してください。

- 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「[5. 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の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. サーバがクライ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. 対戦開始前の準備
  - プレイヤーの参加管理
  - 対戦回数の設定
2. 対戦中の制御

- 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―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

### 3. 対戦終了の判定

- カード有無における対戦終了判定
- 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

「UNO」を行うために参加される皆様にカスタマイズをしてもらいます。

ここでは開発キットに同封されている「javascript版デモプレイヤープログラム」または「Python版デモプレイヤープログラム」のいずれかを選択してください。

※開発キットについては後述「[3. 「UNO」開始までの作業手順](#)」を参照してください。

参加者同士が競い合う「UNO」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）考えていただきます。

※イベントについては後述「[4. 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」を参考に参加される皆様が“個性的な「プレイヤー」”を作り上げてください。

※まずは同封のデモプレイヤーの内容をご確認いただき、実際に「UNO」での対戦内容をご覧になることをお勧めします。

### 2. スペシャルロジック実装

ALGORI大会独自の仕様「スペシャルロジック」を組む込むことが可能です。

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参加者様次第となります。

※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. プレイヤープログラム作成時のご不明な点や、質問事項が発生した場合

開発時におけるご不明点や質問事項がございましたら、以下の宛先まで内容を記載の上、送信をお願いします。

※問い合わせ先：ALGORI大会運営事務局

件名に下記をコピーの上、貼り付けてメールを送信してください。

本文に問い合わせ内容、またはご相談内容をご記載ください。

- メールアドレス：[algori\\_info@east.ntt.co.jp](mailto:algori_info@east.ntt.co.jp)
- 大会に関わるお問い合わせ
  - 件名：【問い合わせ】大会に関する問い合わせ
- 開発に関わるご相談
  - 件名：【相談】開発に関する相談

### 4. プレイヤープログラムの提出について

#### i. 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「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。**

#### ii. アップロードまでの手順

##### a. プレイヤーの作成

- 開発キットに同封されている「javascript版デモプレイヤープログラム」または「Python版デモプレイヤープログラム」のいずれかを選択後、作成してください。

##### b. 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。

##### c. プレイヤーのコンテナ化

※必須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" ]
```

※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、オリジナリティに富んだ名前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

- 手順cでメインシステムと接続し、ローカル実行ができること
- イメージのサイズが規定を超えていないこと  
イメージサイズの確認は以下のコマンドで行います。

```
$ docker image ls  
REPOSITORY          TAG          IMAGE ID          CREATED
```

```
SIZE
uno-procon-playe  latest  81257416985f  6 hours
ago      917MB
```

#### d. プレイヤーの圧縮

- ファイルのサイズはzip形式で最大1.0GB となっておりますのでご注意ください。

#### e. 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

### iii. 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。

- a. 大会運営に支障をきたす目的のプログラム
- b. 不正を狙っていると思われるプログラム
- c. 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、動作確認中に全く挙動が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。

### iv. Socket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「[4. データ通信仕様について](#)」と「[5. その他機能について](#)」を参照してください。

### 3. 「UNO」 開始までの作業手順

---

「UNO」 開始まで以下の手順で作業を行います。

- 環境の構築
- ディーラーの起動
- プレイヤーの起動
- プレイヤーの作成と更新（プレイヤープログラムの作成と更新）
- 「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運営より配布、または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. メインシステム
  - 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

##### 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
- メールアドレス：algori\_info@east.ntt.co.jp
- 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 v16.17以上
- デモプレイヤープログラム

- Node.js v16.17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。

### 3. 開発キットのインストール

ダウンロードした開発キットをPCの任意の場所で解凍します。

## (2) Dockerのインストール

詳細は[Docker公式ページ](#)を参照してください。

## (3) メインシステムの起動

### 1. メインシステムの起動

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メインシステムが起動します。

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

```
# メインシステムビルド
$ docker-compose build

# メインシステム起動
$ docker-compose up
```

メインシステムには、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。

## 2. ディーラーの設定

メインシステム起動後、「管理者ツール」にて、以下の設定を行います。

それぞれの項目毎に選択後、「新規追加」ボタンをクリックすることで設定されます。

- ディーラー名

ディーラー名は過去に使用したものは使用できません。

新規開始する都度、ディーラー名の変更をお願いします。

- 対戦数

対戦数は任意の数字で入力出来ます。

- 白いワイルドの効果

白いワイルドは、大会ごとに効果が異なります。

2022年度の大会では、バインド2を採用します。

- バインド2

次の手番のプレイヤーを2ターン休みにする。休み中のプレイヤーは、ただ山札からカードを引くことしかできない。チャレンジすることもできない。このカードを出した時、場のカードの色は指定できず、前のカードの色を引き継ぐ。

ALGORI Admin tool

ディーラー一覧

ディーラー名を入力してください

対戦数

バインド2

新規追加

ID : 63593d29fcb8ec0028435782    デイラー名 : Dealer 1

ステータス : new    白いワイルド : bind\_2    対戦数 : 0 / 10

参加待ち

作成日 : 10/26/2022, 1:59:05 PM    更新日 : 10/26/2022, 1:59:05 PM

試合開始

<< < 1 > >>

## (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、作成された「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」のいずれかを選択後、起動してください。

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"
```

### ※注意点※

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

## 2. 複数プレイヤーの起動

同じプレイヤーでも名前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"
```



## (5) 「UNO」の開始、および終了

### 1. ゲームの開始

4プレイヤー起動したら、「管理者ツール」を再読み込んでください。

「試合開始」ボタンをクリックすることで「UNO」が開始します。

対戦中はメインシステムのdockerをストップすることでゲームの中断が可能ですが、あくまで中断なので中断時の状態を保存や途中からの再開はできません。

ALGORI Admin tool


ディーラー一覧

ディーラー名を入力してください

対戦数

バインド2

新規追加

ID : 63593d29fcb8ec0028435782      ディーラー名 : Dealer 1      

ステータス : new      白いワイルド : bind\_2      対戦数 : 0 / 10

ID : [635946ccfcb8ec0028435783](#)      勝利数 : 0      ポイント : 0

ID : [635946d0fcb8ec0028435785](#)      勝利数 : 0      ポイント : 0

ID : [635946edfcb8ec0028435787](#)      勝利数 : 0      ポイント : 0

ID : [635946f2fcb8ec0028435789](#)      勝利数 : 0      ポイント : 0

作成日 : 10/26/2022, 1:59:05 PM      更新日 : 10/26/2022, 2:40:50 PM

試合開始

<< < 1 > >>

### 2. ゲームの終了

設定した対戦数を行うことによって「UNO」が終了します。

#### 。メインシステム

ゲームが終了しても、プロセスは終了しません。続けて新しい試合を行うことができます。

プロセスを終了するときは、「ctrl + c」で終了することができます。

ゲームの途中でメインシステムを終了した場合、プレイヤーとのSocket通信は切断され、デモプレイヤーのプロセスは終了します。

#### 。デモプレイヤー

Socket通信が切断されると、デモプレイヤーのプロセスが終了するようになってい

ます。

再度ゲームを行う場合は、「(3) メインシステムの起動」の2.ディーラーの設定から手順を繰り返してください。

### 3. ゲームログの手順

試合のログは/logsディレクトリにディーラー名で生成されます。

または、MongoDBのactivitiesコレクションに出力されています。

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。

#### i. ログファイル確認方法

##### a. 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_app_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                  PORTS
CREATED        STATUS        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
```

##### b. ログファイル表示

```
# ディーラー名のログファイルを表示
less ./logs/Dealer_1.log
```

##### c. コンテナから抜ける

```
# コンテナから抜ける
exit
```

#### ii. MongoDB確認手順

## a. 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。

確認方法は、上記ログファイルの項目を参照してください。

## b. 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
```

## c. 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"),
    event: "join-room",
    dealer: "Dealer 1",
    player: "635946ccfcb8ec0028435783",
    contents: {
      player_id: ObjectId("635946ccfcb8ec0028435783"),
      player_name: "Player 1"
    },
    dateCreated: ISODate("2022-10-26T14:40:12.212Z"),
    dateUpdated: ISODate("2022-10-26T14:40:12.212Z")
  },
  {
    _id: ObjectId("635946d0fcb8ec0028435786"),
    event: "join-room",
    dealer: "Dealer 1",
    player: "635946d0fcb8ec0028435785",
    contents: {
      player_id: ObjectId("635946d0fcb8ec0028435785"),
      player_name: "Player 2"
    },
    dateCreated: ISODate("2022-10-26T14:40:16.773Z"),
    dateUpdated: ISODate("2022-10-26T14:40:16.773Z")
  },
  {
    _id: ObjectId("635946edfcb8ec0028435788"),
    event: "join-room",
    dealer: "Dealer 1",
    player: "635946edfcb8ec0028435787",
    contents: {
      player_id: ObjectId("635946edfcb8ec0028435787"),
      player_name: "Player 3"
    }
  }
]
```

```
    },
    dateCreated: ISODate("2022-10-26T14:40:45.520Z"),
    dateUpdated: ISODate("2022-10-26T14:40:45.520Z")
  },
  {
    _id: ObjectId("635946f2fcb8ec002843578a"),
    event: "join-room",
    dealer: "Dealer 1",
    player: "635946f2fcb8ec0028435789",
    contents: {
      player_id: ObjectId("635946f2fcb8ec0028435789"),
      player_name: "Player 4"
    },
    dateCreated: ISODate("2022-10-26T14:40:50.739Z"),
    dateUpdated: ISODate("2022-10-26T14:40:50.739Z")
  }
]
```

d. Mongoシェルを終了する

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

e. コンテナから抜ける

```
# コンテナから抜ける
exit
```

## 4. 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. 記号カードの種類

- スキップ  
カードを出した次のプレイヤーの順番がスキップされます。
- リバース  
カードを出す順番が逆方向になります。つまり、プレイヤー1～4という順番がプレイヤー4～1へ変わります。
- ドロー2  
カードを出した次の順番のプレイヤーが山札からカードを2枚引きます。  
ドロー2を受けたプレイヤーはカードを出すことは出来ず、次の順番のプレイヤーへ順番が移ります。
- ワイルド  
カードを出したプレイヤーが自由にカードの色を指定できます。次の順番のプレイヤーは指定された色のカードを出します。
- ワイルドドロー4  
カードを出したプレイヤーが自由にカードの色を指定できると同時に、次の順番のプレイヤーに対して山札からカード4枚引かせることができます。その際、手札にあるカードは出すことは出来ず、次のプレイヤーへ順番が移ります。  
※ワイルドドロー4について  
ワイルドドロー4を出すプレイヤーは手札にあるカードの中に使用出来るカードがあるときは使用できません。（一部例外を除く）  
次のプレイヤー（ワイルドドロー4を出したプレイヤーの次の順番）に「チャレンジ」された場合  
「チャレンジ」したプレイヤーに手札を見せなければなりません。  
※「チャレンジ」後の動向について  
「チャレンジ」の際、手札にワイルドドロー4以外に使用出来るカードがあったことが判明した場合、ワイルドドロー4は手札に戻し、ペナルティとして山札からカ

ード4枚を引かされます。

また「チャレンジ」の際、手札にワイルドドロー4以外に使用出来るカードがなかった場合、「チャレンジ」したプレイヤーはワイルドドロー4効果として山札から4枚カードを引かされたうえにペナルティとして更に追加で2枚引かされます（合計で6枚引くことになります）。

- シャッフルワイルド

シャッフルワイルドを出した時は、全プレイヤーの手札のカードを集めてシャッフルを行います。

※シャッフルワイルドを出してゲーム終了した場合はシャッフルカードを出したプレイヤー以外にシャッフルしたカードを残りのプレイヤーへ配ります。

- 白いワイルド

今回のALGORI大会での効果は「バインド2」を採用しております。

詳細は後述の「3.白いワイルドと効果」の項を参照してください。

プログラムで使用する既定値は以下です。

```
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",  
}
```

## 2. カードの色の種類

- 赤（数字とスキップ、およびリバース系カード）
- 黄（数字とスキップ、およびリバース系カード）
- 緑（数字とスキップ、およびリバース系カード）
- 青（数字とスキップ、およびリバース系カード）
- 黒（ワイルド系カード）
- 白（白いワイルド）

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",  
}
```

### 3. 白いワイルドと効果

- バインド2

効果は「2回行動不可」となります。

プログラムで使用する既定値は以下です。

```
export enum WhiteWild {  
  BIND_2 = "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 {  
  number: number; // 0, 1, 2, ... 9  
  color: Color; // red, yellow, green, blue, black, white  
  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",  
}
```

### 1. 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を場札に出された
- ワイルドドロー4  
前のプレイヤーがワイルドドロー4を場札に出されたため
- バインド2  
前のプレイヤーが白いワイルド（バインド2）を場札に出されたため
- 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",  
    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
- `say-uno-and-play-card` : UNO コールをし、カードを場札に出す
- `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` : 場札の色を変更
- `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



### (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IDを取得します。  
以後、自身のプログラムで自分のプレイヤーID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
{
  "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": "633e3f8a0dbb24745a4c02e4", // プレイヤーID - 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	-
Player name is required.	リクエストデータにplayer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 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 starting. You can not join room	既に開始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
- 通信データ

```
// Emit
{
  "card_play": {
    "number": 9,
    "color": "red"
  } // 場札に出すカード - Card
}
```

```
// On
{
  "player": "633e3f8d0dbb24745a4c02e6", // カードを出したプレイヤー ID - string
  "card_play": {
    "number": 9,
    "color": "red"
  } // 場札に出されたカード - Card
}
```

- 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.	リクエストデータにcard_play.specialの値が既定値と異なる	○
Color card play invalid.	リクエストデータにcard_play.colorの値が既定値と異なる	○
Number card play invalid.	リクエストデータにcard_play.numberの値が既定値と異なる	○
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	○

### 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 `receive-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 4 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  
{}
```

```
// On  
{  
  "player": "633e3f8d0dbb24745a4c02e6", // カードを引いたプレイヤー - ID - 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  
}
```

## ◦ 通信データ内の追加説明

`is_draw`: 手札に25枚以上カードを持っているなど、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、ホワイト 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	-

メッセージ	エラー内容	ペナルティ対象
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。
- 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 - string
}
```

```
// On
{
  "player": "633e3f8d0dbb24745a4c02e6", // プレイヤー ID - string
  "is_play_card": true, // 直前で引いたカードが場札に出されたか - string
  "card_play": {
    "number": 9,
    "color": "red"
  } // 場札に出されたカード - Card
}
```

- 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	-

メッセージ	エラー内容	ペナルティ対象
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 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.	リクエストデータにis_play_card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	○

## 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": "633e3f8d0dbb24745a4c02e6", // チャレンジしたプレイヤー - ID - string
}
```

```

"target": "633e3f8a0dbb24745a4c02e4", // ワイルドドロ-4を出したプレイヤー ID - 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 `null` となります。

## 6. say-uno-and-play-card

◦ 機能：UNO コールをしカードを場札に出された

場札に出されたカードが数字カードのとき、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であった場合、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。

※このイベントはプレイヤーの手札が2枚あるときに呼び出せます。

◦ 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

```
// Emit
{
  "card_play": {
    "number": 9,
    "color": "red"
  } // 場札に出すカード - Card
}
```

```
// On
{
  "player": "633e3f8d0dbb24745a4c02e6", // カードを出したプレイヤー ID
  // string
  "card_play": {
    "number": 9,
    "color": "red"
  }, // 場札に出されたカード - Card
  "yell_uno": true // UNO 宣言 - boolean true のみ
}
```

## ◦ 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 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	○
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.	リクエストデータにcard_play.specialの値が既定値と異なる	○



メッセージ	エラー内容	ペナルティ対象
Color card play invalid.	リクエストデータにcard_play.colorの値が既定値と異なる	○
Number card play invalid.	リクエストデータにcard_play.numberの値が既定値と異なる	○
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	○

## 7. 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": "633e3f8d0dbb24745a4c02e6" // 指摘したいプレイヤー ID - string
}
```

```
// On
{
  "pointer": "633e3f8a0dbb24745a4c02e4", // 指摘したプレイヤー ID - string
  "target": "633e3f8d0dbb24745a4c02e6", // 指摘されたプレイヤー ID - 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
-------	-------	---------

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.	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	○

## 8. special-logic

- 機能：スペシャルロジック発動  
1 対戦中に 1 プレイヤーが実行できるのは 10 回までです。  
10 回以上実行された場合は処理を行われません。
- 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	○

メッセージ	エラー内容	ペナルティ対象
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：ゲーム対戦中
  - 全プレイヤーに対し、対戦開始時に 7 枚のカードを配布される時。
  - 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 1 枚配布される時。
  - 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 枚のカードを配布される時。
  - 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": "633e3f8a0dbb24745a4c02e4", // 最初の順番のプレイヤー
    // ID - string
    "first_card": {
        "color": "yellow",
        "number": 8
    }, // 山札から引いた最初のカード - Card
    "play_order": [
        "633e3f8a0dbb24745a4c02e4",
        "633e3f8d0dbb24745a4c02e6",
        "633e3f900dbb24745a4c02e8",
        "633e3f930dbb24745a4c02ea"
    ] // 順番 - string[]
}

```

```

// Emit
// なし

```

## 3. color-of-wild

- 機能：場札の色を変更  
ワイルド、ワイルドドロー 4、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。
- 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

- ワイルドが場札に出された時
- ワイルドドロー 4が場札に出された時
- 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. 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[]
}
```

```
// Emit
// なし
```

#### 5. 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
  - ワイルド、ワイルドドロー 4、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` , `say-uno-and-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": "633e3f8a0dbb24745a4c02e4", // 次の順番のプレイヤー
  - string
  "before_player": "633e3f8d0dbb24745a4c02e6", // 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": {
    "633e3f8a0dbb24745a4c02e4": 4, // プレイヤー 1 の手札枚数 -
    number
    "633e3f8d0dbb24745a4c02e6": 3, // プレイヤー 2 の手札枚数 -
    number
    "633e3f900dbb24745a4c02e8": 6, // プレイヤー 3 の手札枚数 -
    number
    "633e3f930dbb24745a4c02ea": 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** のときは逆順 (D→C→B→A) となります。

## 6. 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": "633e3f8d0dbb24745a4c02e6", // 手札を公開するプレイヤー ID - 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
```

## 7. 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": "633e3f8d0dbb24745a4c02e6", // 今対戦の勝利プレイヤー ID - string
  "score": {
    "633e3f8a0dbb24745a4c02e4": 26, // プレイヤー 1 の点数 - number
    "633e3f8d0dbb24745a4c02e6": -15, // プレイヤー 2 の点数 - number
    "633e3f900dbb24745a4c02e8": -6, // プレイヤー 3 の点数 - number
    "633e3f930dbb24745a4c02ea": -5 // プレイヤー 4 の点数 - number
  }
}
```



```
// Emit  
// なし
```

## 8. finish-game

- 機能：ゲーム終了の通知  
ゲームが終了し、クライアントを切断されます。
- 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": "633e3f8d0dbb24745a4c02e6", // ゲームの勝利プレイヤー ID  
  - string  
  "turn_win": 28, // 勝利プレイヤーの勝数 - number  
  "order": {  
    "633e3f8a0dbb24745a4c02e4": 28, // プレイヤー 1 の勝数 - number  
    "633e3f8d0dbb24745a4c02e6": 27, // プレイヤー 2 の勝数 - number  
    "633e3f900dbb24745a4c02e8": 23, // プレイヤー 3 の勝数 - number  
    "633e3f930dbb24745a4c02ea": 33 // プレイヤー 4 の勝数 - number  
  },  
  "total_score": {  
    "633e3f8a0dbb24745a4c02e4": 560, // プレイヤー 1 の総得点 -  
    number  
    "633e3f8d0dbb24745a4c02e6": -125, // プレイヤー 2 の総得点 -  
    number  
    "633e3f900dbb24745a4c02e8": 65, // プレイヤー 3 の総得点 - number  
    "633e3f930dbb24745a4c02ea": -508 // プレイヤー 4 の総得点 -  
    number  
  }  
}
```

```
// Emit  
// なし
```

## 5. 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。

当ツールはメインシステ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、メインシステ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

javascript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。

## 6. 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、以下の宛先まで内容を記載の上、送信をお願い申し上げます。

※問い合わせ先：ALGORI大会運営事務局

件名に下記をコピーの上、貼り付けてメールを送信してください。

本文に問い合わせ内容、またはご相談内容をご記載ください。

- メールアドレス：[algori\\_info@east.ntt.co.jp](mailto:algori_info@east.ntt.co.jp)
- 大会に関わるお問い合わせ  
件名：【問い合わせ】大会に関する問い合わせ
- 開発に関わるご相談  
件名：【相談】開発に関する相談

以上