

# MAL 和狩プログラム 説明ドキュメント

T-HAWK

## 内容

はじめに	2
動作環境	3
開発環境	3
動作環境	3
ディレクトリ構成	4
フォルダとファイルの説明	4
コンポーネントの説明	5
各プログラムの説明	5
ビルド・実行方法 (Build & Run Instructions)	11
セットアップ手順	11
実行コマンド	11
操作画面説明	12

## はじめに

本ドキュメントは、協力型マルウェア解析ゲーム「MAL 和狩（まるわかり）」のソースコードに関する技術的な情報を提供することを目的としています。本ゲームは Malware 解析初心者が勉強するきっかけづくりとして作成したものです。本アプリケーションの構成、動作環境、遊び方、主要コンポーネント、及び今後の開発方針について記述します。

このドキュメントを通じて、コードの理解を深め、貢献するためのガイドラインを提供します。

動作環境

開発環境

- 必要ソフトウェア

ソフトウェア/ライブラリ等	重要度	要件
Node.js	必須	v18.17.1 以上
npm	必須	V9.6.7 以上
Docker	推奨	V24.05 以上

- 開発動作確認済み OS

- Windows 11 Pro (64bit)
- Ubuntu 22.04 LTS

動作環境

- 動作確認済み

- Microsoft Edge
- Google Chrome

## ディレクトリ構成

### フォルダとファイルの説明

以下に本アプリケーションを構成するフォルダおよびファイルを示す。

```
project-root/  
├── src/  
│   ├── node_modules/  
│   ├── public/  
│   │   └── index.html  
│   ├── challenges/  
│   │   └── sample_challenges.json  
│   ├── server.js  
│   ├── package.json  
│   └── package_lock.json  
├── LICENSE  
└── README.txt
```

各フォルダおよびファイルの役割を以下に示す。

- src: アプリケーションのソースコード一式
- node\_modules/: npm でインストールされた依存ライブラリ（自動生成）
- public/: 静的ファイルを格納
- index.html: アプリケーションのエントリーポイントとなる HTML ファイル
- challenges/: ゲーム内で使用されるチャレンジデータを格納
- sample\_challenges.json: サンプルチャレンジの定義ファイル
- server.js: Node.js によるサーバーのエントリーポイント
- package.json: プロジェクトの依存関係やスクリプトを定義
- package\_lock.json: 依存関係のバージョンを固定するロックファイル
- LICENSE: ライセンス情報
- README.txt:

## コンポーネントの説明

まず、アプリケーション全体を構成する主要なコンポーネントをリストアップし、それぞれの役割を簡潔に説明します。

本アプリケーションは以下の主要コンポーネントで構成されています：

サーバー (server.js) : ゲームロジックと API の処理を担当

チャレンジ管理 (challenges/) : ゲーム内の問題データを管理

フロントエンド (public/index.html) : ユーザーインターフェースを提供

## 各プログラムの説明

### ● server.js

- 役割 : Node.js ベースの HTTP サーバー。ゲームの状態管理、チャレンジの配信、ユーザーとの通信を担当。

- 主な機能 :

- WebSocket によるリアルタイム通信
- チャレンジデータの読み込みと配信
- ログの出力とエラーハンドリング

- 使用ライブラリー一覧

- express: HTTP サーバーの構築とルーティング処理
- http: Node.js 標準の HTTP モジュール
- socket.io: クライアントとのリアルタイム通信
- cors: クロスオリジン通信の許可
- uuid: 一意な ID の生成
- fs: ファイルシステム操作
- path: ファイルパスの操作
- fileURLToPath: ES モジュールの URL からファイルパスを取得

- 主な関数・イベント

- ゲーム進行ロジック

- ✧ function startTimer(roomId) { ... }:

- ・ 1 秒ごとに残り時間を計算し、timer イベントで送信。
- ・ 時間切れ時にライフを減算 (Hard: 3, Normal: 5)。
- ・ ライフが尽きた場合は gameFinished を送信し、ゲーム終了。

- ・ ライフが残っている場合は `roundTimeout` を送信し、1.5 秒後に次ラウンドへ移行。
  - ・ タイマーは `setInterval` により管理され、ルームごとに記録。
- ✧ `function startRound(roomId) { ... }:`
- ・ モード (Easy/Normal/Hard) に応じた未使用チャレンジを選定。
  - ・ 最大ラウンド数 (デフォルトは 1) に達するとゲーム終了。
  - ・ 初回ラウンド時は `gameStarted` イベントで初期情報を送信。
  - ・ 選定されたチャレンジをルーム状態に記録し、`sendQuestion` を呼び出す。
- ✧ `function sendQuestion(roomId) { ... }:`
- ・ 現在のチャレンジの制限時間を設定し、`expiresAt` に記録。
  - ・ プレイヤーA には配列形式、B には文字列形式で問題文を送信。
  - ・ `newQuestion` イベントで各プレイヤーに出題内容を通知。
  - ・ 出題後に `startTimer` を呼び出してタイマーを起動。
- ✧ `function setReadyStatus(roomId, socketId, role, cb) { ... }:`
- ・ 現在のチャレンジの制限時間を設定し、`expiresAt` に記録。
  - ・ プレイヤーA には配列形式、B には文字列形式で問題文を送信。
  - ・ `newQuestion` イベントで各プレイヤーに出題内容を通知。
  - ・ 出題後に `startTimer` を呼び出してタイマーを起動。
- Socket.io イベント
- ✧ `io.on('connection', socket => { ... }):`
- ・ クライアント接続時に各種イベントリスナーを登録。
  - ・ ゲームの全体的な通信制御を行う。
- ✧ `socket.on('createRoom', ({ mode } = {}, cb) => { ... }):`
- ・ 新しいルームを作成し、初期状態を設定。
  - ・ 6 桁のルーム ID を生成してクライアントに返す。
  - ・ 60 秒間未使用の場合は自動削除。
- ✧ `socket.on('joinRoom', ({ roomId }, cb) => { ... }):`
- ・ 指定されたルームに参加。
  - ・ 参加者数を確認し、ルーム情報を更新。
  - ・ 参加状態をクライアントに通知。
- ✧ `socket.on('playerReady', ({ preferredRole, mode } = {}, cb) => { ... }):`
- ・ プレイヤーの役割とモードを設定。
  - ・ モードに応じてライフを初期化 (Normal: 5, Hard: 3)。
  - ・ 役割が未設定の場合は自動割り当て。

- ・ Ready 状態を記録し、両者が Ready ならゲームを開始。
- ✧ `socket.on('submitAnswer', ({ roomId, answer }, cb) => { ... }):`
  - ・ 回答を受け取り、正誤判定を行う。
  - ・ スコアやライフの更新、次の問題への進行を制御。
- ✧ `socket.on('chat', ({ roomId, message }) => { ... }):`
  - ・ チャットメッセージをルーム内の全プレイヤーに送信。
- ✧ `socket.on('disconnect', () => { ... }):`
  - ・ プレイヤーの切断時にルーム状態を更新。
  - ・ 役割の解除やタイマー停止、通知を行う。
- ✧ `socket.on('continueGame', ({ roomId }) => { ... }):`
  - ・ 同じルームで新しいゲームを開始するために状態をリセット。  
ラウンド数、スコア、使用済み課題、モード、ライフを初期化。

## ■ エラー処理

本アプリケーションでは、各イベントや関数において、状態不整合や不正な入力に対するエラーハンドリングが実装されています。たとえば、存在しないルーム ID や満員のルームへの参加要求には、適切なエラーメッセージを返すことで処理を中断します。また、ゲーム進行中でない状態での回答送信や、問題データが未設定の場合にも、明示的なエラー応答を返します。

これにより、ユーザー体験の向上とサーバー側の安定性が確保されています。

### ➤ 主なエラー応答例

- ✧ ``ROOM_NOT_FOUND``: 存在しないルーム ID
- ✧ ``ROOM_FULL``: 参加者が定員 (2 人) に達している
- ✧ ``ROLES_FULL``: 役割がすでに割り当て済み
- ✧ ``NOT_PLAYING``: ゲームが進行中でない
- ✧ ``NO_QUESTION``: 問題が未設定
- ✧ ``gameOver: true``: ライフが尽きてゲーム終了

## ● challenges/sample\_challenges.json

■ 役割: ゲーム内で使用されるチャレンジ (問題) の定義ファイル。

■ 形式: JSON 形式で、問題文、ヒント、正解などを含む。

### ➤ 問題オブジェクト

- ✧ `title`: 問題のタイトル (例: "検体 1")
- ✧ `timeLimitSec`: この問題に割り当てられた制限時間 (秒単位)



- ✧ baseScore: 正解時に加算される基本スコア
- ✧ roles: プレイヤーごとの表示内容
  - ◆ A.view: 指示者に表示される問題文
  - ◆ B.view: 回答者に表示されるコード断片
- ✧ answer: 正解情報
  - ◆ type: 回答形式（現状は "exact" のみ対応）
  - ◆ value: 正解となる文字列

- public/index.html

- 役割: ゲームの UI を提供する HTML ファイル。
- 特徴: JavaScript でサーバーと通信し、チャレンジを表示・操作可能。
- 主な構成要素
  - <head> セクション
    - ✧ メタ情報:
      - ・ UTF-8 文字コード指定
      - ・ viewport 設定によるレスポンス対応
    - ✧ ライブラリ読み込み: Socket.IO ライブラリ
    - ✧ CSS スタイル定義:
      - ・ カラーテーマ変数 (--bg, --text, --ok など) によるダークテーマ設計
      - ・ `grid` レイアウトによる 2 カラム構成
      - ・ メディアクエリ (`@media (max-width: 900px)`) によるスマホ対応
      - ・ ボタンの状態 (通常・primary・disabled) に応じた視認性向上
      - ・ チャットログ・コードビュー (`#chatLog`, `#roleView`) の表示最適化
      - ・ ゲーム終了時のオーバーレイ表示 (`#clearScreen`)
    - ✧ フォント設定: 日本語環境に配慮したフォント指定
  - <body> セクション: UI セクション
    - ✧ ルーム操作:
      - ・ #create: ルーム作成ボタン。
      - ・ #copyRoom: room ID コピー。
      - ・ #roomId: ルーム ID 入力欄。既存ルームに参加する際に使用。
      - ・ #mode: モード選択 (Easy / Normal / Hard)。
      - ・ #role: 役割選択 (A: 指示者 / B: 回答者 / 自動)。

- ・ #join：指定したルームに参加。
- ・ #ready：準備完了を通知。初期状態では無効。
- ・ #system：現在の待機・進行状況など状態メッセージ表示。
- ・ #status：参加状態表示。
- ・ #players：プレイヤーの役割状況表示。
- ◇ チャット機能:
  - ・ #chatLog：過去のチャット履歴表示領域。
  - ・ #chatMsg：メッセージ入力欄。最大 100 文字まで入力可能。
  - ・ #sendChat：チャットメッセージを送信。
- ◇ ゲーム情報表示
  - ・ #timer：残り時間の表示領域。ゲーム進行中 1 秒ごとに更新。
  - ・ #lives：プレイヤーの残りライフ数を表示。
  - ・ #scores：現在の累積スコアを表示。
- ◇ 回答送信:
  - ・ #answer：回答入力欄。
  - ・ #submit：送信ボタン。
  - ・ #result：回答結果表示。
- ◇ 現在の課題と役割ビュー
  - ・ #challengeTitle：現在の課題タイトルを表示。
  - ・ #roleView：指示者ビュー。コード断片などの情報を表示。
  - ・ #viewControls：ページ操作ボタン群。前後移動・ページジャンプ可能。
  - ・ #prevView / #nextView：前後ページへの移動
  - ・ #jumpToPage：ページ番号入力欄
  - ・ #jumpBtn：指定ページへの移動ボタン
- ◇ ゲーム終了画面:
  - ・ #clearScreen：ゲーム終了時に表示されるオーバーレイ領域。
  - ・ #clearBox：終了メッセージを表示する枠。
- スクリプトセッション (<script>)
  - ◇ ユーティリティ関数: HTML 要素の取得・状態変更・ログ追加などを簡潔に行うための補助関数群。UI 更新の基本操作を担う。
    - ・ \$：ID 指定で HTML 要素を取得する簡易関数。
    - ・ setDisabled(ids, d)：指定した要素群の有効／無効状態を一括で切り替え。

- ・ `appendLog(msg)` : チャットログ領域にメッセージを追加し、自動スクロールする。
- ✧ 状態管理: クライアント側で現在のルーム ID、プレイヤーの役割、Ready 状態などを保持し、UI や通信処理の分岐に利用する。
  - ・ `CURRENT_ROOM` : 現在参加しているルーム ID。
  - ・ `CURRENT_ROLE` : プレイヤーの役割 (A: 指示者 / B: 回答者)。
  - ・ `LOCAL_READY` : 自分が Ready ボタンを押したかどうかの状態。
  - ・ `CURRENT_REMAIN_MS` : 残り時間 (ミリ秒単位)。
  - ・ `CURRENT_PAGES` : 課題ビューの現在ページ番号。
  - ・ `VIEW_PAGE` : 課題ビューの内容を保持する配列。
- ✧ ページ送り機能
  - ・ `updateViewDisplay()` : 現在のページ内容と番号を表示し、前後ボタンの表示・有効状態を制御。
  - ・ `#prevView` / `#nextView` : 前後ページへの移動。
  - ・ `#jumpBtn` : 指定ページへのジャンプ。
- ✧ 回答送信
  - ・ `sendAnswer()` : 回答者 (B) が回答を送信。正誤判定結果を表示。
  - ・ `#submit` : 送信ボタン。クリックで回答送信。
  - ・ `#answer` : 回答入力欄。Enter キーでも送信可能。
- ✧ チャット送信
  - ・ `#sendChat` : チャットメッセージを送信。送信後に入力欄をクリア。
- ✧ Socket.IO イベント受信
  - ・ `connect` / `connect_error` : サーバー接続状態を表示。
  - ・ `roomUpdate` : プレイヤーの参加状況を更新。
  - ・ `readyUpdate` : Ready 状態を更新。
  - ・ `roomReset` : ゲーム再開時のメッセージ表示。
  - ・ `gameStarted` : ゲーム開始時に役割・課題・スコア・ライフを表示し、UI を切り替え。
  - ・ `newQuestion` : 新しい問題の出題。ビュー表示とページ送り初期化。
  - ・ `livesUpdate` : ライフ数をハート記号で表示。
  - ・ `timer` : 残り時間を mm:ss 形式で表示。
  - ・ `updateScore` : 累積スコアを表示。

- ・ `roundCleared`：正解時のスコアと経過時間を表示。
- ・ `roundTimeout`：時間切れメッセージを表示。・ `answerResult`：回答結果（正解／不正解）を表示。
- ・ `chat`：チャットメッセージをログに追加。
- ・ `system`：システムメッセージを表示。
- ・ `gameFinished`：ゲーム終了時にオーバーレイ画面を表示し、最終スコアと再スタート／継続ボタンを提供。

## ビルド・実行方法 (Build & Run Instructions)

### セットアップ手順

1. Node.js をインストール
  - ・ 推奨バージョン: Node.js 18 以上
  - ・ 公式サイトからインストールしてください。
2. プロジェクトの依存関係をインストール
  - ・ プロジェクトルートで以下のコマンドを実行します
  - ・ `npm install`
3. 課題データの確認
  - ・ `challenges/sample_challenges.json` に課題データが格納されています。必要に応じて編集・追加してください。

### 実行コマンド

- ① 以下のコマンドでサーバーを起動
  - ・ `node server.js`
  - ・ `node --experimental-specifier-resolution=node server` (ESM 環境の場合)
- ② クライアントは任意のブラウザからサーバーの URL  
(`http://XXX.XXX.XXX.XXX:4000`)に接続（デフォルト設定ではポートは 4000）

## ゲーム画面説明

### 操作画面



- ① ルーム作成・参加エリア
- ② チャットエリア
- ③ ゲーム情報表示エリア
- ④ 回答エリア
- ⑤ 課題ビュー・ユーザー情報エリア

## ゲーム操作

### I. ゲーム開始まで

1. ①にあるルーム作成ボタンをクリックして新しいルームを作成し、

参加ボタンをクリック

2. room id が発行されるため、room id コピーのクリック・他のツール  
で共有などして一緒に遊ぶプレイヤーと共有する

3. 共有されたプレイヤーは room id を入力し、参加ボタンをクリック

4. ゲームの難易度 (Easy / Normal / Hard) や役割 (A: 指示者 / B: 回答者 / 自動) を選択し、Ready をクリック

※先にクリックしたほうが優先され、役割は自動でクリックした場合、A から埋まる

5. 両者が Ready になると、1.5 秒後にゲームが開始、③にあるゲームステータスが更新

6. ⑤に指示者 (A) は課題文、回答者 (B) はコード断片が表示されるため、それを元に問題を解く

7. 回答者 (B) は④に回答を入力する。

no③にあるゲームステータスが正解か不正解によって変動

正解ならスコアが更新

不正解なら、Normal か Hard の場合にはライフが減少

8. ③にあるタイマーが 0 になり、時間切れになると次ラウンドへ進行

モードが Normal か Hard の場合、ライフが減少

9. ライフが尽きるかラウンド上限に達するとゲーム終了

「もう一度遊ぶ」ボタンでページ再読み込み

「同じルームで続ける」ボタンで再プレイ可能