「AR Robot Battle with V-Sido CONNECT」
ユーザーマニュアルと開発の手引き

第 0.50 版

2016/8/16
アスラテック株式会社

Asratec

目　次

Asratec

1. はじめに

このマニュアルは、「V-Sido CONNECT RC」と「JavaScript SDK for V-Sido CONNECT」（以下、V-Sido JS SDK）を使ったサンプルプログラム「AR Robot Battle with V-Sido CONNECT」（以下、AR Robot Battle）の利用方法やソースコードを解説しています。

1-1. サンプルプログラムの概要

AR Robot Battle は、2 体のロボットを使った AR ロボット対戦ゲームです。物理的なロボット格闘ではなく、スマートフォンなどで映像を見ながら AR で攻撃を行い、仮想のライフを減らして相手を倒す形式の対戦システムとなっています。

なお、提供するコードは場合により変更・修正される可能性があります。本マニュアル上で示すソースコードと完全に一致していない場合もあります。

1-2. 事前に確認すべきマニュアル

AR Robot Battle は、Intel Edison での利用を想定しています。V-Sido CONNECT RC と Intel Edison との接続方法は、別紙のマニュアル『「JavaScript SDK for V-Sido CONNECT」利用の手引き――Intel Edison 編』を参照してください。

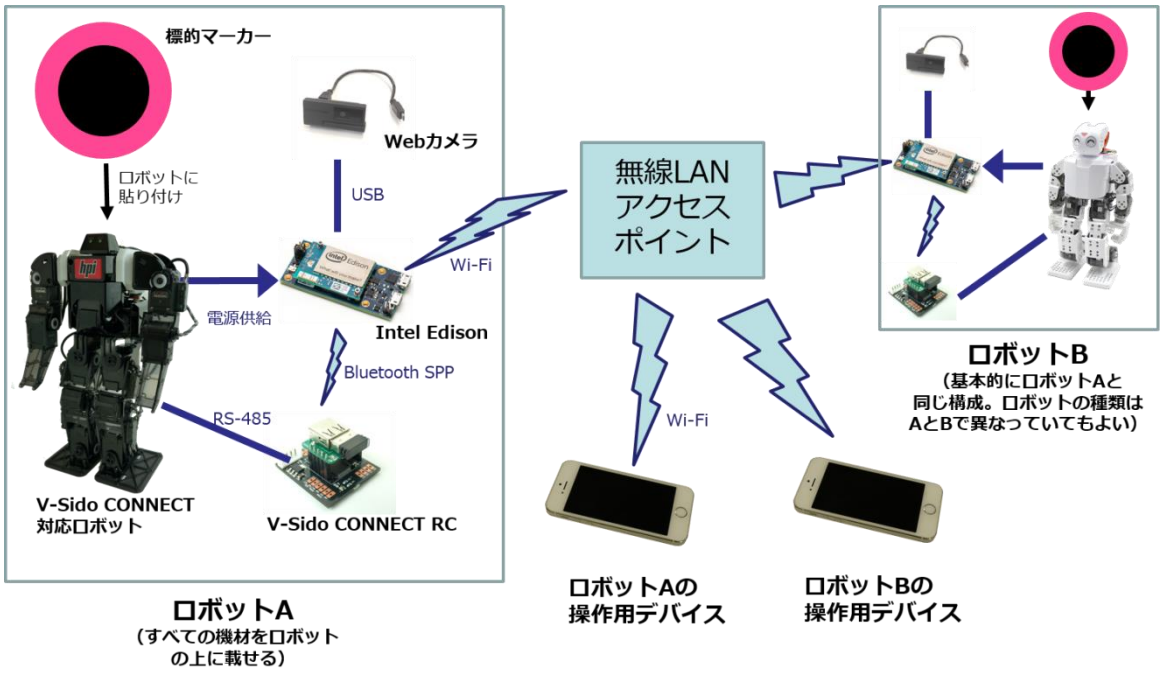
そのほか V-Sido CONNECT RC とロボットとの接続については、『「V-Sido CONNECT RC」スタートアップガイド』（DARWIN-MINI 編および GR-001 編）を参照してください。

各マニュアルは Web サイト（https://v-sido-developer.com/）で公開しています。

2. ハードウェア構成

2-1. 接続構成とネットワーク

AR Robot Battle では、下図のようにロボットとロボットを操作するデバイスが、無線 LAN ですべて同一ネットワークに接続されている必要があります。



2-2. 必要となる機材など

必要なハードウェア類は、下記の通りです。なお、標的マーカーはこのマニュアルで提供するものを印刷するなどして各自ご用意ください（後述）。

- V-Sido CONNECT RC 対応ロボット×2
- V-Sido CONNECT RC（Bluetooth 無線化対応済みのもの）×2
- Intel Edison ＋ Breakout Board ×2
- 小型の USB 接続 Web カメラ×2
- Web ブラウザが利用可能な端末（スマートフォンなど）×2
- 無線 LAN ルータ
- 各機器を接続するためのケーブル類など（USB ホストケーブル含む）

※ ロボットは 2016 年 8 月時点で GR-001 および DARWIN-MINI に対応しています。

※ USB 接続 Web カメラの機種は特に指定はありません。動作検証はエレコムの「UCAM-DLI500TBK」で行いました。

※ 無線 LAN ルータの機種は特に指定はありません。

3. ロボットのセットアップ手順

3-1. 初期準備

① 『「V-Sido CONNECT RC」スタートアップガイド』を参照して、V-Sido CONNECT RC の Bluetooth 無線化を行い、各ロボットに V-Sido CONNECT RC を接続します。

② 『「JavaScript SDK for V-Sido CONNECT」利用の手引き――Intel Edison 編』を参照して、Intel Edison の WiFi 設定や、「VSidoConn4Edison」のインストールを行います。このインストールによって、AR Robot Battle のプログラムも同時に Intel Edison 上にコピーされます。

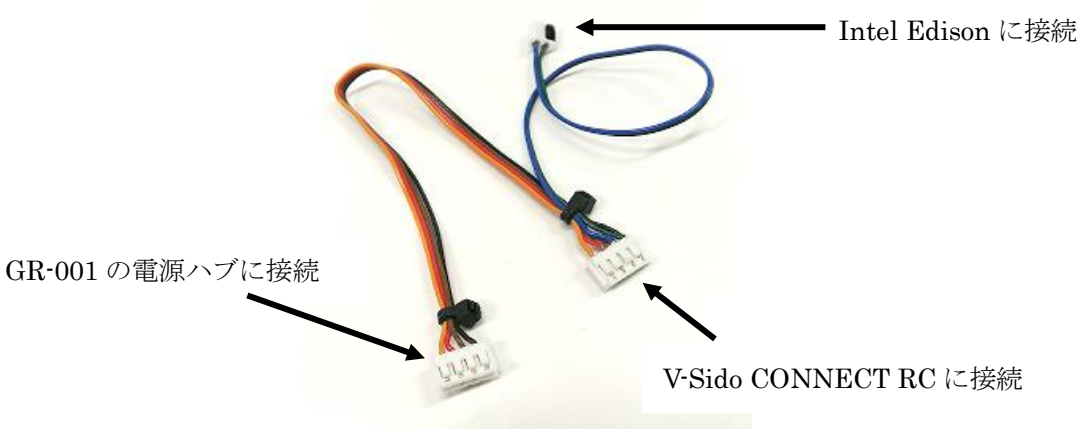
③ 同様に『「JavaScript SDK for V-Sido CONNECT」利用の手引き――Intel Edison 編』を参照して、V-Sido CONNECT RC と Intel Edison との Bluetooth ペアリングを行います。

3-2. Intel Edison のロボットへの搭載

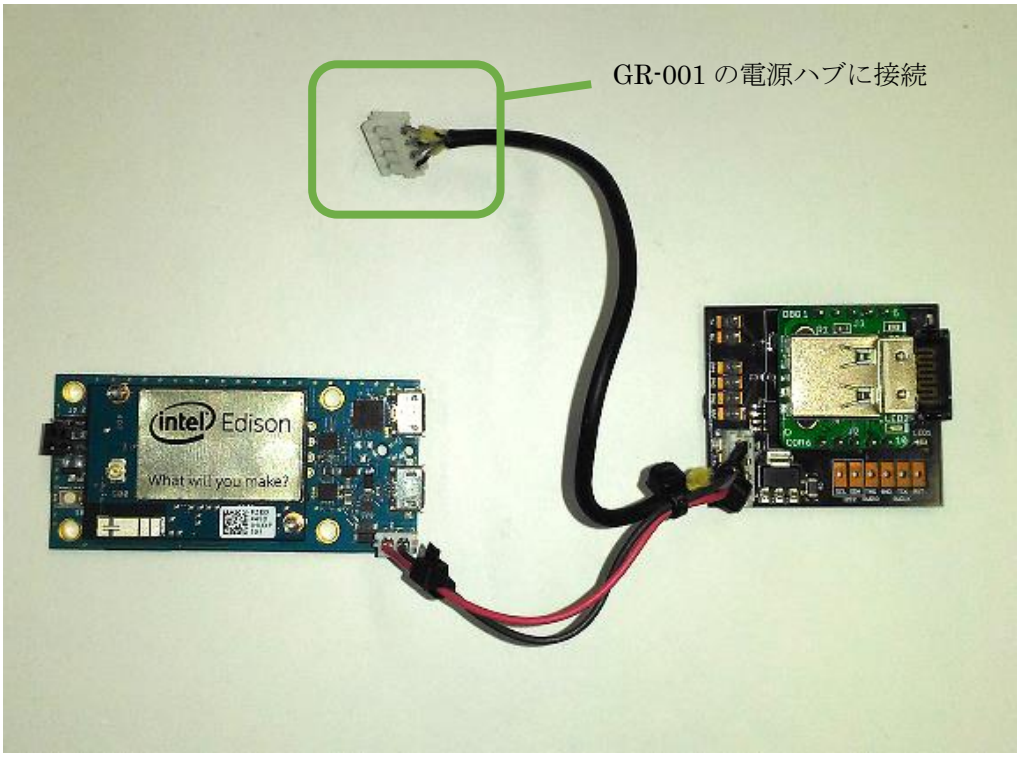
マニュアル『「JavaScript SDK for V-Sido CONNECT」利用の手引き――Intel Edison 編』で解説したシステム構成では、Intel Edison を開発用 PC に接続して USB で電源を供給していますが、今回構築するシステムでは、Intel Edison をロボットに載せるため、ロボットのバッテリーから電源を供給することになります。

3-2-1. GR-001 から Intel Edison への電源供給

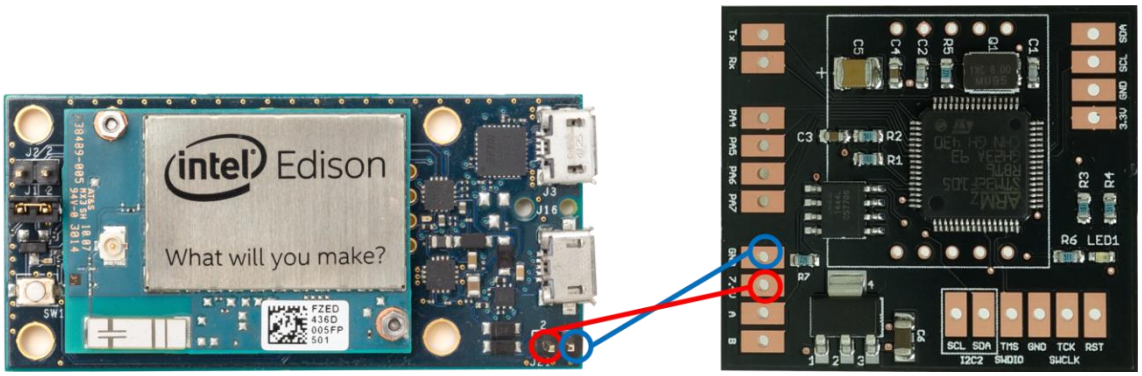
GR-001 の場合、GR-001 の電源ハブと V-Sido CONNECT RC の接続に使っているケーブルを、以下に示す写真のように加工する必要があります（一方のコネクタ側から電源供給用に 2 本の線を追加する）。



接続した状態を写真で示すと、下図のようになります。



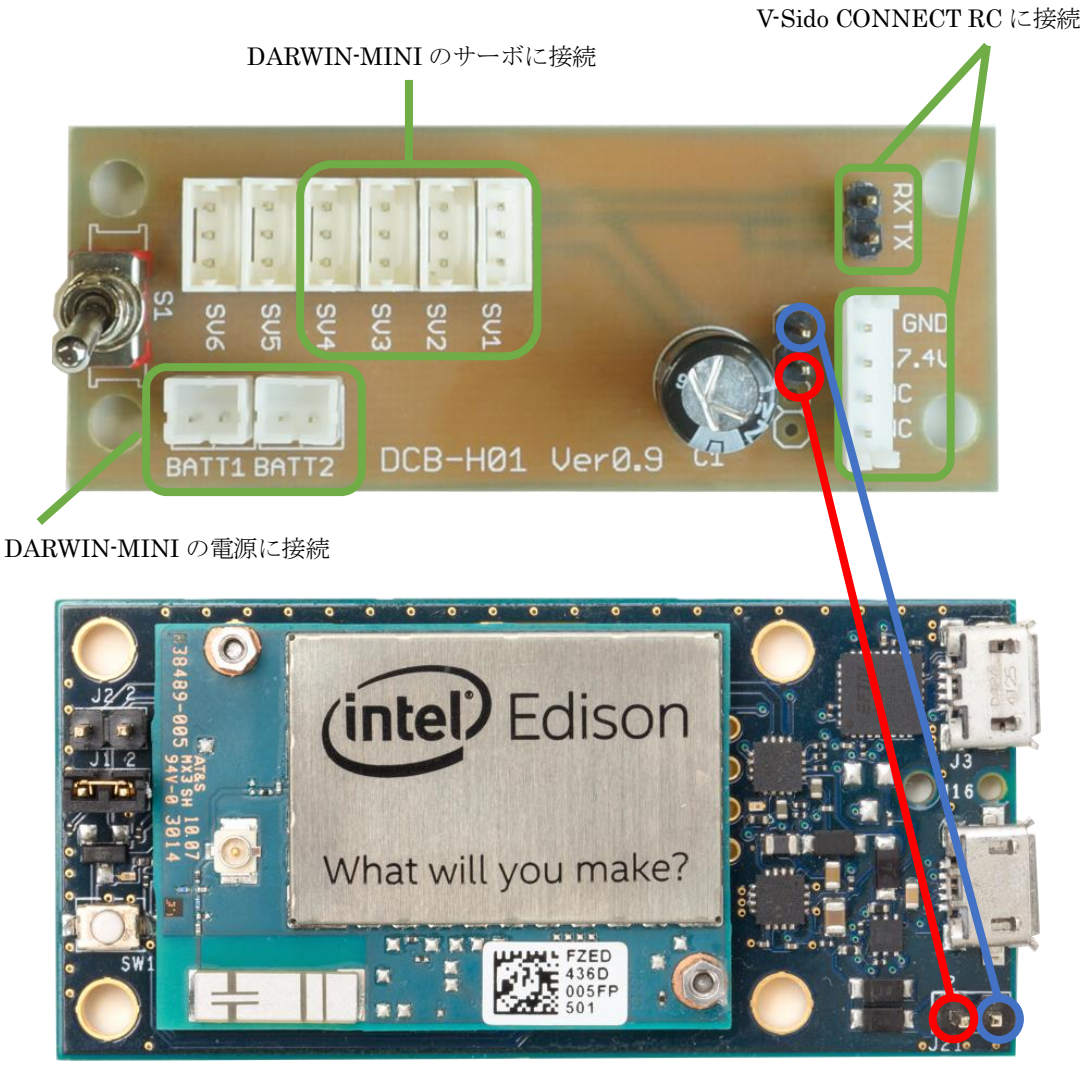
接続の際は、ピンの位置に注意してください。下図のように V-Sido CONNECT RC の 7.4V 端子からの線が Intel Edison の 2 本並んだピンヘッダの内側に、V-Sido CONNECT RC の GND 端子からの線が Intel Edison の 2 本並んだピンヘッダの外側に接続されるようにします。



Intel Edison への電源供給用ケーブルを接続したら、両面テープなどを用いて、Intel Edison を GR-001 の背中部分などに固定してください。

3-2-2. DARWIN-MINI から Intel Edison への電源供給

DARWIN-MINI の場合、専用の接続ハブ「DCB-H01」のピンヘッダからジャンパ線で Intel Edison に接続して電源を供給します（ピンヘッダは必要に応じてはんだ付けしてください)。下図で示すように、Intel Edison と DCB-H01 のピンヘッダを接続してください。



Intel Edison への電源供給用ケーブルの接続が完了したあと、両面テープなどを用いて、Intel Edison を DARWIN-MINI の背中部分などに固定してください。

3-3. Web カメラのロボットへの搭載

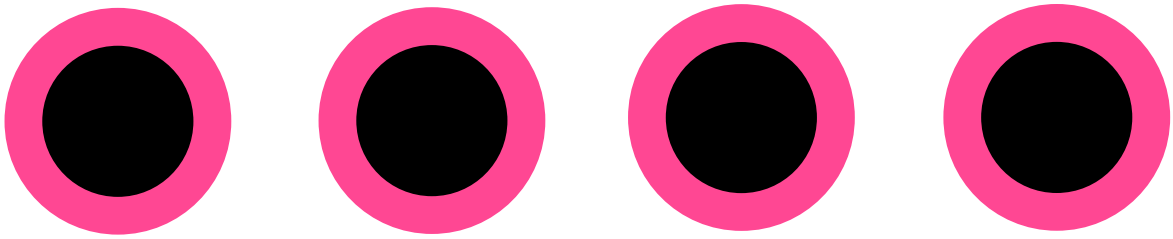
小型の USB 接続の Web カメラを、両面テープなどを使ってロボットの頭部などに固定してください。

ロボットに載せた Web カメラと Intel Edison を、USB ホストケーブルで接続します（なるべく短いものを推奨)。このとき、通常の USB ケーブルではなく、必ず USB ホストケーブル（もしくは変換アダプタ）を用いるようにしてください。また、Web カメラからの USB ホストケーブルは、Intel Edison の下側の USB コネクタに接続します。

USB ホストケーブルで Web カメラを接続

3-4. 標的マーカーの貼り付け

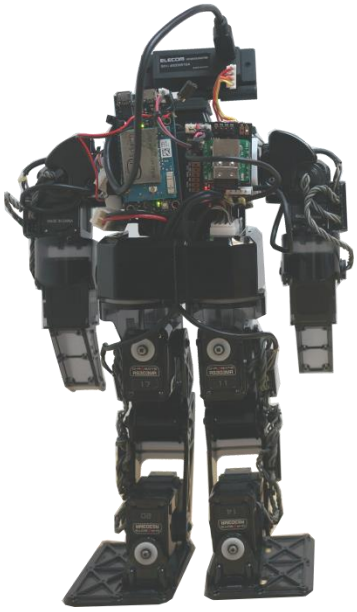
AR Robot Battle では、攻撃判定のために標的マーカーを利用しています。下の図をカラーで印刷して、ロボットの前面に貼り付けてください（マーカーの大きさは、本マニュアルを A4 サイズで印刷したときのもので動作検証を行っています)。



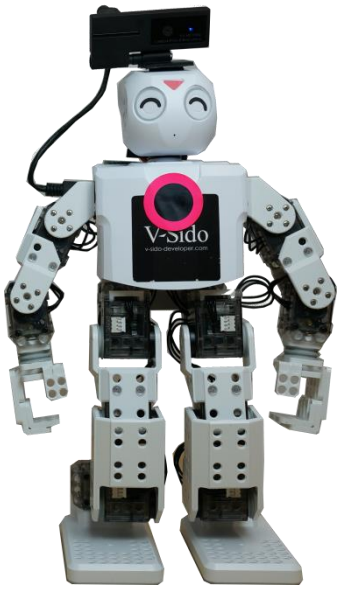
3-5. セットアップ完了後のロボット

ここまでの作業で、AR Robot Battle で使うロボットの設定は完了です。完成写真は、下図のようになります。

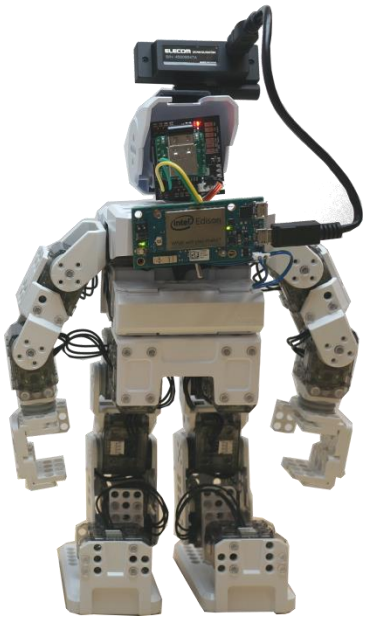
GR-001（正面）



GR-001（背面）



DARWIN-MINI（正面）

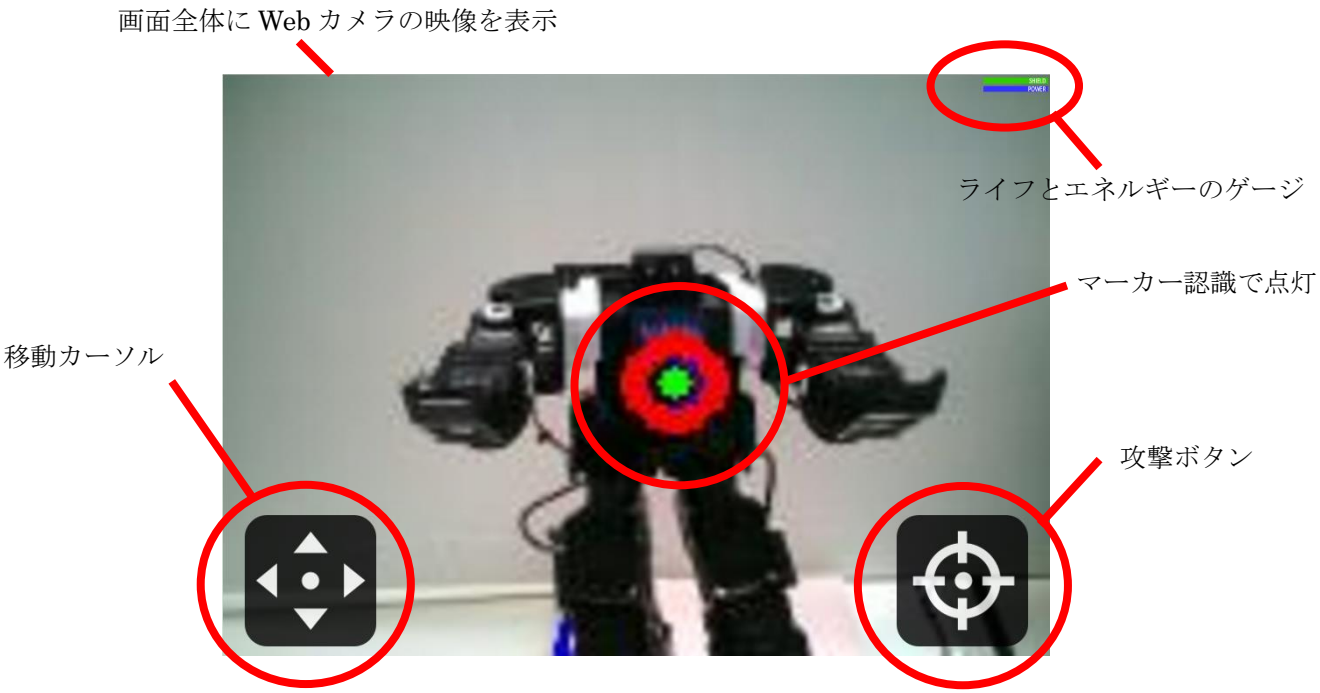


DARWIN-MINI（背面）

4. ゲームの仕様

4-1. 操作画面のインターフェイス

ロボットの操作は、下図のような Web UI（Web ブラウザ）で行います。



4-2. ゲームのルール

- 各ロボットは「エネルギー（POWER）」と「ライフ（SHIELD）」のパラメータを持ちます
- 「エネルギー」は攻撃に必要。最大値は 150 で、攻撃するごとにエネルギーを 20 消費します
- 攻撃は、攻撃ボタンを押すことで実行されます。エネルギーが足りない場合は攻撃ボタンを押しても相手にダメージを与えられません
- 「エネルギー」は時間経過で自動的に回復します
- “カメラで標的マーカーを捉えた状態”かつ“相手と一定距離以内にいる状態”で攻撃を実行することで、相手の「ライフ」が減少します（相手との距離はカメラが認識したマーカーの大きさで判定）
- 攻撃を受けたときの「ライフ」の減少量は、攻撃を受けたときの自分の「エネルギー」によって変動します（自分のエネルギーが少ないと、より多いダメージを受けます）
- ライフの最大値（初期値）は 150 で、0 になった時点でそのプレイヤーの負けと

なります

4-3. ロボットの動作仕様

① 移動カーソルのスワイプまたはクリックでロボットが移動します
② 攻撃ボタンを押したときに、攻撃モーションを行います。攻撃モーションは、両腕を前に突き出す姿勢とします
③ ライフがゼロになったとき（敗北時）にしゃがみます
④ ③の動作のあと、一定時間後に立ち上がります
⑤ 操作デバイスの回転/傾きに合わせて首を回します（カメラの向きを変えます）

※ ⑤の動作は、iPhone など方位センサーに対応したデバイスの使用時かつ、GR-001 など首関節にサーボモータを有するロボットのときに有効です（DARWIN-MINI では首が回転しないので無効となります)。動作検証は iPhone を用いて行っています。ほかのデバイスでは方位センサーに対応していても正しく動作しない可能性があります

※ ⑤ではデバイスを portrait から landscape に変更する際に向いた方向がロボットの真正面となります。逆の場合も同様です

5. ゲームの始め方

5-1. ゲームのインストール

先述の通り、Intel Edison に「VSidoConn4Edison」をインストールすることで、AR Robot Battle も同時にインストールされます。なお、AR Robot Battle のプログラムは、下記に示す Intel Edison のフォルダ下に展開されます。

```
/home/sysroot/usr/share/WebServerGame/
```

5-2. 無線 LAN ルータへの接続

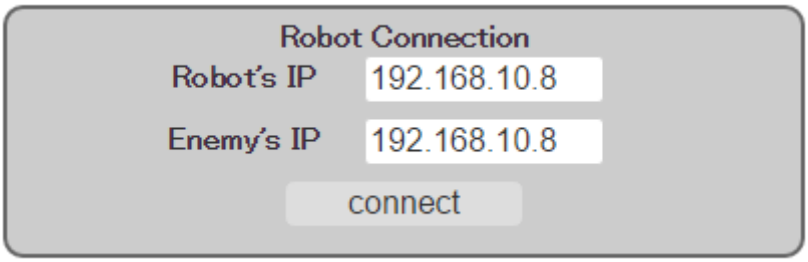
2-1 で示したように、2 台のロボットに接続した Intel Edison と 2 つの操作用デバイスを同じ無線 LAN に接続します。Intel Edison は無線 LAN に接続後、割り当てられた IP アドレスを確認しておいてください。

操作用デバイスの無線 LAN への接続方法は各デバイスの取扱説明書などを、Intel Edison の無線 LAN への接続方法や IP アドレスの確認方法は、『「JavaScript SDK for V-Sido CONNECT」利用の手引き――Intel Edison 編』などを参照してください。

5-3. Web ブラウザから Intel Edison へのアクセス

操作用デバイス A の Web ブラウザを起動して、ロボット A の IP アドレス（ロボット A に接続した Intel Edison の IP アドレス）にポート 8088 でアクセスします（たとえばロボット A の IP アドレスが 192.168.10.8 だった場合、「http://192.168.10.8:8088」にアクセスします)。

無事にアクセスできると、下記のような画面が表示されます（なおこの画面の実体は、Intel Edison の/home/sysroot/usr/share/WebServerGame/index.html です)。



Robot's IP の欄には、このときアクセスしているロボット A の IP アドレスが入っているはずです。ここで「Enemy's IP」の箇所に、ロボット B の IP アドレスを入力

して「connect」ボタンを押してください。対戦ゲームの画面（4-1 項で示した画面）が表示されます。

ロボット B に関しても、操作用デバイス B を使って同様の手続きを行って、対戦ゲーム画面が表示されるようにしてください。

以上で AR Robot Battle の準備は完了です。実際にロボットを動かして「4. ゲームの仕様」で示したような動きをとることを確認して、遊んでみてください。

6. AR Robot Battle のプログラム解説

6-1. ファイル構成

AR Robot Battle のプログラムは HTML、JavaScript、CSS などを使用して構成されています。先述の通り、/home/sysroot/usr/share/WebServerGame の下に展開されますが、このフォルダ直下にある index.html が画面を構成する HTML ファイルで、その他の CSS、画像、JavaScript などは assets フォルダ以下にまとめられています。ディレクトリ直下のもう 1 つの WebServer.js は Web サーバプログラムです。

/home/sysroot/usr/share/WebServerGame/フォルダ以下ファイル構成は以下のようになっています。

```
/home/sysroot/usr/share/WebServerGame/
├──WebServer.js  ──Web サーバプログラム
├──index.html  ──対戦ゲーム画面の HTML ファイル
└──assets
    ├──js
    │   ├──game.js  ── 対戦ゲームのプログラム本体
    │   ├──vsido.client.api.js  ── V-Sido CONNECT API（V-Sido JS SDK の主要部）
    │   ├──battle.camera.js  ── AR Robot Battle 用の API（カメラクラス）
    │   └──battle.r2r.js  ── AR Robot Battle 用の API（ロボット間通信クラス）
    ├──css
    │   └──style.css  ── 対戦ゲーム画面のスタイルファイル
    └──images
        ├──…  ── 対戦ゲームで使う画像類
```

対戦ゲーム自体のプログラムは assets/js/フォルダの game.js ファイルとなります。たとえば、AR Robot Battle を自分でカスタマイズしたい場合は、このファイルを改編していく形となります。

6-2. js フォルダ以下の API

js フォルダの下にある、game.js 以外の 3 つの JavaScript ファイルは、アスラテックが提供する API です。これらの JavaScript ファイル/API の詳細については、下記の Web サイトを参照してください。

●V-Sido CONNECT API（vsido.client.api.js）

https://v-sido-developer.com/files/VSidoConnServer/api/v0.96/index.html

●AR Robot Battle API（battle.camera.js/battle.r2r.js）

https://v-sido-developer.com/files/VSidoConn4Edison/ar_robot_battle_api/v0.96/index.html

V-Sido CONNECT API（vsido.client.api.js）は、V-Sido CONNECT RC が備えているほとんどのロボット制御機能を利用できますが、game.js ではそのうちの

vsido.SetIK　IK 設定コマンドを生成
vsido.Walk　歩行設定コマンドを生成

といったクラスを呼び出しています。

これらを使って、「4-3. ロボットの動作仕様」で示したロボットの動作をプログラミングしています。4-3 項の①は歩行設定コマンド、②～⑤は IK 設定コマンドを用いています。

なお、4-3 項の⑤のような首の動きは、IK 設定ではなくサーボモータの角度設定（vsido.SetServoAngle）でも実装できますが、その場合はサーボ ID が各ロボットで異なる点に注意が必要です（GR-001 では首のサーボモータの ID は 2 ですが、DARWIN-MINI ではサーボ ID2 は左肩部分となります）。

AR Robot Battle API は、AR Robot Battle のために用意した API です。Web カメラ関連の処理（カメラビューやマーカー検出）を行う battle.camera.js と、ロボット同士の通信（R2R）を処理する battle.r2r.js の 2 つのファイルに分かれています。

6-2-1. vsido.SetIK

vsido.SetIK は IK 設定コマンドを生成するクラスです。ロボットのさまざまな部位に対して、位置（position）、姿勢（rotation）、トルク（torque）のいずれかを指定して、X 軸、Y 軸、Z 軸を記述することで IK を設定できます（ただし現状の V-Sido CONNECT RC では、“頭部”の“姿勢”による IK 設定、“右手/左手/右足/左足”の各部位の“位置”による IK 設定のみをサポートしています）。

たとえば位置で IK を指定する場合は、下記のような使い方になります。

```
var ik = new vsido.SetIK({'position':true});
ik.setPosition(kid,x,y,z);
```

パラメータの kid はロボットの部位を示し、部位に応じて下記の数値が入ります。またパラメータの x、y、z はそれぞれ－100～100（%）の値で表します。

KID	1	2	3	4	5
部位	頭部	右手	左手	右足	左足

4-3 項で示した IK で行うモーションのうち、攻撃姿勢（attack）、立ち上がる（standup）、しゃがむ（crouch）の動きは、game.js の 109～220 行で一つの動きを連想配列（オブジェクト）の配列として定義し、それを時間順に並べた配列として定義しています。なお、そのうち 112～114 行で両腕を下ろしたポーズ（lowerhands）のモーションも定義していますが、これは腕を初期位置に戻すためのものです。これらの IK の配列は、後ほど解説する game.js の RobotController Class の制御ループから呼び出して再生します。

残る IK モーションの「首を回す」動きについては、配列として定義せず、406～409 行でプログラムしています。

6-2-2. vsido.Walk

vsido.Walk は歩行設定コマンドを生成するクラスです。vsido.Walk は、下記のような使い方になります。

```
var walk = new vsido.Walk(forward, turnCW);
```

パラメータの forward は前後の移動速度、turnCW は旋回速度を示しており、－100～100 の値で指定します（単位は%。forward は＋が前進、－が後進。turnCW は＋が時計回り、－が反時計回り）。

game.js では、369～370 行で呼び出しています。

6-3. ゲームプログラミングの解説

game.js には、主に 3 つの Class が定義されています。ロボットの制御を行う RobotController Class、画面描画を行う DisplayManager Class、ゲーム全体の進行管理やルールを決めている GameManager Class です。

6-3-1. RobotContloller Class（ロボット制御関連）

ロボットの制御を行う RobotController Class は、10～425 行で定義されています。V-Sido CONNECT API を利用しロボットに命令を送る処理は、すべてこの Class の関数で行っています。

実際のロボットへの命令は、制御ループの中で行っており、制御ループ部の実体は RobotController Class の mainLoop_関数（346～424 行）となります。

ループの実行は setInterval により一定時間ごと（別途定義した fps によって決定される）に呼び出されます。289～294 行の startMainLoop 関数がロボットとの接続後に実行され、ループがスタートします。

このループではまず 352～363 行の箇所で、フラグを利用してコマンドの優先順位を決めて、順次コマンドを実行させるようにしています。これは、ロボットに同時に送ることのできるコマンド数が限られるためです。

実際のコマンド処理は、以下の箇所で記述しています。

367～371 行：ロボットの移動（歩行制御）のコマンド
372～402 行：カスタムモーション（IK アニメーション）のコマンド
403～410 行：カメラの向きの動き（首の回転）のコマンド

IK のコマンドは、首の回転や歩行コマンドと違って、決められた IK の値を再生することで実現します。このときの IK の再生の管理には再生フレームを用いており、ある IK の再生中に同じ KID の IK モーションセットを再生できないようになっています（異なる KID のモーションセットは同時に再生できます）。

歩行や首を動かすなどの処理も発生するため、1 フレームぶん各種 IK モーションを再生したらループを進め、順にほかの処理をしたあとに、IK モーションの次のフレームを再生する、というような形で制御が実行されていきます。

これらは大量のコマンドを処理しながらロボットを制御するための工夫です。

6-3-2. DisplayManager Class（描画関連）

画面の描画は、428～667 行で定義する DisplayManager Class が受け持ちます。

多くの画面構成要素が HTML の DOM 要素（実際には div 要素）となっており、動的にそれらを変化させて画面を構成しています。

たとえば、SHIELD や POWER のメーターゲージは、528～535 行の setMeterValue:関数の中で変化させますが、実際には DOM 要素の width パラメータを変化させることで実現しています。

また、攻撃エフェクト（黄色い弾が飛んでいく）には Canvas を利用しています。556～605 行の setAttackEffect 関数で処理をしています。また、この攻撃エフェクトのアニメーションには setInterval を用いたループ処理を利用しています。581～604 行でループ処理を定義しています。

6-3-3. GameManager Class（ゲーム進行とルール関連）

GameManager Class は 670～1170 行で定義されており、この中でゲーム全体の進行管理やルールの確認、またロボット間の R2R 通信、ブラウザのイベント処理定義などを行います。

この Class のコンストラクタの中で DisplayManager のインスタンスの生成（730～733 行）を行っており、ロボットの IP アドレスなどを指定して接続する「connect」

ボタン押下時の処理関数である connectRobots（795～836 行）の中では、RobotController Class のインスタンスの生成や、カメラとの接続、R2R 通信の接続なども行っています。

また、GameManager Class の rechargePower_関数（244～308 行）は、定期的に POWER を回復する処理を行うループとなっています。ループの実行は 869～870 行の setInterval により定期的に呼びだされます。

6-3-4. イベント関連

イベントはおもに、以下の 3 つのパターンに分けられます。

- 方位センサーからのイベント（対応デバイス時のみ）
- UI 操作からのイベント
- 相手ロボットからのリモートコールイベント

方位センサーイベントは、デバイスの向きに合わせてロボットの頭の向きを制御しています。

移動ボタンや攻撃ボタンなどは UI 操作イベントに分類されます。ユーザーの操作によって、移動コマンド（歩行制御）を発行したり、攻撃コマンド（IK アニメーションループの再生と攻撃エフェクトの描画）を発行したりします。

対戦ゲームを成立させるためのリモートコールイベントは、基本的にロボット同士とのメッセージ通信に利用しています（1064～1102 行の GameManager Class の remotecallback 関数を参照）。おもに敵を攻撃する際、敵に「攻撃した」というメッセージと一緒に自分からのカメラ画像に基づいた敵マーカーを認識しているかどうかとその距離情報を送り、マーカーがある決められた距離以内にあった場合に、敵がダメージを受ける判定になり、敵のライフを減らすことに利用しています。また、勝敗が決定したときのメッセージのやり取りに利用しています。

6-4. カスタマイズ&オリジナルのゲーム開発

この AR Robot Battle のプログラムはサンプルプログラムであり、ユーザーは自由に改編するなどの利用ができます。FTP、SCP などの方法で game.js をコピーして、このコードを編集して機能の追加などを行ってください。また、必要な機能を game.js から真似て、独自のゲームを開発することも可能です。

Asratec

Appendix：AR Robot Battle（game.js）のコード

```
  1: /**
  2:  * @fileoverview V-Sido対応ロボットをV-SidoのJavaScriptライブラリで操作するサンプルコードのロボットAR対戦ゲームです
  3:  * @author Daisuke IMAI <daisimai@asratec.co.jp>
  4:  */
  5: (function(){
  6:   //
  7:   // Class定義部
  8:   //
  9:
 10:   /**
 11:    * ロボットのコントロールを司るClass
 12:    * @param {Object} connectParameters ロボットに接続するためのパラメータ
 13:    * @constructor
 14:    */
 15:   var RobotController = function(connectParameters) {
 16:
 17:     /**
 18:      * ロボットへの接続オブジェクトのインスタンス保持用
 19:      * @type {Object}
 20:      */
 21:     this.connect = null;
 22:
 23:     /**
 24:      * 接続状態管理
 25:      * @type {Boolean}
 26:      */
 27:     this.connected = false;
 28:
 29:     /**
 30:      * 接続がダミー接続(テスト用)であるか否か
 31:      * @type {Boolean}
 32:      */
 33:     this.isDummyConnection = false;
 34:
 35:     /**
 36:      * ロボットへ秒間何回命令を送るか
 37:      * 30以上はSPPの処理が追いつかない可能性があるので、30くらいを目安に。
 38:      * @type {String}
 39:      */
 40:     this.fps = 30;
 41:
 42:     /**
 43:      * 歩行パラメータ
 44:      * @type {Object}
 45:      */
 46:     this.walkParameters = {
 47:       // 前後方向のパラメータ
 48:       forward: 0,
 49:       // 左右回転方向のパラメータ
 50:       turnCW: 0,
 51:     };
 52:
 53:     /**
```

```
 54:      * カメラ角度(実際には首のヨー軸角度に割り当てられている)
 55:      * @type {Number}
 56:      */
 57:     this.cameraAngle = 0;
 58:
 59:     /**
 60:      * 最後に処理したプロセス種別
 61:      * @type {Number}
 62:      */
 63:     this.lastProcess = 0;
 64:
 65:     /**
 66:      * プロセス種別ごとに動作中かを保持する
 67:      * @type {Object}
 68:      */
 69:     this.processInUse = this.initProcessInUse_();
 70:
 71:     /**
 72:      * カスタムモーションごとに動作中かを保持する
 73:      * @type {Object}
 74:      */
 75:     this.motionState = this.initMotionState_();
 76:
 77:     /**
 78:      * メインループのインターバル処理用
 79:      * @type {Object}
 80:      */
 81:     this.mainLoop = null;
 82:
 83:     // ロボットに接続(WSのconnect)する。
 84:     // ※接続するロボットを指定したい場合にipを設定する。例：{'ip':'127.0.0.1'}（コ
ンストラクタの引数として来ているものを利用する）
 85:     if (connectParameters.ip === 'dummy') {
 86:       // 接続先が"dummy"の場合、ロボットには接続せず、ダミー接続モードで起動
 87:       this.connected = true;
 88:       this.isDummyConnection = true;
 89:       console.log("dummy接続モード");
 90:     } else {
 91:       // V-Sidoのインスタンスを生成、接続済み時にはonConnected()をコールバックする
 92:       this.connect = new vsido.Connect(connectParameters);
 93:       this.connect.onOpen = function(event) {
 94:         this.connected = true;
 95:         this.onConnected(event);
 96:       }.bind(this);
 97:     }
 98:   };
 99:   RobotController.prototype = {
100:     // Class定数定義
101:
102:     // 処理の種別ならびに順序付け
103:     PROCESS_TYPE: [
104:         'walk',
105:         'motion',
106:         'camera'
107:     ],
108:
109:     // カスタムモーションの定義
110:     CUSTOM_MOTION: {
111:       // 腕を下ろす動き
```

```
 112:      'lowerhands': [
 113:        [{'kid': 2, 'position': {x: -15, y: 0, z: -95}}, {'kid': 3, 'positio
n': {x: 15, y: 0, z: -95}}]
 114:      ],
 115:      // 攻撃アクション(両腕を上げる)
 116:      'attack': [
 117:        [{'kid': 2, 'position': {x: 0, y: 0, z: -100}}, {'kid': 3, 'position
': {x: 0, y: 0, z: -100}}],
 118:        [{'kid': 2, 'position': {x: 0, y: -25, z: -75}}, {'kid': 3, 'positio
n': {x: 0, y: -25, z: -75}}],
 119:        [{'kid': 2, 'position': {x: 0, y: -50, z: -50}}, {'kid': 3, 'positio
n': {x: 0, y: -50, z: -50}}],
 120:        [{'kid': 2, 'position': {x: 0, y: -75, z: -25}}, {'kid': 3, 'positio
n': {x: 0, y: -75, z: -25}}],
 121:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 122:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 123:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 124:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 125:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 126:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 127:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 128:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 129:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 130:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 131:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 132:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 133:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 134:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 135:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 136:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 137:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 138:        [{'kid': 2, 'position': {x: 0, y: -100, z: 0}}, {'kid': 3, 'position
': {x: 0, y: -100, z: 0}}],
 139:        [{'kid': 2, 'position': {x: -15, y: 0, z: -25}}, {'kid': 3, 'positio
n': {x: 15, y: 0, z: -25}}],
 140:        [{'kid': 2, 'position': {x: -15, y: 0, z: -35}}, {'kid': 3, 'positio
n': {x: 15, y: 0, z: -35}}],
 141:        [{'kid': 2, 'position': {x: -15, y: 0, z: -45}}, {'kid': 3, 'positio
n': {x: 15, y: 0, z: -45}}],
 142:        [{'kid': 2, 'position': {x: -15, y: 0, z: -55}}, {'kid': 3, 'positio
n': {x: 15, y: 0, z: -55}}],
 143:        [{'kid': 2, 'position': {x: -15, y: 0, z: -65}}, {'kid': 3, 'positio
```

```
n': {x: 15, y: 0, z: -65}}],
 144:        [{'kid': 2, 'position': {x: -15, y: 0, z: -75}}, {'kid': 3, 'positio
n': {x: 15, y: 0, z: -75}}],
 145:        [{'kid': 2, 'position': {x: -15, y: 0, z: -85}}, {'kid': 3, 'positio
n': {x: 15, y: 0, z: -85}}],
 146:        [{'kid': 2, 'position': {x: -15, y: 0, z: -95}}, {'kid': 3, 'positio
n': {x: 15, y: 0, z: -95}}]
 147:      ],
 148:      // しゃがみ込む
 149:      'crouch': [
 150:        [{'kid': 4, 'position': {x: 0, y: 20, z: -90}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -90}}],
 151:        [{'kid': 4, 'position': {x: 0, y: 20, z: -89}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -89}}],
 152:        [{'kid': 4, 'position': {x: 0, y: 20, z: -88}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -88}}],
 153:        [{'kid': 4, 'position': {x: 0, y: 20, z: -87}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -87}}],
 154:        [{'kid': 4, 'position': {x: 0, y: 20, z: -86}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -86}}],
 155:        [{'kid': 4, 'position': {x: 0, y: 20, z: -85}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -85}}],
 156:        [{'kid': 4, 'position': {x: 0, y: 20, z: -84}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -84}}],
 157:        [{'kid': 4, 'position': {x: 0, y: 20, z: -83}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -83}}],
 158:        [{'kid': 4, 'position': {x: 0, y: 20, z: -82}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -82}}],
 159:        [{'kid': 4, 'position': {x: 0, y: 20, z: -81}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -81}}],
 160:        [{'kid': 4, 'position': {x: 0, y: 20, z: -80}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -80}}],
 161:        [{'kid': 4, 'position': {x: 0, y: 20, z: -79}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -79}}],
 162:        [{'kid': 4, 'position': {x: 0, y: 20, z: -78}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -78}}],
 163:        [{'kid': 4, 'position': {x: 0, y: 20, z: -77}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -77}}],
 164:        [{'kid': 4, 'position': {x: 0, y: 20, z: -76}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -76}}],
 165:        [{'kid': 4, 'position': {x: 0, y: 20, z: -75}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -75}}],
 166:        [{'kid': 4, 'position': {x: 0, y: 20, z: -74}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -74}}],
 167:        [{'kid': 4, 'position': {x: 0, y: 20, z: -73}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -73}}],
 168:        [{'kid': 4, 'position': {x: 0, y: 20, z: -72}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -72}}],
 169:        [{'kid': 4, 'position': {x: 0, y: 20, z: -71}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -71}}],
 170:        [{'kid': 4, 'position': {x: 0, y: 20, z: -70}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -70}}],
 171:        [{'kid': 4, 'position': {x: 0, y: 20, z: -69}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -69}}],
 172:        [{'kid': 4, 'position': {x: 0, y: 20, z: -68}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -68}}],
 173:        [{'kid': 4, 'position': {x: 0, y: 20, z: -67}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -67}}],
 174:        [{'kid': 4, 'position': {x: 0, y: 20, z: -66}}, {'kid': 5, 'position
```

```
': {x: 0, y: 20, z: -66}}],
 175:        [{'kid': 4, 'position': {x: 0, y: 20, z: -65}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -65}}],
 176:        [{'kid': 4, 'position': {x: 0, y: 20, z: -64}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -64}}],
 177:        [{'kid': 4, 'position': {x: 0, y: 20, z: -63}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -63}}],
 178:        [{'kid': 4, 'position': {x: 0, y: 20, z: -62}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -62}}],
 179:        [{'kid': 4, 'position': {x: 0, y: 20, z: -61}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -61}}],
 180:        [{'kid': 4, 'position': {x: 0, y: 20, z: -60}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -60}}]
 181:      ],
 182:      // しゃがみ込みからの立ち上がり
 183:      'standup': [
 184:        [{'kid': 4, 'position': {x: 0, y: 20, z: -60}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -60}}],
 185:        [{'kid': 4, 'position': {x: 0, y: 20, z: -61}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -61}}],
 186:        [{'kid': 4, 'position': {x: 0, y: 20, z: -62}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -62}}],
 187:        [{'kid': 4, 'position': {x: 0, y: 20, z: -63}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -63}}],
 188:        [{'kid': 4, 'position': {x: 0, y: 20, z: -64}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -64}}],
 189:        [{'kid': 4, 'position': {x: 0, y: 20, z: -65}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -65}}],
 190:        [{'kid': 4, 'position': {x: 0, y: 20, z: -66}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -66}}],
 191:        [{'kid': 4, 'position': {x: 0, y: 20, z: -67}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -67}}],
 192:        [{'kid': 4, 'position': {x: 0, y: 20, z: -68}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -68}}],
 193:        [{'kid': 4, 'position': {x: 0, y: 20, z: -69}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -69}}],
 194:        [{'kid': 4, 'position': {x: 0, y: 20, z: -70}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -70}}],
 195:        [{'kid': 4, 'position': {x: 0, y: 20, z: -71}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -71}}],
 196:        [{'kid': 4, 'position': {x: 0, y: 20, z: -72}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -72}}],
 197:        [{'kid': 4, 'position': {x: 0, y: 20, z: -73}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -73}}],
 198:        [{'kid': 4, 'position': {x: 0, y: 20, z: -74}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -74}}],
 199:        [{'kid': 4, 'position': {x: 0, y: 20, z: -75}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -75}}],
 200:        [{'kid': 4, 'position': {x: 0, y: 20, z: -76}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -76}}],
 201:        [{'kid': 4, 'position': {x: 0, y: 20, z: -77}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -77}}],
 202:        [{'kid': 4, 'position': {x: 0, y: 20, z: -78}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -78}}],
 203:        [{'kid': 4, 'position': {x: 0, y: 20, z: -79}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -79}}],
 204:        [{'kid': 4, 'position': {x: 0, y: 20, z: -80}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -80}}],
 205:        [{'kid': 4, 'position': {x: 0, y: 20, z: -81}}, {'kid': 5, 'position
```

```
': {x: 0, y: 20, z: -81}}],
 206:        [{'kid': 4, 'position': {x: 0, y: 20, z: -82}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -82}}],
 207:        [{'kid': 4, 'position': {x: 0, y: 20, z: -83}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -83}}],
 208:        [{'kid': 4, 'position': {x: 0, y: 20, z: -84}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -84}}],
 209:        [{'kid': 4, 'position': {x: 0, y: 20, z: -85}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -85}}],
 210:        [{'kid': 4, 'position': {x: 0, y: 20, z: -86}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -86}}],
 211:        [{'kid': 4, 'position': {x: 0, y: 20, z: -87}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -87}}],
 212:        [{'kid': 4, 'position': {x: 0, y: 20, z: -88}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -88}}],
 213:        [{'kid': 4, 'position': {x: 0, y: 20, z: -89}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -89}}],
 214:        [{'kid': 4, 'position': {x: 0, y: 20, z: -90}}, {'kid': 5, 'position
': {x: 0, y: 20, z: -90}}]
 215:      ],
 216:      // ここに上記に習ってオリジナルモーションを作ることができます(サンプルの中では未
使用)
 217:      'original': [
 218:        [{'kid': 0, 'position': {}}]
 219:      ],
 220:    },
 221:
 222:    // Class関数定義
 223:
 224:    /**
 225:     * ロボットに接続された時に呼び出されるメソッド
 226:     * (このサンプルではGameManager側でオーバーライドされる)
 227:     */
 228:    onConnected: function() {
 229:    },
 230:
 231:    /**
 232:     * 接続状態を返す
 233:     * @return {Boolean} 接続していればTrue
 234:     */
 235:    isConnected: function() {
 236:      return this.connected;
 237:    },
 238:
 239:    /**
 240:     * カスタムモーションを再生する状態にする
 241:     * ※実際のモーション再生はtakeActionのメソッドの中で他の処理との優先順位を見なが
ら実行される
 242:     * @param {String} motion モーション名
 243:     */
 244:    activateMotion: function(motionName) {
 245:      if ((motionName in this.motionState) && !(this.motionState[motionNam
e].active)) {
 246:        this.motionState[motionName].currentFrame = 0;
 247:        this.motionState[motionName].active = true;
 248:        this.processInUse.motion = true;
 249:      }
 250:   },
 251:
```

```
252:     /**
253:      * カメラアングルの変更
254:      * @param {Number} angle カメラの左右回転角度
255:      */
256:     setCameraAngle: function(angle) {
257:       this.cameraAngle = angle;
258:       this.processInUse.camera = true;
259:     },
260:
261:     /**
262:      * カメラアングルをゼロ位置に戻す
263:      */
264:     resetCameraAngle: function() {
265:       this.cameraAngle = 0;
266:       this.processInUse.camera = true;
267:     },
268:
269:     /**
270:      * 歩行パラメータのセット
271:      * @param {Number} forward 前後方向の程度
272:      * @param {Number} turnCW 左右回転方向の程度
273:      */
274:     setWalkParameters: function(forward, turnCW) {
275:       this.walkParameters.forward = forward;
276:       this.walkParameters.turnCW = turnCW;
277:       this.processInUse.walk = true;
278:     },
279:
280:     /**
281:      * 歩行パラメータをリセットし、歩行を止める
282:      */
283:     stopWalking: function() {
284:       this.walkParameters.forward = 0;
285:       this.walkParameters.turnCW = 0;
286:       this.processInUse.walk = false;
287:     },
288:
289:     /**
290:      * ロボットに行動指示を送るメインループをスタートする
291:      */
292:     startMainLoop: function() {
293:       this.mainLoop = setInterval(this.mainLoop_.bind(this), Math.round(10
00 / this.fps));
294:     },
295:
296:     // 以下Private method
297:
298:     // 各プロセスの利用状態を保持するための配列生成
299:     initProcessInUse_: function() {
300:       var processInUse = {};
301:       for (var process in this.PROCESS_TYPE) {
302:         processInUse[process] = false;
303:       }
304:       return processInUse;
305:     },
306:
307:     // モーションの管理のための配列生成
308:     initMotionState_: function() {
309:       var motionState = {};
```

```
 310:       for (var motion in this.CUSTOM_MOTION) {
 311:         motionState[motion] = {'active': false, 'currentFrame': 0};
 312:       }
 313:       return motionState;
 314:     },
 315:
 316:     // 何か一つでもモーションを実行しているか確認
 317:     isMotionActive_: function() {
 318:       var inUse = false;
 319:       for (var motion in this.CUSTOM_MOTION) {
 320:         inUse |= this.motionState[motion].active;
 321:       }
 322:       return inUse;
 323:     },
 324:
 325:     // ロボットへのコマンド送信
 326:     sendCommand_: function (message, callback) {
 327:       this.waitForConnection_(function () {
 328:         this.connect.send(message);
 329:         if (typeof callback !== 'undefined') {
 330:           callback();
 331:         }
 332:       }.bind(this), 5);
 333:     },
 334:
 335:     // WebSocketの接続を待つ
 336:     waitForConnection_: function (callback, interval) {
 337:       if (this.connect.ws.readyState === 1) {
 338:         callback();
 339:       } else {
 340:         var that = this;
 341:         setTimeout(function () {
 342:           that.waitForConnection_(callback, interval);
 343:         }, interval);
 344:       }
 345:     },
 346:     // ロボットへの行動指示コマンドを作成するメインループ
 347:     mainLoop_: function() {
 348:       // 歩行処理、カスタムモーション処理、カメラ処理のいずれかを順次行う
 349:
 350:       var command = null;
 351:
 352:       // 前の処理の次の種類の処理を調べる
 353:       var currentProcess = this.lastProcess + 1;
 354:       if (currentProcess >= this.PROCESS_TYPE.length) {
 355:         currentProcess = 0;
 356:       }
 357:       // 次の実行すべき処理が発見されるか、前の処理と同じになるまで次々と進む
 358:       while (!this.processInUse[this.PROCESS_TYPE[currentProcess]] && curr
entProcess !== this.lastProcess) {
 359:         currentProcess++;
 360:         if (currentProcess >= this.PROCESS_TYPE.length) {
 361:           currentProcess = 0;
 362:         }
 363:       }
 364:       if (this.processInUse[this.PROCESS_TYPE[currentProcess]]) {
 365:         switch (this.PROCESS_TYPE[currentProcess]) {
 366:           // 歩行処理
 367:           case 'walk':
```

```
 368:           this.processInUse.walk = false;
 369:           // 歩行コマンドの生成
 370:           command = new vsido.Walk(this.walkParameters.forward, this.walk
Parameters.turnCW);
 371:           break;
 372:         // カスタムモーション処理
 373:         case 'motion':
 374:           // 念のため、動かすべきモーションがあるかどうかの確認
 375:           if (this.isMotionActive_()) {
 376:             // IKコマンドの生成（position情報を利用するので、追加が必要）
 377:             command = new vsido.SetIK({'position': true});
 378:             // motionStateの各モーションについてアクティブ状態かチェックする
 379:             for (var motion in this.motionState) {
 380:               if (this.motionState[motion].active) {
 381:                 // アクティブなモーションについては、現在のフレームを再生する
 382:                 if (this.motionState[motion].currentFrame < this.CUSTOM_MO
TION[motion].length) {
 383:                   // フレームに含まれる全てのIK情報を追加する
 384:                   var frameData = this.CUSTOM_MOTION[motion][this.motionSt
ate[motion].currentFrame];
 385:                   for (var i = 0; i < frameData.length; i++) {
 386:                     // IKコマンドのposition情報追加
 387:                     command.setPosition(frameData[i].kid, frameData[i].pos
ition.x, frameData[i].position.y, frameData[i].position.z);
 388:                   }
 389:                 }
 390:                 // 次のループのために次のフレームにすすめ、もし次のフレームがなければ
そのモーションの再生は終了
 391:                 this.motionState[motion].currentFrame++;
 392:                 if (this.motionState[motion].currentFrame >= this.CUSTOM_M
OTION[motion].length) {
 393:                   // モーションを最後のフレームまで再生したら、モーションのフラグを落
とし、現在のフレームを0に戻す
 394:                   this.motionState[motion].active = false;
 395:                   this.motionState[motion].currentFrame = 0;
 396:                 }
 397:               }
 398:             }
 399:           }
 400:           // 全てのモーションの全てのフレームの再生が終わった場合、motionのフラグを
落とす
 401:           this.processInUse.motion = this.isMotionActive_();
 402:           break;
 403:         // カメラ処理
 404:         case 'camera':
 405:           this.processInUse.camera = false;
 406:           // IKコマンドの生成（rotation情報を利用するので、追加が必要）
 407:           command = new vsido.SetIK({'rotation': true});
 408:           // IKコマンドのrotation情報追加
 409:           command.setRotation('head', 0, 0, this.cameraAngle);
 410:           break;
 411:       }
 412:       // ロボットにコマンドを送信する
 413:       if (command !== null) {
 414:         if (this.isDummyConnection) {
 415:           // ダミー接続モードの場合はコマンドを送る代わりにコンソールに表示する
 416:           console.log(command);
 417:         } else {
 418:           this.sendCommand_(command);
```

```
419:           }
420:         }
421:       }
422:       // 今回処理したプロセス種別の保持
423:       this.lastProcess = currentProcess;
424:     },
425:   };
426:
427:
428:   /**
429:    * 画面表示関連Class
430:    * @constructor
431:    */
432:   var DisplayManager = function() {
433:
434:     // 変数初期化
435:
436:     /**
437:      * 画面書き換えのfps
438:      * @type {Object}
439:      */
440:     this.fps = 60;
441:
442:     /**
443:      * マーカー表示用
444:      * @type {Object}
445:      */
446:     this.marker = {
447:       x: 0,
448:       y: 0,
449:     };
450:
451:     /**
452:      * 攻撃エフェクトのcanvasを追加していく親要素
453:      * @type {Object}
454:      */
455:     this.attackEffect = document.getElementById('attack-effect');
456:
457:   };
458:   DisplayManager.prototype = {
459:     // Class定数定義
460:
461:     // カメラ画像サイズ
462:     CAMERA_VIDEO: {
463:       WIDTH: 160,
464:       HEIGHT: 120,
465:     },
466:
467:     // 攻撃エフェクト関連定義
468:     ATTACK_EFFECT: {
469:       DELAY: 150,
470:       COLOR: '#FFFF00',
471:     },
472:
473:     // カラーを利用したエフェクトの定義
474:     COLOR_EFFECT: {
475:       damage: {
476:         COLOR: '#FF0000',
477:         DELAY: 500,
```

```
478:         FADE_OUT: 500,
479:       },
480:     },
481:
482:     // マスク画像を用いるエフェクトの定義
483:     MASK_EFFECT: {
484:       knockout: {
485:         URL: 'url(./assets/images/damage.gif)',
486:         DELAY: 10000,
487:         FADE_OUT: 3000,
488:       },
489:     },
490:
491:     // Class関数定義
492:
493:     /**
494:      * 攻撃エフェクトのサイズのリセット(画面回転時などに実行)
495:      */
496:     resetCanvasSize: function() {
497:       this.attackEffect.width = window.innerWidth;
498:       this.attackEffect.height = window.innerHeight;
499:     },
500:
501:     /**
502:      * システムメッセージを一定時間表示する
503:      * @param {String} message 表示したいメッセージ
504:      * @param {String} message 表示したいメッセージ
505:      */
506:     showSystemMessage: function(message, ms) {
507:       this.setSystemMessage(message);
508:       var messageTimeout = setTimeout(function() {
509:         this.clearSystemMessage();
510:       }.bind(this), ms);
511:     },
512:
513:     /**
514:      * システムメッセージを設定する(自動的には消えない)
515:      * @param {String} message 表示したいメッセージ
516:      */
517:     setSystemMessage: function(message) {
518:       document.getElementById('system-message').innerText = message;
519:     },
520:
521:     /**
522:      * システムメッセージを消去する
523:      */
524:     clearSystemMessage: function() {
525:       this.setSystemMessage('');
526:     },
527:
528:     /**
529:      * 右上のステイタスメーターの値をセットする
530:      * @param {Number} type 表示したいメッセージ
531:      * @param {Number} value 表示したいメッセージ
532:      */
533:     setMeterValue: function(type, value) {
534:       document.getElementById(type + '-bar').style.width = value + 'px';
535:     },
536:
```

```
 537:     /**
 538:      * ターゲットマーカー位置をセットする
 539:      * @param {Number} x ターゲットのx座標値
 540:      * @param {Number} y ターゲットのy座標値
 541:      * @param {Boolean} direct
 542:      */
 543:     setTargetPosition: function(x, y) {
 544:       this.marker.x = x;
 545:       this.marker.y = y;
 546:     },
 547:
 548:     /**
 549:      * ターゲット位置をリセットする(画面センターに)
 550:      */
 551:     resetTargetPosition: function() {
 552:       // マーカーの座標は元カメラ画像のサイズ160x120の範囲で送られてくる
 553:       this.setTargetPosition(this.CAMERA_VIDEO.WIDTH / 2, this.CAMERA_VIDE
O.HEIGHT / 2);
 554:     },
 555:
 556:     /**
 557:      * 攻撃エフェクト
 558:      */
 559:     setAttackEffect: function() {
 560:       // 現在時刻を取得
 561:       var nowDate = new Date();
 562:       var beginTime = nowDate.getTime();
 563:       // 時刻データを元にしたユニークなidをつけたcanvas要素を制裁
 564:       var canvasId = "attack-effect-" + beginTime;
 565:       var attackCanvas = document.createElement('canvas');
 566:       attackCanvas.id = canvasId;
 567:       attackCanvas.width = this.attackEffect.width;
 568:       attackCanvas.height = this.attackEffect.height;
 569:       // canvas要素を追加する
 570:       this.attackEffect.appendChild(attackCanvas);
 571:       // canvasのコンテクストを取得
 572:       var canvasContext = attackCanvas.getContext('2d');
 573:       // 目標地点はマーカー位置
 574:       var targetX = attackCanvas.width * (this.marker.x / 160);
 575:       var targetY = attackCanvas.height * (this.marker.y / 120);
 576:       // スタート位置
 577:       var startPositionLX = attackCanvas.width / 4;
 578:       var startPositionRX = attackCanvas.width / 4 * 3;
 579:       var startPositionY = attackCanvas.height;
 580:       var attackEffectCount = 1;
 581:       // 画面書き換えのタイミングに合わせて攻撃エフェクトを描画
 582:       var attackEccectInterval = setInterval(function(){
 583:         var elapsedTime = new Date() - beginTime;
 584:         canvasContext.clearRect(0, 0, attackCanvas.width, attackCanvas.hei
ght);
 585:         canvasContext.globalAlpha = 1.0;
 586:         if (elapsedTime >= this.ATTACK_EFFECT.DELAY) {
 587:           this.attackEffect.removeChild(attackCanvas);
 588:           clearInterval(attackEccectInterval);
 589:         } else {
 590:           canvasContext.beginPath();
 591:           var frameRatio = attackEffectCount / (this.fps * this.ATTACK_EFFE
CT.DELAY / 1000);
 592:           var radius = 60 * (1.0 - frameRatio);
```

```
 593:          var shotPositionLX = startPositionLX + (targetX - startPositionLX
) * frameRatio
 594:          var shotPositionRX = startPositionRX + (targetX - startPositionRX
) * frameRatio
 595:          var shotPositionY = startPositionY + (targetY - startPositionY) *
frameRatio
 596:          canvasContext.arc(shotPositionLX, shotPositionY, radius, 2 * Math
.PI, false);
 597:          canvasContext.arc(shotPositionRX, shotPositionY, radius, 2 * Math
.PI, false);
 598:          canvasContext.fillStyle = 'yellow';
 599:          canvasContext.lineWidth = 10;
 600:          canvasContext.fill();
 601:          canvasContext.closePath();
 602:        }
 603:        attackEffectCount++;
 604:      }.bind(this), 1000 / this.fps);
 605:    },
 606:
 607:    /**
 608:     * 画像マスクを用いたエフェクトを有効にする
 609:     * @param {String} effectName
 610:     */
 611:    setMaskEffect: function(effectName) {
 612:      var maskEffect = document.getElementById('mask-effect');
 613:      if (effectName in this.MASK_EFFECT) {
 614:        // エフェクトの設定で指定されたURLの画像に差し替え
 615:        maskEffect.style.backgroundImage = this.MASK_EFFECT[effectName].URL;
 616:        maskEffect.style.opacity = 1.0;
 617:        maskEffect.style.display = 'inline';
 618:        setTimeout(function() {
 619:          // まずはエフェクトの期待する持続時間を待ってからフェードアウト処理を開始する
 620:          var beginTime = new Date() - 0;
 621:          var fadeout = setInterval(function(){
 622:            var elapsedTime = new Date() - beginTime;
 623:            if (elapsedTime >= this.MASK_EFFECT[effectName].FADE_OUT) {
 624:              // 指定された時間が過ぎたら、マスク画像の表示をなくし、インターバルタイマーをクリアする
 625:              maskEffect.style.display = 'none';
 626:              maskEffect.style.opacity = 1.0;
 627:              maskEffect.style.backgroundImage = '#';
 628:              clearInterval(fadeout);
 629:            } else {
 630:              // 時間に応じて不透明度をコントロールする
 631:              maskEffect.style.opacity = 1.0 - (elapsedTime / this.MASK_EFFECT[effectName].FADE_OUT);
 632:            }
 633:          }.bind(this), 1000 / this.fps);
 634:        }.bind(this), this.MASK_EFFECT[effectName].DELAY);
 635:      }
 636:    },
 637:
 638:    /**
 639:     * 色のエフェクトを有効にする
 640:     * @param {String} effectName
 641:     */
 642:    setColorEffect: function(effectName) {
 643:      var colorEffect = document.getElementById('color-effect');
```

```
644:       if (effectName in this.COLOR_EFFECT) {
645:         colorEffect.style.backgroundColor = this.COLOR_EFFECT[effectName].
COLOR;
646:         colorEffect.style.opacity = 1.0;
647:         colorEffect.style.display = 'inline';
648:         setTimeout(function() {
649:           // まずはエフェクトの期待する持続時間を待ってからフェードアウト処理を開始する
650:           var beginTime = new Date() - 0;
651:           var fadeout = setInterval(function(){
652:             var elapsedTime = new Date() - beginTime;
653:             if (elapsedTime >= this.COLOR_EFFECT[effectName].FADE_OUT) {
654:               // 指定された時間が過ぎたら、色の表示をなくし、インターバルタイマーをクリ
アする
655:               colorEffect.style.display = 'none';
656:               colorEffect.style.opacity = 1.0;
657:               colorEffect.style.backgroundColor = '#000000';
658:               clearInterval(fadeout);
659:             } else {
660:               // 時間に応じて不透明度をコントロールする
661:               colorEffect.style.opacity = 1.0 - (elapsedTime / this.COLOR_EF
FECT[effectName].FADE_OUT);
662:             }
663:           }.bind(this), 1000 / this.fps);
664:         }.bind(this), this.COLOR_EFFECT[effectName].DELAY);
665:       }
666:     },
667:   };
668:
669:
670:   /**
671:    * ゲーム全体を管理するメインClass
672:    * @constructor
673:    */
674:   var GameManager = function() {
675:
676:     /**
677:      * 秒間どのくらいエネルギーチャージするか
678:      * @type {Number}
679:      */
680:     this.rechargePerSecond = 10;
681:
682:     /**
683:      * パワーチャージのインターバル処理用
684:      * @type {Object}
685:      */
686:     this.chargeLoop = null;
687:
688:     /**
689:      * 画面の回転に対応するかを決定する変数
690:      * @type {Boolean}
691:      */
692:     this.bindOrientationActive = false;
693:
694:     /**
695:      * ロボットのゲーム内でのステイタス管理のための変数
696:      * @type {Object}
697:      */
698:     this.myStatus = {
699:       shield: this.STATUS_MAX_VALUE.SHIELD,
```

```
700:       power: this.STATUS_MAX_VALUE.POWER,
701:     };
702:
703:     /**
704:      * ロボット制御のインスタンスのための変数（ここではまだインスタンスを生成しない）
705:      * @type {Object}
706:      */
707:     this.robot = null;
708:
709:     /**
710:      * カメラのインスタンスのための変数（ここではまだインスタンスを生成しない）
711:      * @type {Object}
712:      */
713:     this.camera = null;
714:
715:     /**
716:      * ロボット間通信のインスタンスのための変数（ここではまだインスタンスを生成しない）
717:      * @type {Object}
718:      */
719:     this.r2r = null;
720:
721:     /**
722:      * ゲームのステイタス管理の変数
723:      * @type {Object}
724:      */
725:     this.gameStatus = {
726:       active: true,
727:     };
728:
729:     /**
730:      * 画面描画のインスタンスの生成
731:      * @type {Object}
732:      */
733:     this.display = new DisplayManager();
734:
735:     // 画面の方向変化に対応する
736:     this.enableBindOrientation_();
737:
738:     // マウスダウン状態保持
739:     this.mousedownflag = false;
740:
741:     // 縦スクロールしないようにDocument全体のtouchmoveイベントを無視するよう設定
742:     document.addEventListener('touchmove', function(e) {
743:       e.preventDefault();
744:     });
745:
746:     this.debug = new Debug();
747:
748:     /* 画面の回転、表示などに関するイベント処理設定 */
749:     window.addEventListener('orientationchange', this.orientationchangeHa
ndler_.bind(this));
750:     window.addEventListener('pageshow', this.pageshowHandler_.bind(this))
;
751:     window.addEventListener('pagehide', this.pagehideHandler_.bind(this))
;
752:     window.addEventListener('resize', this.resizeHandler_.bind(this));
753:   };
754:   GameManager.prototype = {
755:     // Class定数定義
```

```
 756:
 757:     // システムメッセージ用
 758:     TEXT: {
 759:       WIN: "YOU WIN!!!",
 760:       LOSE: "YOU LOSE!!!",
 761:       RESET: ""
 762:     },
 763:
 764:     // 各種デバイスの制限値 //
 765:     DEVICE: {
 766:       MAX_ROTATION: 60,
 767:       MAX_CAMERA_ROTATION: 30,
 768:       MINUS_CAMERA_ORIGIN: 300,
 769:       CAMERA_ROTATION_DELIMITER: Math.sin(59 * Math.PI / 180),
 770:     },
 771:
 772:     // 自機ステイタスの上限値
 773:     STATUS_MAX_VALUE: {
 774:       SHIELD: 150,
 775:       POWER: 150,
 776:     },
 777:
 778:     // 攻撃関連パラメータ
 779:     ATTACK: {
 780:       POWER: 20,
 781:       MIN_DAMAGE: 10,
 782:       MIN_DISTANCE: 4,
 783:     },
 784:
 785:     // r2rのメッセージタイプ
 786:     R2R_MESSAGE_TYPE: {
 787:       NONE: 0,
 788:       ATTACK: 1,
 789:       WIN: 2,
 790:       RESPAWN: 3,
 791:     },
 792:
 793:     // Class関数定義
 794:
 795:     /**
 796:      * ロボットやカメラへの接続を試みる
 797:      */
 798:     connectRobots: function() {
 799:       // フォームから接続先を抜き出す
 800:       var robot_ip = document.getElementById('robot-ip').value;
 801:       var enemy_ip = document.getElementById('enemy-ip').value;
 802:
 803:       // 自ロボットへの接続
 804:       try{
 805:         this.robot = new RobotController({'ip': robot_ip});
 806:
 807:         // ロボット接続のインスタンスのコールバック関数のオーバーライド
 808:         this.robot.onConnected = function(event) {
 809:
 810:           // カメラ接続
 811:           this.camera = new Camera({'ip': robot_ip});
 812:           this.camera.listen(this.remoteCallback);
 813:           this.camera.viewMarkerDetect(document.getElementById('camera-ima
ge'));
```

```
 814:           // ゲームを開始する
 815:           this.robot.startMainLoop();
 816:           this.gameStart();
 817:         }.bind(this);
 818:
 819:         // R2R通信接続
 820:         if ((robot_ip === 'dummy') || (enemy_ip === 'dummy')) {
 821:           // デバグモードの場合は接続させない
 822:         } else {
 823:           // 対戦ロボットとのr2r通信の接続
 824:           this.r2r = new BattleR2R({'i': robot_ip, 'you': enemy_ip}, this.r
2rRemoteCallback_.bind(this)); // 対戦相手のip
 825:         }
 826:
 827:         // ダミー接続の場合、接続した後のイベントが発生しないのでここで強引に起こす。
 828:         if (robot_ip === 'dummy') {
 829:           document.getElementById('camera-image').src = './assets/images/du
mmy.jpg';
 830:           this.gameStart();
 831:         }
 832:       }
 833:       catch(e) {
 834:         alert('ロボットに接続できませんでした');
 835:       }
 836:     },
 837:
 838:     /**
 839:      * ゲーム開始(ロボットなどへの接続開始後)
 840:      */
 841:     gameStart: function() {
 842:
 843:       // 接続ダイアログを消す
 844:       document.getElementById('connection-dialog').style.display = 'none';
 845:
 846:       // ステイタスメーターの初期化
 847:       this.resetMyStatus_();
 848:
 849:       // ゲームUIを表示する
 850:       this.enableGameUI_();
 851:
 852:       // 画面の向きの処理
 853:       if (window.orientation === 0) {
 854:         this.display.resetCanvasSize();
 855:       }
 856:       if (this.bindOrientationActive) {
 857:         this.bindOrientation_();
 858:       }
 859:
 860:       // カメラアングルのリセット
 861:       this.robot.resetCameraAngle();
 862:
 863:       // ロボットを初期姿勢にする
 864:       this.robot.activateMotion('lowerhands');
 865:
 866:       // ゲーム開始メッセージ
 867:       this.display.showSystemMessage('GAME START', 1000);
 868:
 869:       // 体力回復を指定たされたrechargePerSecondで回す
 870:       this.chargeLoop = setInterval(this.rechargePower_.bind(this), Math.r
```

```
ound(1000 / this.rechargePerSecond));
 871:     },
 872:
 873:     // 画面の向きが変わった時に処理を行うように設定する(実際の処理はイベントハンドラ内
)
 874:     enableBindOrientation_: function() {
 875:       this.bindOrientationActive = true;
 876:     },
 877:
 878:     /**
 879:      * 画面の向きが変わった時に処理を行わないように設定する(実際の処理はイベントハンド
ラ内)
 880:      */
 881:     disableBindOrientation_: function() {
 882:       this.bindOrientationActive = false;
 883:     },
 884:
 885:     // ゲームUIを表示させ、イベントを処理できるようにする
 886:     enableGameUI_: function() {
 887:       // ゲームUIの表示
 888:       document.getElementById('game-ui').style.display = 'inline';
 889:
 890:       // コントローラーのためのイベント処理定義
 891:       document.getElementById('game-movement').addEventListener('mousedown
', this.movementTouchstartHandler_.bind(this));
 892:       document.getElementById('game-movement').addEventListener('touchstar
t', this.movementTouchstartHandler_.bind(this));
 893:
 894:       document.getElementById('game-movement').addEventListener('mousemove
', this.movementTouchmoveHandler_.bind(this));
 895:       document.getElementById('game-movement').addEventListener('touchmove
', this.movementTouchmoveHandler_.bind(this));
 896:
 897:       document.getElementById('game-movement').addEventListener('mouseup',
 this.movementTouchendHandler_.bind(this));
 898:       document.addEventListener('mouseup', this.documentMouseupHandler_.bi
nd(this));
 899:       document.getElementById('game-movement').addEventListener('touchend'
, this.movementTouchendHandler_.bind(this));
 900:
 901:       document.getElementById('game-fire').addEventListener('mousedown', t
his.fireTouchstartHandler_.bind(this));
 902:       document.getElementById('game-fire').addEventListener('touchstart',
this.fireTouchstartHandler_.bind(this));
 903:     },
 904:
 905:     // 画面の向きが変更になった場合の処理
 906:     orientationchangeHandler_: function(event) {
 907:       this.display.resetCanvasSize();
 908:       this.display.resetTargetPosition();
 909:       this.robot.resetCameraAngle();
 910:     },
 911:
 912:     // 画面が(隠された後)表示された場合の処理
 913:     pageshowHandler_: function(event) {
 914:       this.display.resetCanvasSize();
 915:       this.display.resetTargetPosition();
 916:     },
 917:
```

```
 918:     // 画面が隠された場合の処理
 919:     pagehideHandler_: function(event) {
 920:       if (this.bindOrientationActive) {
 921:         this.unbindOrientation_();
 922:         this.bindOrientation_();
 923:       }
 924:     },
 925:
 926:     // 画面サイズが変更になった場合の処理
 927:     resizeHandler_: function(event) {
 928:       if (this.bindOrientationActive) {
 929:         this.unbindOrientation_();
 930:         this.bindOrientation_();
 931:       }
 932:     },
 933:
 934:     // デバイスの向き変更イベントを処理へバインドする
 935:     bindOrientation_: function() {
 936:       window.addEventListener('deviceorientation', this.orientationControl
_.bind(this));
 937:     },
 938:
 939:     // デバイスの向き変更イベントの処理をしないようにする
 940:     unbindOrientation_: function() {
 941:       window.removeEventListener('deviceorientation', this.orientationCont
rol_.bind(this));
 942:     },
 943:
 944:     // 移動UIのtouchstartイベントの処理用
 945:     movementTouchstartHandler_: function(event) {
 946:       var elementPosition = document.getElementById('game-movement').getBo
undingClientRect();
 947:       var x = 0;
 948:       var y = 0;
 949:       var forward = 0;
 950:       var turnCW = 0;
 951:       var touch = false;
 952:       if (event.type == 'mousedown') {
 953:         y = event.offsetY;
 954:         x = event.offsetX;
 955:       } else {
 956:         y = event.touches[0].pageY - elementPosition.top;
 957:         x = event.touches[0].pageX - elementPosition.left;
 958:         touch = true;
 959:       }
 960:       forward = 99 - Math.round((y / document.getElementById('game-movement
').clientHeight) * 200);
 961:       turnCW = Math.round((x / document.getElementById('game-movement').cl
ientWidth) * 200) - 99;
 962:
 963:       // ロボットへの歩行指示
 964:       this.robot.setWalkParameters(forward, turnCW, touch);
 965:       this.mousedownflag = true;
 966:     },
 967:
 968:     // 移動UIのtouchmoveイベントの処理用
 969:     movementTouchmoveHandler_: function(event) {
 970:       var elementPosition = document.getElementById('game-movement').getBo
undingClientRect();
```

```
 971:       var x = 0;
 972:       var y = 0;
 973:       var forward = 0;
 974:       var turnCW = 0;
 975:       var touch = false;
 976:       if (event.type == 'mousemove') {
 977:         if (!this.mousedownflag) {
 978:           return;
 979:         }
 980:         y = event.offsetY;
 981:         x = event.offsetX;
 982:       } else {
 983:         y = event.touches[0].pageY - elementPosition.top;
 984:         x = event.touches[0].pageX - elementPosition.left;
 985:         touch = true;
 986:       }
 987:       forward = 99 - Math.round((y / document.getElementById('game-movement
').clientHeight) * 200);
 988:       turnCW = Math.round((x / document.getElementById('game-movement').cl
ientWidth) * 200) - 99;
 989:
 990:       this.robot.setWalkParameters(forward, turnCW, touch);
 991:     },
 992:
 993:     // 移動UIのtouchendイベントの処理用
 994:     movementTouchendHandler_: function(event) {
 995:       this.robot.stopWalking();
 996:       this.mousedownflag = false;
 997:     },
 998:
 999:     // 移動UIのtouchendイベントの処理用
1000:     documentMouseupHandler_: function(event) {
1001:       if (this.mousedownflag) {
1002:         this.robot.stopWalking();
1003:         this.mousedownflag = false;
1004:       }
1005:     },
1006:
1007:     // 攻撃UIのtouchstartイベントの処理用
1008:     fireTouchstartHandler_: function(event) {
1009:       this.attack_();
1010:     },
1011:
1012:     // 画面の向きの変更時の処理
1013:     orientationControl_: function(event) {
1014:       if (this.isPowerDown_()) {
1015:         return;
1016:       }
1017:       // 加速度が取れる場合は加速度のalpha値を使う（取れない時は0）
1018:       var px = null;
1019:       if (typeof event.alpha !== 'undefined') {
1020:         px = event.alpha;
1021:       } else {
1022:         px = 0;
1023:       }
1024:
1025:       // カメラの回転を計算し、ロボットに渡す。
1026:       if ((px > this.DEVICE.MAX_ROTATION) && (px <= 180)) {
1027:         px = this.DEVICE.MAX_ROTATION;
```

```
1028:       } else if ((px > 180) && (px < this.DEVICE.MINUS_CAMERA_ORIGIN)) {
1029:         px = this.DEVICE.MINUS_CAMERA_ORIGIN;
1030:       }
1031:       px = (Math.sin(px * Math.PI / 180) / (this.DEVICE.CAMERA_ROTATION_DEL
IMITER));
1032:       if (this.robot !== null) {
1033:         this.robot.setCameraAngle(-Math.floor(this.DEVICE.MAX_CAMERA_ROTAT
ION * px));
1034:       }
1035:     },
1036:
1037:     // パワーダウン(負け)かどうか確認する
1038:     isPowerDown_: function() {
1039:       return !this.gameStatus.active;
1040:     },
1041:
1042:     // 負けた後の復活の処理
1043:     respawn_: function() {
1044:       this.display.showSystemMessage(this.TEXT.RESET_MSG, 1000);
1045:       this.sendMessage_(this.R2R_MESSAGE_TYPE.RESPAWN);
1046:       this.gameStatus.active = true;
1047:       this.resetMyStatus_();
1048:       this.robot.activateMotion('lowerhands');
1049:       this.robot.activateMotion('standup');
1050:     },
1051:
1052:     // ステイタスの変更
1053:     setMyStatus_: function(type, value) {
1054:       this.myStatus[type] = value;
1055:       this.display.setMeterValue(type, value);
1056:     },
1057:
1058:     // ステイタスの初期化
1059:     resetMyStatus_: function() {
1060:       this.setMyStatus_('power', this.STATUS_MAX_VALUE.POWER);
1061:       this.setMyStatus_('shield', this.STATUS_MAX_VALUE.SHIELD);
1062:     },
1063:
1064:     // r2r通信でメッセージを受け取った時のコールバック処理
1065:     r2rRemoteCallback_: function(message) {
1066:       if (this.isPowerDown_()) {
1067:         return;
1068:       }
1069:
1070:       if ('marker' in message) {
1071:         var marker = message.marker;
1072:         if (marker) {
1073:           this.display.setTargetPosition(marker.x, marker.y);
1074:         }
1075:       }
1076:       if ('r2r' in message) {
1077:         var r2rMessage = message.r2r;
1078:         if (r2rMessage) {
1079:           var attack = r2rMessage.attack;
1080:           if (attack) {
1081:             var type = attack.type;
1082:             var data = attack.data;
1083:             var markerAttack = attack.marker;
1084:
```

```
1085:           switch (type) {
1086:             case this.R2R_MESSAGE_TYPE.ATTACK:
1087:               if (markerAttack) {
1088:                 if (this.ATTACK.MIN_DISTANCE <= markerAttack.distance) {
1089:                   this.recieveAttack_();
1090:                 }
1091:               }
1092:               break;
1093:             case this.R2R_MESSAGE_TYPE.WIN:
1094:               this.display.setSystemMessage(this.TEXT.WIN);
1095:               break;
1096:             case this.R2R_MESSAGE_TYPE.RESPAWN:
1097:               this.display.showSystemMessage(this.TEXT.RESET, 1000);
1098:               break;
1099:           }
1100:         }
1101:       }
1102:     }
1103:   },
1104:
1105:   // 攻撃処理
1106:   attack_: function() {
1107:     var power = this.myStatus.power - this.ATTACK.POWER;
1108:     if (power > 0) {
1109:       // 攻撃できるだけのパワーがあれば、攻撃処理を行う
1110:       this.sendMessage_(this.R2R_MESSAGE_TYPE.ATTACK);
1111:       this.setMyStatus_('power', power);
1112:       this.display.setAttackEffect();
1113:       //this.display.activateEffect(this.display.EFFECT_TYPE.ATTACK);
1114:       this.robot.activateMotion('attack');
1115:     }
1116:   },
1117:
1118:   // r2r通信でメッセージを相手に送る
1119:   sendMessage_: function(type) {
1120:     var message = {'attack': {'type': 0, 'data': 0}};
1121:     switch (type) {
1122:       case this.R2R_MESSAGE_TYPE.ATTACK:
1123:         message.attack.type = this.R2R_MESSAGE_TYPE.ATTACK;
1124:         break;
1125:       case this.R2R_MESSAGE_TYPE.WIN:
1126:         message.attack.type = this.R2R_MESSAGE_TYPE.WIN;
1127:         break;
1128:       case this.R2R_MESSAGE_TYPE.RESPAWN:
1129:         message.attack.type = this.R2R_MESSAGE_TYPE.RESPAWN;
1130:         break;
1131:     }
1132:     if (this.r2r !== null) {
1133:       this.r2r.send(message);
1134:     }
1135:   },
1136:
1137:   // 攻撃を受けた時の処理
1138:   recieveAttack_: function() {
1139:     var damage = Math.floor(this.ATTACK.POWER * (1 - this.myStatus.power
/ this.STATUS_MAX_VALUE.POWER)) + this.ATTACK.MIN_DAMAGE;
1140:     this.setMyStatus_('shield', this.myStatus.shield - damage);
1141:     if (this.myStatus.shield <= 0) {
1142:       this.gameStatus.active = false;
```

```
1143:         this.robot.stopWalking();
1144:         this.robot.activateMotion('crouch');
1145:         this.sendMessage_(this.R2R_MESSAGE_TYPE.WIN);
1146:         this.setMyStatus_('shield', 0);
1147:         this.display.setMaskEffect('knockout');
1148:         this.display.setSystemMessage(this.TEXT.LOSE);
1149:         setTimeout(function() {
1150:           this.respawn_();
1151:         }.bind(this), 13000);
1152:       } else {
1153:         //this.display.setColorEffect('damage');
1154:       }
1155:     },
1156:
1157:     // 体力回復
1158:     rechargePower_: function() {
1159:       if (this.isPowerDown_()) {
1160:         return;
1161:       }
1162:       var power = this.myStatus.power + 1;
1163:       if (power > this.STATUS_MAX_VALUE.POWER) {
1164:         power = this.STATUS_MAX_VALUE.POWER;
1165:       } else {
1166:         this.setMyStatus_('power', power);
1167:       }
1168:     }
1169:
1170:   };
1171:
1172:   /* デバグ用class */
1173:   var Debug = function() {
1174:     this.enable = true;
1175:     //$('#debug-reload').on('click', reloadpage);
1176:   };
1177:   Debug.prototype = {
1178:     reloadpage: function() {
1179:       location.reload(true);
1180:     },
1181:     print: function(text) {
1182:       document.getElementById('debug-print').textContent = text;
1183:     },
1184:   };
1185:
1186:   /* 各種処理メソッド用class */
1187:   var Utility = function() {
1188:   };
1189:   Utility.prototype = {
1190:     getUrlVars: function() {
1191:       var vars = [], max = 0, hash = "", array = "";
1192:       var url = window.location.search;
1193:
1194:       //?を取り除くため、1から始める。複数のクエリ文字列に対応するため、&で区切る
1195:       hash = url.slice(1).split('&');
1196:       max = hash.length;
1197:       for (var i = 0; i < max; i++) {
1198:         array = hash[i].split('=');    //keyと値に分割。
1199:         vars.push(array[0]);    //末尾にクエリ文字列のkeyを挿入。
1200:         vars[array[0]] = array[1];    //先ほど確保したkeyに、値を代入。
1201:       }
```

```
1202:       return vars;
1203:     }
1204:   };
1205:
1206:
1207:   /* 処理部 */
1208:   window.onload = function() {
1209:
1210:     // 念のため、ゲームUI接続ダイアログを隠す
1211:     document.getElementById('game-ui').style.display = 'none';
1212:     document.getElementById('connection-dialog').style.display = 'none';
1213:
1214:     // 便利な関数群を管理しているClassのインスタンス生成
1215:     var utility = new Utility();
1216:
1217:     // クエリパラメータ処理
1218:     var query_parameter = utility.getUrlVars();
1219:
1220:     // 引数にIPアドレスなどの指定があればそれを利用する、なければ現在のサイトのURLから
取得
1221:     if ('robotip' in query_parameter) {
1222:       document.getElementById('robot-ip').value = query_parameter.robotip;
1223:     } else {
1224:       document.getElementById('robot-ip').value = location.hostname;
1225:     }
1226:     if ('enemyip' in query_parameter) {
1227:       document.getElementById('enemy-ip').value = query_parameter.enemyip;
1228:     } else {
1229:       document.getElementById('enemy-ip').value = document.getElementById(
'robot-ip').value;
1230:     }
1231:     // 接続ダイアログの表示
1232:     document.getElementById('connection-dialog').style.display = 'inline'
;
1233:
1234:     // ゲームマネージャのインスタンス生成
1235:     var game = new GameManager();
1236:
1237:     // connectボタンの有効化
1238:     document.getElementById('connect-button').addEventListener('click', g
ame.connectRobots.bind(game), false);
1239:   };
1240:
1241: }());
```

Asratec

・ 記載された社名、製品名は一般に各社の商標または登録商標です。

「AR Roboto Battle for V-Sido CONNECT」
ユーザーマニュアルと開発の手引き

アスラテック株式会社
〒101-0042　東京都千代田区神田東松下町 45