プログラミング演習(JavaScript)③【実践】

ゲームを作るときに、時間を設定して一定の間 隔で動かしたり、アニメーションをさせたりすることがある。今回は動きのあるプログラムを作るときに使える方法を説明する。

11 一定の時間間隔で、位置の移動をする■オブジェクト指向プログラミング

■setTimeout clearTimeout

setTimeout でタイマー処理を実行し、 clearTimeout でタイマー処理を取り消す。 setTimeout は、 第2引数に与えられた時間 で、第1引数に指定した関数を実行するもの で 、 時間の単位はミリ秒となる。今回のプログラムは、位置が左から300を超える ま で の間、100 ミリ秒＝0.1 秒間隔 で 、位置 が 右 へ 20 移動させてお り、 clearTimeout は 使っていない 。

setTimeout(処理内容, 待ち時間)	待ち時間になると関数または指定されたコードの断片を実行する
clearTimeout(ID)	setTimeout() の呼び出しによって以前に確立されたタイムアウトを解除する

js11-1.htm　setTimeout()メソッドを使って画像を移動する

```
008   <img id="img1" style="position:absolute; top:50px;" src="js11ball.png">
009   <script>
010   window.onload=function(){
011         let gazo=document.getElementById("img1");
012         let x=50;
013     function move(){
014       gazo.style.left=x+"px";
015       x=x+20;
016       if(x<=300){
017         setTimeout(move,100);
018       }
019     }
020    move();
021   }
022   </script>
```

8行 CSS でレイアウトの設定をする

10行 読み込まれると実行する

11行 ID 名が「img1」の画像を変数 gazo に格納する

12 行 最初に左側からの距離を設定する

13～18行 move 関数

14行 gazo の左からの距離を x ピクセルにする

15行 xにx＋20を代入する

16行　300以下まで15行を繰り返す
17行　0.1秒待って move 関数を実行する。※距離が 300 以下の状態で再帰的に move 関数が実行される
18行　最初に move 関数を実行する

■ setInterval, clearInterval

ここでは、setInterval を使って画像を動かす。ボールをクリックしたときに、現在の時刻を取得して、2000 ミリ秒(2秒)超えるまで描画が繰り返し行われ、2秒超えたら clearInterval でタイマー処理を取り消している。

setInterval(関数 処理間隔)	一定の遅延間隔を置いて関数やコードスニペットを繰り返し呼び出します
clearInterval(ID)	以前に setInterval() の呼び出しによって確立されたタイマーを利用した繰り返し動作を取り消す

js11-2.htm　setInterval()メソッドを使って、画像を移動する

```
008   <img id="img2" style="position:relative; cursor: pointer; top:50px;"
      src="js11ball.png">
009   <script>
010     img2.onclick = function() {
011       let start = Date.now();
012       let timer = setInterval(function() {
013         let timePassed = Date.now() - start;
014         img2.style.left = timePassed / 5 + 'px';
015         if (timePassed > 2000) clearInterval(timer);
016       }, 20);
017     }
018   </script>
```

(参考)JAVASCRRIPT.INFO JavaScript アニメーション(2021 年 12 月 15 日)
https://ja.javascript.info/js-animation

8行　画像の設定をする　ID 名は img2　位置は相対的、カーソルはポインター、上から50ピクセルの場所に画像を表示する
10行　img2 の画像をクリックすると実行する。ここでは、クリックした後に実行する関数を、関数名をつけて外部に持たせずに、直接＝(イコール)で結んで記述している。
11行　現在の時刻を取得して変数 start に代入する
12～16行　timer に setInterval の内容を入れ、16行目までの処理を20ミリ秒(0.02 秒)間隔で実行する。
13行　timePassed に経過時間(現在の時刻から start を引いた値)を代入する

14行　img2 の左側の位置を timePassed を5で割った値にする
15行　timePassed が 2000 を超えたら（2秒を超えたら）、timer をクリアする

■キーで操作して動かす

ここでは、押したキーに合わせて画像を動かすプログラムを作る。まず、「キーが押された」とか、「マウスによってクリックされた」とか、「フォームで文字が入力された」といった、イベントが発生した時になんらからの処理をすることを「イベント処理」といい、addEventListener メソッドを使って、イベント処理を行うことができる。

addEventListener(種類, 関数, false);	ターゲットに特定のイベントが配信されるたびに呼び出される関数を設定する

第1引数はイベントの種類で、ここに「クリックした」とか「文字を入力した」といったイベントの種類を入れる。
第2引数は第1引数で指定したイベントが発生したときに実行する関数を入れる。
第3引数はイベントの伝搬の方式を指定する場所で、通常は false を指定する。

第1引数のイベントの種類（一部）

click	ボタンをクリックしたとき
mousemove	カーソルがターゲット内に移動したとき
mouseover	カーソルがターゲット内の上に重なったとき
mousedown	マウスのボタンを押したとき
mouseup	マウスのボタンを離したとき
mouseout	カーソルがターゲットから離れたとき
keypress	キーを押して離したとき
keydown	キーを押したとき
keyup	キーを離したとき

第2引数の注意点としては、外部の関数名を指定する場合は()をつけない。しているのではなく、関数自体を入れる書き方をする場合は()をつける。
参考　SAMURAI ENGINNER　https://www.sejuku.net/blog/57625
参考　Qiita【javascript】addEventListener イベントまとめ
https://qiita.com/whw3312/items/94a2bdf632ef77555579
Qiita　addEventListener type 一覧と各ブラウザ対応
https://qiita.com/mrpero/items/156968e3512d42fffc5e

js11-3.htm　キーを押したときに、押したキーの文字コードをポップアップで表示する。

```
010     addEventListener(”keydown”,keydownfunc);
011     function keydownfunc(event) {
012       alert(event.keyCode);
```

```
013        }
```

10行 キーが押されたときに keydownfunc メソッドを呼び出す

11～13行 keydownfunc 関数

11 行 event に押されたキーの情報が入る

12行 keyCode プロパティでキーコードを取得され、押されたキーの文字コードが alert でポップアップに表示される。なお、keyCode プロパティは非推奨です。

js11-4.htm js11-3 のプログラムで、矢印キーのキーコードを記録し、そのキーコードを条件にして、押したキーに合わせて、画像を移動させる。

```
006     <style>#ball4 {position: absolute; top: 0; left: 0;}</style>
007   </head>
008 <body>
009   <script>
010    document.write( '<img id="ball4" src="js11ball.png">' );
011     var y = 0;  //ボールの x 座標
012     var x = 0;  //ボールの y 座標
013     addEventListener( "keydown", keydownfunc );
014     function keydownfunc(event){
015       var key_code = event.keyCode;    //キーのコードを key_code に代入
016         if( key_code === 37 ) x = x - 50;   //「左ボタン」が押されたとき、x の値から 50 を引き算する(x -= 32)
017         if( key_code === 38 ) y = y - 50;   //「上ボタン」が押されたとき、y の値から 50 を引き算する
018         if( key_code === 39 ) x = x + 50;   //「右ボタン」が押されたとき、x の値に 50 を足し算する
019         if( key_code === 40 ) y = y + 50;   //「下ボタン」が押されたとき、y の値に 50 を足し算する
020           document.getElementById( 'ball4' ).style.left = x + "px";   //ボールの画像の x 座標を反映させる
021           document.getElementById( 'ball4' ).style.top = y + "px";   //ボールの画像の y 座標を反映させる
022     }
```

6 行 styleタグにボールのCSSの設定を入れる

10行 id名「ball4」のボールの画像を表示させる(top:0,left:0 なので、左上に表示させる)

11, 12行 ボールの位置を示す変数x、yの設定をする。

13行 キーが押されたときに keydownfunc 関数が実行される

14～22行 keydownfunc 関数

14行 押されたキーの情報を event に渡す

15行　キーの文字コードを key_code に代入する
16～19行　左:37　上:38　右:39　下:40のキーコードに合わせて、座標を増減する
20, 21行　ボールの左からの位置、上からの位置に変えて再描画する

12 CANVAS

CANVASは、ブラウザ上で図を描くための仕様で、動かすにはJavaScriptを使って、描画と再描画を繰り返す必要がある。

js12-1.htm CANVASで円と四角形を描く

```
008   <canvas id="canvas" width="600" height="600">canvas 要素をサポートしていません
009 </canvas>
010   <script>
011     const canvas = document.getElementById('canvas'); // canvas 要素への参照の取得
012     const ball = canvas.getContext('2d'); // コンテキストの取得
013     const square = canvas.getContext('2d'); // コンテキストの取得
014     /* 円のコンテキスト設定 */
015     ball.strokeStyle = '#000'; // 塗りつぶしは暗めの色
016     ball.fillStyle = '#f00'; // 線は赤色
017     ball.lineWidth = 1; // 線の幅は 5px
018     /* 円の描画 */
019     ball.beginPath(); // パスの初期化
020     ball.arc(50, 50, 30, 0, 2 * Math.PI); // (50, 50)の位置に半径 30px の円
021     ball.closePath(); // パスを閉じる
022     ball.fill(); // 軌跡の範囲を塗りつぶす
023     ball.stroke(); // 枠線を描く
024     /* 四角形のコンテキスト設定 */
025     square.strokeStyle = '#00f'; // 塗りつぶしは暗めの色
026     square.fillStyle = '#0ff'; // 線は赤色
027     square.lineWidth = 1; // 線の幅は 5px
028     /* 四角形の描画 */
029     square.beginPath(); // パスの初期化
030     square.fillRect(120, 20, 120,60); // (120, 30)の位置に幅 120、高さ 60 の長方形
031     square.strokeRect(120, 20, 120,60); // 枠線を描く
```

8行 CANVAS要素を追加して、範囲を幅600、高さ600ピクセルする。CANVASにサポートしていないブラウザを使っている場合の代替テキストをつけている。

10行 CANVAS要素が参照できるようにしている。

11, 12、行 コンテキストを取得して描画できるようにしている。

14～16行 コンテキストの設定

18～22行 円の描画

23～29行 四角形の描画

js12-2.htm CANVASで描いた円を動かす

js12－1のようにHTML上にCANVAS要素を作らずに、JavaScript側から createElement メソッドを使って CANVAS を作ってから円を描き、window.requestAnimationFrame()メソッドを使って、繰り返し処理をさせてアニメーションを実現している。

```
009       const canvas = document.createElement('canvas'); //CANVAS を作る
010       const ball = canvas.getContext('2d'); // コンテキストの取得
011       canvas.width=500;
012       canvas.height=500;
013       document.body.appendChild(canvas);
014     let x = 50;
015     let y = 50;
016     /* コンテキスト設定 */
017       ball.strokeStyle = '#000'; // 線の色は黒(#000000 と同意)
018       ball.lineWidth = 2; // 線の幅は 2px
019       ball.fillStyle = '#f00'; // 塗りつぶしは赤(#FF0000 と同意)
020       const circleSize =30; //円の半径は 30px
021 //メインループ
022 function draw() {
023     ball.clearRect(0, 0, canvas.width, canvas.height);    // 描画内容を消去する
024     // x の値を増やして、右に移動する
025       x = x + 1
026     // 求めた座標に円を描画する。
027       ball.beginPath();
028       ball.arc(x,y, circleSize, 0, 2 * Math.PI); // (x,y)の位置に半径 30px の円を描く
029       ball.fill(); // 軌跡の範囲を塗りつぶす
030       ball.stroke(); // 線を描く
031       window.requestAnimationFrame(draw);
032 }
033 window.requestAnimationFrame(draw);
```

9行 CANVAS 要素を作る

10行 コンテキストを取得する

11, 12行 CANVAS の幅と高さを設定する

13行 挿入した CANVAS を BODY のノードリストに登録する。

14、15行 円を描画する x、y座標の設定をする

17～20行 円の設定

22～31行 メインループの関数 draw

23行 描画されている内容を消す

24行 円を描画する座標を右にずらす

26～29行 座標に円を描画する

30行 draw の処理が 60fps の速さで繰り返される。

13　プログラムによる動的シミュレーション

HTML5の canvas 要素と JavaScript を組み合わせて、初速度、角度を決めて、リンゴを飛ばす。ブラウザに図形を描画するには、キャンバスオブジェクトとコンテキストオブジェクトを用いる。

js13-1.htm　放物運動のシミュレーション

```
008     <canvas id = "mycanvas" width = "1000" height = "500">
009     canvas に対応したブラウザを使ってください。
010   </canvas>
011   <script>
012     function draw(){
013 //  <A>一定時間ごとに実行させる関数の本体> ↓
014       if (0 < x && x < canvas.width && 0 < y && y < canvas.height ){
015         x = x + vx0 * dt;
016         var v1 = vy , v2 = vy - g * dt;
017         var r = 20;
018         y= y + (v1 + v2) /2.0 * dt;
019         vy = v2;
020         var msgv0 = '初速度:' + v0
021         var msgdeg = '角度  :' + degrees
022         var msgx = 'x:' + Math.round(x)
023         var msgy = 'y:' + Math.round(y)
024         t = t + dt;
025       }else{
026         x = x0, y = y0 , vy = vy0, t = t0;
027     }
028     context.clearRect(0, 0, canvas.width, canvas.height);
029     context.beginPath();
030     context.arc(x, canvas.height - y, r, 0 , 2*Math.PI, false);
031     context.moveTo(x,canvas.height - y - 30);
032     context.lineTo(x,canvas.height - y - 10);
033     context.closePath();
034     context.fillStyle = 'rgb(255,0,0)';
035     context.fill();
036     context.strokeStyle = 'rgb(0,0,0)';
037     context.lineWidth = 2;
038     context.stroke();
039     context.font = '18px serif';                    /* canvas に文字を表示する */
040     context.fillStyle = 'rgb(100,100,100)';         /* 文字を少し薄くした */
```

```
041        context.lineWidth = 0;                          /* context.strokeText にした時の線の太さ */
042        context.fillText(msgv0, 10, 25); /* context.strokeText(msgx, 10, 25); で文字は表示できる */
043        context.fillText(msgdeg, 10, 45);
044        context.fillText(msgx, 120, 25);
045        context.fillText(msgy, 120, 45);
046    // <A>↑
047        }
048        var canvas = document.getElementById('mycanvas');
049        canvas.style.border = "1px solid #888888";    /* canvas に枠線を付ける 線を薄くした */
050        var context = canvas.getContext('2d');
051        var timerID = setInterval('draw()',50);
052        canvas.width = 1000 , canvas.height = 500;
053    // <B>変数の初期値設定↓
054      var t0 = 0.0, dt = 0.1, g = 9.8;
055      var v0 = Number(prompt('初速度を入力してください',''));
056      var degrees = Number(prompt('角度°を入力してください',''));
057      var vx0 = v0 * Math.cos(degrees * Math.PI /180);
058      var vy0 = v0 * Math.sin(degrees * Math.PI /180);
059      var x0 = canvas.width / 4, y0 = 3 * canvas.height / 4 ;
060      var x = x0 , y = y0, vy = vy0, t = t0;
061    // <B>↑
```

8行　canvas 要素をブラウザ上に作成する。Id 名「mycanvas」、高さ500ピクセル、幅1000ピクセルと設定する

9行　canvas要素に対応していないブラウザにこの文字列が表示させる

12～47行　51行目で呼び出す物体の位置を計算して描画をする draw 関数の定義

　14行　x、yの位置が、canvasの範囲内なら

　15行　x座標の位置を移動($x(t+\Delta t)=x(t)+v_{x0}\Delta t$)

　16行　$v_1=v_y(t)$ ，$v_2=v_y(t)-g\Delta t$

　18行　$y(t+\Delta t)=y(t)+1/2(v_1+v_2)\Delta t$

　19行　$v_y(t+\Delta t)=v_2$

　20～24行　表示させる文字列をそれぞれ変数に代入する

　26行　範囲外に出た場合は初期に戻す

　28～38行　描画する物体の描画

　39～45行　文字の描画

48行　キャンバスの取得

49～52行　キャンバスの描画

　50行　コンテキストの取得

　51行　タイマーの設定

52行　キャンバスサイズの設定

54～60行　編集の初期値設定

54行　変数の設定

55, 56行　入力した初速度と角度を変数に代入する

57, 58行　入力した値からx座標、y座標に分けた速度を代入する

59行　発射地点を設定する

表13－1　コンテキストオブジェクトのメソッド例

メソッドなど	機能
clearRect(x,t,w,h)	canvas 上の指定された四角形の範囲のすべての図形をクリア(消去)する。引数は、四角形の左上のx座標、y座標、四角形の幅w、四角形の高さhとなっている。
beginPath()	新しいパスを作成する。 ※図形の各点をつないだ経路をパスという
arc(x,t,z,s,e,a)	円弧を描く。引数は中心位置の x 座標、y座標、半径r、ラジアン単位で指定した開始角度s、終了角度e(false は時計回り、true は反時計回り)
closePath()	パスを閉じる(最後の座標から、開始座標に向けて線を引く)
fillStyle	色を指定する。' rgb(R:赤,G:緑,B:青)の各値は0～255で指定する。
fill()	閉じたパスの内部を塗りつぶす

表13－2　関数

関数など	機能
Number	数値に変換
prompt	入力
Math.sin	サイン
Math.cos	コサイン
Math.PI	円周率

14　時間を制御する

　ゲーム等で制限時間を設けて点数を競ったり、目標を達成した時の時間を計るときに使う方法を考える。

　現在の時刻を格納するのが Date オブジェクトで、現時刻をプラットフォームに依存しない形であらわしており、1970年1月1日午前0時(UTC)からのミリ秒を表す整数値を記録して、時刻を表している。この値から getHours()　メソッドで「時」、get.

js14-0.htm　Date オブジェクトと各メソッド

```
009  let d = new Date();
010  alert(d);
011  alert(d.getHours()+"時"+d.getMinutes()+"分"+d.getSeconds()+"秒");
```

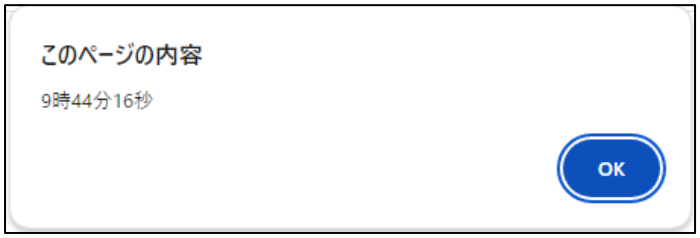
9行　Date()オブジェクト(現時刻)の値をdに代入

10行　dの値(現時刻)を表示

11 行　現時刻の「時」「分」「秒」を取り出し、〇時〇分〇秒の形で表示する。

10行の結果

11行の結果



■ストップウォッチの作成

経過時間を得るには、開始時刻を記録しておき、現時刻との差を取って表示する

js14-1.htm　ストップウォッチ

```
005        <style>
006         .hyouji{color:#772;font-size:28px;
007         background-color:#FFC;
008         border:3px solid #660;
009         font-family: arial, sans-serif;
010         max-width:250px;margin: 0 auto;
011         text-align: center;}
012         #time{font-size: 32px; margin: 0px 0; margin-top :5px;}
013         button {padding: 5px 10px 5px 10px; font-size: 16px;
014                 border: none; color: #FFF;background-color: #772;
015                 border-radius: 3px; margin-bottom :5px;}
016         button:hover {cursor: pointer; background-color: #DB9;}
017         button:disabled {cursor: default;background-color: #DB9;}
018       </style>
```

5～18行　CSS でボタンや時間を表示する文字の設定をする

```
019       <title>14-1</title>
020     </head>
021   <body>
022     <div class="hyouji">
023      <div id = "time">00:00.000</div>
024      <button id="start" onclick="start()">Start</button>
025      <button id="stop" onclick="stop() " disabled>Stop</button>
026      <button id="reset" onclick="reset()" disabled>Reset</button>
027      </div>
028      </div>
```

22行　ストップウォッチ全体のレイアウトを作るために ID 名「hyouji」の div 要素を作る

23行　22行で作った hyouji の中に、文字を表示させるための ID 名「time」のdiv 要素を作り、初期値を表示させる

24～26行　各ボタンに名前を付けて、クリックしたときにそれぞれの処理をする関数が実行されるようにする

```
029 <script>
030 let startButton;      // start ボタン
031 let stopButton;       // stop ボタン
032 let resetButton;      // reset ボタン
033 let showTime;          // 表示時間
034 let timer;             // setinterval, clearTimeout で使用
035 let startTime;         // 開始時間
036 let elapsedTime = 0; // 経過時間
037 let holdTime = 0;      // 一時停止用に時間を保持<body>
```

30～37行　各変数の初期値を設定する

```
038 window.onload = function () {
039     startButton = document.getElementById("start");
040     stopButton = document.getElementById("stop");
041     resetButton = document.getElementById("reset");
042     showTime = document.getElementById("time");
043 }
```

38行　onload ですべての DOM ツリー構造及び関連リソースが読み込まれた後に39～42行が実行される。

39行　ID 名「start」、「stop」、「reset」、「time」の情報をそれぞれの変数に代入する

```
044 // スタートボタン押下時
045 function start(){
046     startTime = Date.now(); // 開始時間を現在の時刻に設定
047     measureTime();       // 時間計測
048     startButton.disabled = true;
049     stopButton.disabled = false;
050     resetButton.disabled = false;
051 }
```

45～51行　スタートボタンを押したときに処理をするstart()関数

46行　現在の時刻を startTime 関数に代入する

47行　時間を計測する measureTime 関数を呼び出す

48行　スタートボタンの disabled プロパティを true にして、ボタンを無効にする。

49行　ストップボタンの disabled プロパティを false にして、ボタンを有効にする。

50行　リセットボタンの disabled プロパティを false にして、ボタンを無効にする。

```
052 // ストップボタン押下時
053 function stop(){
```

```
054     clearInterval(timer);  // タイマー停止
055     holdTime += Date.now() - startTime;  // 停止時間を保持
056     startButton.disabled = false;
057     stopButton.disabled = true;
058     resetButton.disabled = false;
059 }
```

53～58行 ストップボタンを押したときの処理の stop()関数

54行 claarInteval で timer のタイマーを停止させる

55行 現在表示させている時間 holdTime に、スタートボタンを押してからストップボタンを押すまでの時間を加える

56行 スタートボタンの disabled プロパティを false にして、ボタンを有効にする。

57行 ストップボタンの disabled プロパティを true にして、ボタンを無効にする。

58行 リセットボタンの disabled プロパティを false にして、ボタンを無効にする。

```
060 // リセットボタン押下時
061 function reset(){
062     clearInterval(timer);       // タイマー停止
063     elapsedTime = 0;      // 変数、表示を初期化
064     holdTime = 0;
065     showTime.textContent = ”00:00.000”;
066     startButton.disabled = false;
067     stopButton.disabled = true;
068     resetButton.disabled = true;
069 }
```

61～69行 リセットボタンを押したときの処理の stop()関数

62行 claarInteval で timer のタイマーを停止させる

63、64行 経過時間、保持時間ともに0にする

65行 時間の表示を最初に戻す

66行 スタートボタンの disabled プロパティを false にして、ボタンを有効にする。

67行 ストップボタンの disabled プロパティを true にして、ボタンを無効にする。

68行 リセットボタンの disabled プロパティを false にして、ボタンを無効にする。

```
070 // 時間を計測
071 function measureTime() {
072     timer = setTimeout(function () {          // タイマーを設定
073         elapsedTime = Date.now() - startTime + holdTime;    // 経過時間を設定し、画面へ表示
074         showTime.textContent = new Date(elapsedTime).toISOString().slice(14, 23);
075         measureTime();      // 関数を呼び出し、時間計測を継続する
```

```
076         }, 10);
077 }
```

71～76行　時間を計測する measureTimer 関数

　72行　75行で設定した10ミリ秒（0. 01秒）になると function（）{73～75行}を実行し、タイマーをコントロールできるようにそれを timer に代入している

　73行　経過時間＝現在の時間―開始した時間＋保持している時間

　74行　new Date（）で本日の時刻を取得する。toISOString（）メソッドは、簡潔な拡張表記の ISO 形式（ISO 8601）の文字列を返し、例えば「2023-04-17T01:52:26.878Z」を戻す。常に 24 文字または 27 文字の長さになり、それぞれ、YYYY-MM-DDTHH:mm:ss.sssZ または ±YYYYYY-MM-DDTHH:mm:ss.sssZ となる。なお、UTC エリア以外では正しくない日付を戻す、slice（）は文字列の一部分を取り出し、それを新しい文字列として返す。例えば「52:26.878」つまり、現在経過時間の範囲指定した時、分、秒の文字列を showTime.textContent に代入する

参考　ストップウォッチの作り方
https://nyanblog2222.com/programming/javascript/4829/

■タイマーの作成

残り時間を得るには、設定した時間から、開始時刻を記録しておき、現時刻との差を取って表示する

js14-2.htm　タイマー

js14－1と同様に5～18行　CSS でボタンや時間を表示する文字の設定をする

```
021 <div class="hyouji">
022 <div id="timer">00:00</div> <!-- タイマーの表示部分 -->
023 <div id="controls"> <!-- ボタンコントロール部分 -->
024   <div class="box">
025     <button id="min">分</button>
026     <button id="sec">秒</button>
027   </div>
028   <div class="box">
029     <button id="start">スタート</button>
030     <button id="reset">リセット</button>
031   </div>
032 </div>
033 </div>
```

21行　タイマー全体を表示する ID 名「hyouji」の div 要素を追加する
22行　タイマーの時間を表示する ID 名「timer」のdiv要素を追加し、初期値の00：00を表示する
23行　タイマーの分と秒の設定をするボタン全体を表示する ID 名「controls」のdiv要素を追加する
24、28行　controls の div 要素の中に「分」と「秒」、「スタート」と「リセット」のボタンを2つ表示させる class 名ボックスを追加する
25、26行　分と秒のボタンを作り、分には ID 名「min」、秒にはID名「sec」をつける
29、30行　スタートとリセットのボタンを作り、スタートにはID名「start」、リセットには ID 名「reset」をつける。

```
035 (function () {  //即時関数
036     let timer = document.getElementById('timer');
037     let min = document.getElementById('min');
038     let sec = document.getElementById('sec');
039     let start = document.getElementById('start');
040     let reset = document.getElementById('reset');
041     let startTime; // スタートタイムを押した時の時間を入れる変数
042     let timeLeft;  // 残り時間を計算するための変数
043     let timeToCountDown = 0; // タイマーに設定された時間
044     let timerId; // clearTimeout メソッドを使いたいので、その時用に変数定義
045     let isRunning = false;  //カウントダウンの状態を管理できるようにする変数
```

35～112行　即時関数。関数を定義すると同時に実行する。
35～39行　タイマーの数値の表示場所を ID 名「timer」、4つのボタンのID名は、分を「min」、秒を「sec」、スタートを「start」、リセットを「reset」っと設定する。
40～44行　各変数の宣言

```
046 function updateTimer(t) {   // 残り時間を表示するために、ミリ秒を渡すと、分や秒に直してくれる関数
047         let d = new Date(t); // 引数として渡された t で、変数 d と名前を付けてデータオブジェクトを作る
048         let m = d.getMinutes();
049         let s = d.getSeconds();
050         m = ('0' + m).slice(-2);
051         s = ('0' + s).slice(-2);
052         timer.textContent = m + ':' + s;
053         let title = timer.textContent = m + ':' + s;;    // タイマーをタブにも表示する
054         document.title = title;
055     }
```

46行　残り時間を幼児する updateTimer()関数。返り値はt
47～49行　dに現在の時刻、そこからmには分、sには秒を代入する
50、51行　2けたになるように整える
52行　「mm:ss」の形でID名「timer」に表示する
53、54行　　52行目と同じようにタイトルバーに表示する

```
056     function countDown() {
057         timerId = setTimeout(function () {              // 10 ミリ秒後に実行する
058             timeLeft = timeToCountDown - (Date.now() - startTime);  // 残り時間 = カウントされる時間
    - 現在時刻
059             if (timeLeft < 0) {    // 残り時間が 0 になった時の処理
060                 isRunning = false;
061                 start.textContent = 'スタート';
062                 clearTimeout(timerId);
063                 timeLeft = 0;
064                 timeToCountDown = 0;
065                 updateTimer(timeLeft);
066                 return;
067             }
068             updateTimer(timeLeft)
069             countDown();    // countDown を再帰的に呼び出すために記述
070         }, 10);
071     }
```

56～71行　カウントダウンをする contDown 関数
57 行　58～69行の処理を10ミリ秒(0. 01秒)間隔で実行する。制御ができるように、timerId 変数に代入している
58行　残り時間＝タイマーに設定された時間ー現在の時間
59～67行　残り時間が0になった時の処理
　60行　実行されているか判別する isRunning を false にする
　61行　スタートボタンにスタートという文字を表示する
　62行　timerId のタイマーの設定を外す
69行　countDown を再帰的に繰り返し呼び出す

```
072     start.addEventListener('click', function () {  // スタートを押したときの処理
073         if (isRunning === false) {
074             isRunning = true;
075             start.textContent = 'ストップ';
076             startTime = Date.now();
077             countDown();  // カウントダウンの機能は再帰的に実行
078         } else {
079             isRunning = false;
080             start.textContent = 'スタート';  // 表記を Start に戻す
081             timeToCountDown = timeLeft;  // この時点の timeLeft で更新してあげる
082             clearTimeout(timerId);  // カウントを止めたいので clearTimeout する
083         }
084     });
```

72～84行　スタートボタンを押したときの処理
　73～77行　実行しているときの処理
　78～83行　実行していないときの処理

```
085     min.addEventListener('click', function () {    // 分を押した時の処理
086         if (isRunning === true) { // カウントダウン中に設定時間を変更できないようにする
087             return;
088         }
089         timeToCountDown += 60 * 1000;  // 分 = 60 秒なので
090         if (timeToCountDown >= 60 * 60 * 1000) {  // 60 分、60 秒を超えたら 0 にする
091             timeToCountDown = 0;
092         }
093         updateTimer(timeToCountDown);   // timeToCountDown を timer に反映させたいので
    upDatetimer を使う
094     });
```

85～94行　分ボタンを押したときの処理

```
095     sec.addEventListener('click', function () {    // 秒を押した時の処理
096         if (isRunning === true) {    // カウントダウン中に設定時間を変更できないようにする
097             return;
098         }
099         timeToCountDown += 1000;  // 1秒なので
100         if (timeToCountDown >= 60 * 60 * 1000) {
101             timeToCountDown = 0;
102         }
103         updateTimer(timeToCountDown);    // timeToCountDown を timer に反映させたいので
    upDatetimer を使う
104     });
```

95～104行　秒ボタンを押したときの処理

```
105     reset.addEventListener('click', function () {   // リセットを押した時の処理
106         if (isRunning === true) {   // カウントダウン中に設定時間を変更できないようにする
107             return;
108         }
109         timeToCountDown = 0;
110         updateTimer(timeToCountDown);     // timeToCountDown を timer に反映させたいので
    upDatetimer を使う
111     });
```

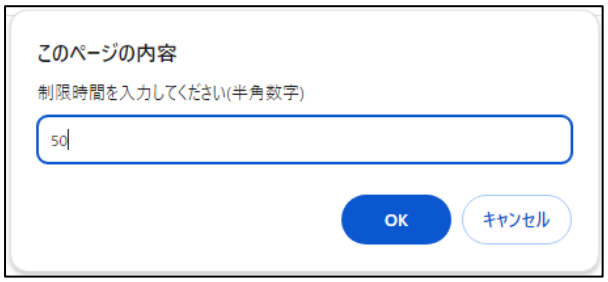
105～111行　リセットボタンを押したときの処理

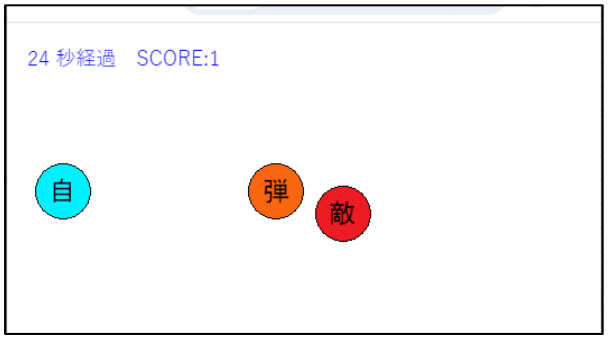
15　プログラミング実践　ゲーム制作

今まで学んだことを使ってゲームを作りなさい。必要であれば、図書館などで書籍を借りて作りなさい。

（例1）js15-1. 自力で制作したシューティングゲーム

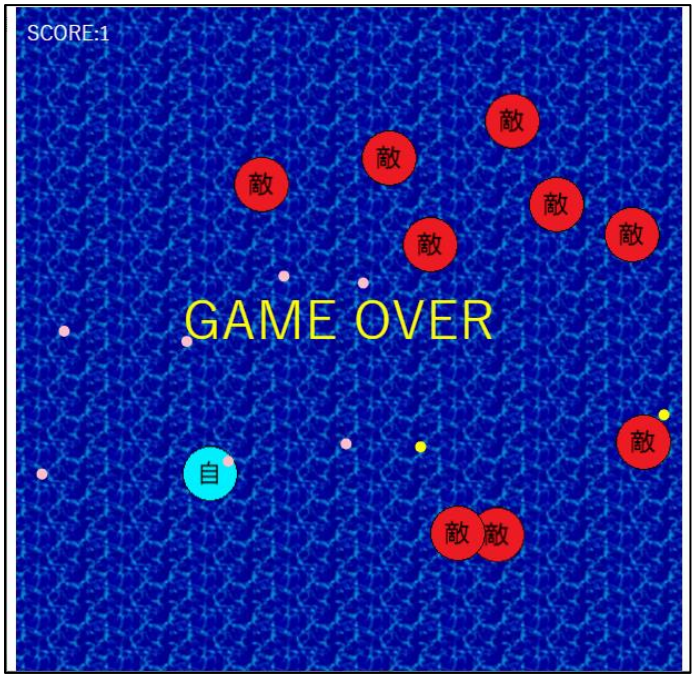
制限時間を決めて、制限時間までに何点取れるか競うゲーム
弾が敵に当たった時に、敵の画像が変わる。インターネットで情報収集をしながら、ほぼ自力で制作した。





（例2）js15-2. 書籍を参考にして制作したシューティングゲーム

クラス、配列を使って、複数の敵が登場したり、複数の弾を発射したりできるようにしている。
図書館を回って、横浜市立図書館で『田中賢一郎　著　「ゲームで学ぶ JavaScript 入門」(インプレス)』のプログラムを参考にして作成した。


SCORE:1
敵
敵
敵
敵
敵
敵
GAME OVER
敵
自
敵
敵