【このサンプルで学べること】

・canvas要素への描画

・アニメーション

・衝突判定

・localstrage

・ondeviceorientation

・メソッド

【まえがき】→この章の概要

【アプリ紹介】

■■ゲーム画面(top.png)

【プログラムの構造・流れ】

【第1節】 定数・変数・オブジェクト・関数について

1-1.定数について

■■ステータス情報

■■画像パス

■■カラー情報

1-2.変数について

■■Elementオブジェクト

■■itemsAry配列

1-3.オブジェクトについて

■■blockオブジェクト

■■ゲームステータスオブジェクト

■■ballオブジェクト

■■自機オブジェクト

■■サウンドオブジェクト

■■ユーティリティオブジェクト

1-4.classについて

■■Itemクラスの解説

1-5.関数について

■■各関数の解説

【第2節】 アイテムについて

2-1.Itemクラスについて

■■各プロパティ・メソッドの解説

2-2.関数の解説

■■constructor関数

■■itemTimeCheck関数

【第3節】 canvas要素での画像、テキスト描画について

3-1.htmlファイルへの記述

■■index.htmlへcanvas要素を記述

3-2.javascriptからの制御

■■javascriptからcanvas要素を取得

■■図形を描画

■■テキストを描画

■■画像を描画

■■アニメーション(ループ)処理

【第4節】 イベント

4-1.イベント記述場所

4-2.各イベント解説

■■keydown部分

■■keyup部分

■■click部分

■■DOMContentLoaded部分

【第5節】 ボールを動かす

5-1.ボールオブジェクトのプロパティ・メソッドについて

5-2.loop関数内での記述について

■■ボール描画部分

■■ba.move();メソッドについて

【第6節】 衝突判定

6-1.ボールと壁との衝突

■■ballHitsWalls関数

6-2.ボールと自機との衝突

■■ballHitsPlayer関数

6-3.ブロック

■■blオブジェクト

■■mapプロパティの解説

■■initMapメソッドについて

■■initblocksColorメソッドについて

■■util.blockDrowメソッドでブロック描画

6-4.ブロックとの衝突

■■ballHitsBlocks関数

■■アイテム表示とブロック削除

【第7節】 スマホジャイロセンサー

7-1.window.ondeviceorientationについて

■■eventオブジェクト内のgammaプロパティを利用する

■■p.moveLeft・p.moveRightプロパティへの反映

【まえがき】

この章では、ブロック崩しゲームを作成します。昔からある懐かしいゲームです。

pcからのキー操作・スマホの傾き操作で自機を左右に移動させプレイします。



ゲーム画面（index.html）

完成したコードはこちらになります。（リンク）

canvas要素をクリック（スマホでは画面をタップ）、又はenterキー、スペースキー押下でゲームがスタートします。

画面下方にある自機となる四角い物体を方向キー（左右）またはA（左へ移動）,D（右へ移動）キーで左右に動かすことができます。スマートフォンでプレイする場合は、スマートフォンを傾けて自機を操作することができます。

ゲームがスタートすると、ロゴマークが入ったボールが動き始め、壁や、画面上部のブロックに当たると反射します。

Macintosh HD:Users:mosohiko:programming:javascript:apps:ballGame:update_es2015:images:gsacLogoBlue50x50.png

ボール

ボールが画面最下部まで到達してしまうとゲームオーバーとなります。

ボールが画面下方に移動して来たら自機を移動させ、ボールが画面最下部に落ちないようにはじき返します。

ボールが画面上部のブロックに当たると、ブロックが消えます。

すべてのブロックを消し終わると、ゲームクリアとなります。

ブロックの中の幾つかには、アイテムが隠されており、そのブロックにボールが当たると、ブロックが消えた後、アイテムが出現し、上から降ってきます。

自機を動かしてアイテムを取る事ができます。

アイテムは色によって効果が異なり、グリーンのアイテムを取得すると自機の幅が広くなり、ピンクのアイテムを取ると、自機の幅が狭くなります。

Macintosh HD:Users:mosohiko:programming:javascript:apps:ballGame:update_es2015:images:gsacLogoGreen50x50.png

グリーンのアイテム

Macintosh HD:Users:mosohiko:programming:javascript:apps:ballGame:update_es2015:images:gsacLogoPink50x50.png

ピンクのアイテム

ゲームをスタートすると、時間のカウントが始まります。

ゲームをクリアした際、クリア時の時間が過去最短のタイムであった場合、「besttime」として登録されます。

besttimeはブラウザのlocalstrageへ登録され、次回このアプリを開いた際にbesttimeが読み込まれます。

【第1節】 定数・変数・オブジェクト・関数について

それでは「js」フォルダ内のmain.jsを解説していきます。

main.jsファイルは、

・定数定義ブロック

・変数定義ブロック

・オブジェクト作成ブロック

・class定義ブロック

・関数定義ブロック

・イベント設定ブロック

上記6つのブロックに分けることができます。

「イベント」ブロックについては第4節で解説いたしますので、ここでは説明いたしません。まずは「定数」ブロックから見ていきましょう。

■定数ブロック

定数ブロックでは、後程値を変更する必要のないものを定義しています。

「"./images/gsacLogoBlue50x50.png"」など画像パス情報や、「"#294"」などの色の文字情報は、特に変更して使用する必要がないため、定数で定義します。

しかし、javascriptには定数という概念がありませんでした。

先日策定された、es2015で「let」「const」キーワードによる変数宣言が可能になりましたので、constキーワードで定数を宣言します。

es2015について気になる方はご自身で調べてみてください。

今回使用する定数は以下の通りです。

//ブロックを描画する際に使用する定数

const NO\_BLOCK = 0; //ブロックがない

const EXIST\_BLOCK = 1; //ブロックがある

const BLOCK\_WITH\_GREEN\_ITEM = 2; //ブロックがあり、緑のアイテムが入っている

const BLOCK\_WITH\_PINK\_ITEM = 3; //ブロックがあり、ピンクのアイテムが入っている

//効果中のアイテムを表す定数

const NO\_ITEM = 0; //アイテム効果中でないことを示す

const GREEN\_ITEM = 1; //greenのアイテム効果中であることを示す

const PINK\_ITEM = 2; //pinkのアイテム効果中であることを示す

//自機の幅

const NORMAL\_WIDTH = 100; //自機の標準時の幅

const SHORT\_WIDTH = 60; //緑のアイテムを取った際の自機の幅

const LONG\_WIDTH = 140; //ピンクのアイテムを取った際の自機の幅

//画像パス

const PATH\_ARY = {

LOGO\_BLUE: './images/gsacLogoBlue50x50.png',

LOGO\_GREEN: './images/gsacLogoGreen50x50.png',

LOGO\_PINK: './images/gsacLogoPink50x50.png'

};

//カラー

const COLOR\_ARY = {

WHITE: '#fff'fff,

GREEN: '#229944',

PINK: '#ff4488',

BLUE: '#4444f',

SKYBLUE: '#add8e6',

DODGERBLUE: '#0088dd',

PURPLE: '#C48793'

};

// canvas要素を取得

const canvas = document.getElementById('canvas'); //canvasタグを定数へ格納

// コンテキストを定数へ格納

const ctx = canvas.getContext('2d');

「自機の幅」は、実際の幅を数値で格納しています。自機の幅を変更する際に、これらの定数名で幅を指定します。

・「ブロックを描画する際に使用する定数」

・「効果中のアイテムを表す定数」

以上については、整数を格納しています。

整数なので、わざわざ定数として命名する必要は無いかもしれませんが、数にはそれぞれ意味があります。例えば「ブロックを描画する際に使用する定数」では「0」は「ブロックが無い」という状態をしめしています。 数値のままでは、一見するとどういう状態を意味するものなのか理解しづらい状態です。

そのため、「const NO\_BLOCK = 0;」というように、すぐに意味が把握できるような名前をつけることで、コードの意味を把握し易くしました。

「画像パス」「カラー」については、const PATH\_ARYやconst COLOR\_ARYのようにオブジェクトを作成し、そのプロパティとして情報を格納してあります。

■変数ブロック

変数ブロックでは以下の変数をまとめて宣言してあります。

let itemsAry = []; //アイテムを格納する配列

let bl; // ブロックの情報をまとめたオブジェクト。blocksの略でbl

let gs; //ゲームのステータスをまとめたオブジェクト。gameStatusの略でgs

let ba; //ballの情報をまとめたオブジェクト。ballの略

let p; //自機の情報をまとめたオブジェクト。playerの略

let sf = {}; // soundをまとめたオブジェクト。soundFilesの略

let util; // ユーティリティ関数をまとめたオブジェクト。

■「オブジェクト」ブロック

「オブジェクト」ブロックでは、以下のものを定義しました。

let gs = {};・・・ゲームステータスオブジェクト

ゲームの状態に関する情報をまとめたオブジェクト。

ゲーム中かどうか、ゲームスタートから経過した時間、最短クリア時間などの情報をプロパティに格納しています。

ba = {};・・・ballオブジェクト

ボールの情報をまとめたオブジェクト。

ボールの位置や、x方向・y方向のスピード成分、ボールの直径などの情報を格納しています。

p = {};・・・自機オブジェクト

自機に関するオブジェクト。自機を描画する際に必要な座標情報、スピード、アイテム効果中か否か等の情報を格納しています。

sf = {};・・・soundFilesオブジェクト

音に関する情報を格納したオブジェクトです。

bl = {};・・・ブロックオブジェクト

ブロックに関する情報をまとめたオブジェクト。

util = {};・・・ユーティリティオブジェクト

ユーティリティ関数をまとめたオブジェクト。

※中身については省略

baオブジェクトはballに関するプロパティ・メソッドを格納しています。

中身を全て変数で管理しても良いのですが、必要な情報はオブジェクトのプロパティとして記述し、オブジェクト単位でまとめ整理しました。

■classブロック

classブロックではItemクラスを定義しています。

Itemはゲーム中に複数同時に画面上に表示されるケースがあるため、classとして定義しました。

es2015が策定され、classキーワードを使用してclassを定義できるようになった為、今回はclassキーワードでItemクラスを作成しました。

■関数ブロック

init関数　// 初期関数

gameStart関数　// ゲームをスタートする際に必要な処理をまとめた関数

startStop関数　// ゲーム中の時はゲームを終了、そうでない時はスタートする関数

gameOver関数　// ボールが最下部に来た際にゲームを終了させる関数

gameClearFunc関数　// ゲームクリア処理関数

ballHitsBlocks関数　// ボールとブロックの当たり判定関数

ballHitsWalls関数　// ボールと壁との衝突判定関数

itemTimeCheck関数　// アイテムの効力時間を監視する関数

itemMove関数　// itemを動かすand自機との当たり判定

ballHitsPlayer関数　// 自機とボールとの当たり判定関数

keyDown関数　// キーダウンイベントに渡す関数

keyUp関数　// キーアップイベントに渡す関数

loop関数　// ループ関数

定数・変数・オブジェクト・関数について、簡単に触れていきました。

詳細については以降の章で解説していきます。

【第2節】 Itemクラスについて

2-1. Itemクラスについて

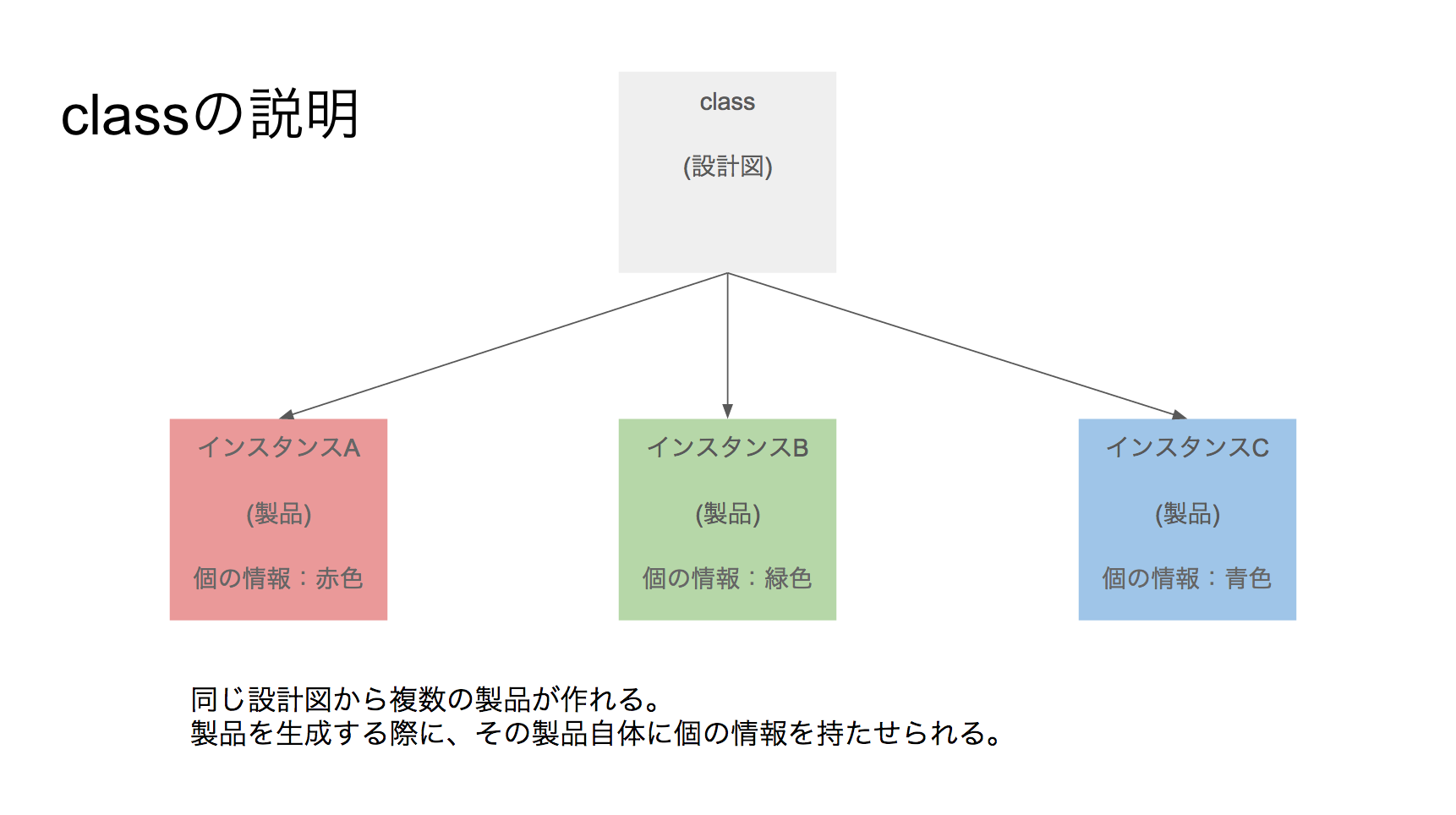


Itemは、ブロックを消した際に消したブロックの位置で出現し、画面の下方へ落ちてきます。

Itemは画面上に同時に複数個描画されるものなので、複製しやすいようにクラスで記述してあります。

クラスは設計図のようなものです。この設計図を基に、「製品」のような、同じような値や機能を持ったオブジェクトを複製することができます。複製されたものを「インスタンス」と呼びます。

クラスで設計されたインスタンスを複製し、Itemを画面上に複数描画しています。



■■各プロパティ・メソッドの解説

Itemクラスについて解説いたします。

//------------------------------------------------------------Item class

class Item {

constructor(x, y, speed, logoImgPath, type){

this.x = x;

this.y = y;

this.speed = speed;

this.acceleration = 1.02;

this.alive = true; // 生死フラグ

this.diameter = 25; // 直径

this.radius = this.diameter / 2; // 半径

this.width = this.diameter; // 幅

this.height = this.diameter; // 高さ

this.logoImg = new Image(); // imgタグ

this.logoImg.src = logoImgPath; // 画像パス

this.type = type; // 種別

this.GREEN\_ITEM = 1; // greenのアイテム効果中であることを示す

this.PINK\_ITEM = 2; // pinkのアイテム効果中であることを示す

}

move(){

if(this.type === this.PINK\_ITEM) this.speed \*= this.acceleration;

this.y += this.speed;

}

}

constructor関数について

constructor関数は、クラスからインスタンスを生成する際に、インスタンスが持つプロパティを生成する為の特別な関数です。内部は以下のようになっております。

class Item {

constructor(x, y, speed, logoImgPath, type){

this.x = x;

this.y = y;

this.speed = speed;

…

引数として、x座標、y座標、スピードの値、画像パス、タイプの５つの情報を受け取り、「this.プロパティ名 = 引数で受け取った値」という形で格納されます。

この「this」は、生成されたインスタンス自身を指すものになりますので、インスタンスの個の情報として格納されます。

インスタンスの生成方法

インスタンスを生成するには、「new」というキーワードを使って、一般的には以下のように書きます。

new Item( 引数1, 引数2, 引数3, 引数4, 引数5 );

このように書くことで、設計図を基に、引数として渡した個の情報を持ったオブジェクトを生成し使用することができます。

move関数について

クラス内で定義された関数については、そのクラスのインスタンスから使用することができます。

move関数の内部は以下のようになっております。

move(){

if(this.type === this.PINK\_ITEM) this.speed \*= this.acceleration; // typeがPINK\_ITEMであった場合はスピードを加速させる。

this.y += this.speed; // speedの値分下方向へ移動

}

move関数を一度実行すると、インスタンス生成時に指定した速度の値分、下方へ移動します。

画面が１秒間に何度も更新されるたびにこのメソッドが実行されるように記述しているので、Itemインスタンスが表示された後、落下しているようにアニメーションさえることができます。

itemのタイプによって、落ち方を変化させており、GREENのアイテムの場合は等速度で、PINKのアイテムだった場合は加速しながら落ちていきます。

【第3節】 canvas要素での画像、テキスト描画について

この節では、canvas要素部分について解説いたします。

3-1.htmlファイルへの記述

■■index.htmlへcanvas要素を記述

今回のアプリは、ゲーム部分の描画を全てcanvas要素上で行っているため、htmlファイルの記述は非常にシンプルになっています。

Index.htmlファイルのbodyタグ直下に

<canvas id="canvas" width="375" height="540"></canvas>

と記述いたしました。width属性とheight属性を設定

3-2.javascriptからの制御

htmlファイルのcanvas要素をjavascriptから取得し、描画などの処理を行います。

■■javascriptからcanvas要素を取得

javascriptからcanvas要素を取得する方法は以下のとおりです。

main.js内

const canvas = document.getElementById('canvas');//canvasタグを定数へ格納

const ctx = canvas.getContext('2d');//コンテキストを定数へ格納

document.getElementByIdメソッドの引数として、canvas要素のid属性値として記述した、” canvas”という文字列を渡し実行する事で、canvasのdom要素を取得し、let canvas変数へ格納しました。

canvas を扱いやすくするため、canvas.getContext('2d');メソッドを実行し、ctxへ格納します。

canvas.getContext メソッドはcanvas に描画するための API にアクセスできるオブジェクトを返します。

■■図形を描画

それでは、自機を描画する部分について、詳しく見てみましょう。

自機の描画はloop関数内の以下の記述部分で行われています。

//自機を描画する

util.drawRect(p.posX, p.posY, p.width, p.height, p.color);

上記メソッド内の、

ctx.fillRect(x, y, width, height);

この部分で部分で四角形を描画しています。

ctxはcanvasのコンテキストです。このコンテキストオブジェクト内に、四角を描画する「fillRect」というメソッドがあります。このメソッドに、描画開始x座標、描画開始y座標、幅、高さ、以上4つの引数を渡すことで四角をcanvasへ描画します。

このメソッドを実行する前に、「ctx.beginPath();」で現在のパスをリセットし、「ctx.fillStyle」プロパティに塗りつぶしの色を設定する必要があります。

ctx.fillRect メソッドを格納しているutilは、ユーティリティ関数をまとめたオブジェクトです。utilオブジェクト内にdrawRectという、canvasに四角形を描画するメソッドを定義しており、それを使用して自機を描画しています。

utilオブジェクト内のdrawRectメソッドを詳しく見てみましょう。

util = {

　//〜〜〜〜〜〜〜〜

//四角を描画する関数

drawRect: (x = 0, y = 0, width, height, color = COLOR\_ARY.WHITE) => { //canvasに描画する処理をまとめたもの

ctx.beginPath(); //現在のパスをリセット

ctx.fillStyle = color; //塗りつぶしの色を設定

ctx.fillRect(x, y, width, height); //塗りつぶし四角形を描画

} //〜〜〜〜〜〜〜〜

}

util.drawRect(aaa,bbb,ccc,ddd);

と記述することで、メソッドを実行することができます。

util.drawRectメソッドは引数を5つ取ります。

第1引数(x)： 四角を描画する左側のposition

第2引数(y)：四角を描画する上側のposition

第3引数(width)： 四角形の幅

第4引数(height)：四角形の高さ

第5引数(color)：四角形の色

以上5つのデータを渡し、第1引数から第4引数のデータはctx.fillRectメソッドの引数として使われ、第5引数はctx.fillStyleプロパティの値として引き渡し、データを引き渡し、四角を描画しています。

このメソッドは自機やブロックを描画する際に使用します。

■■画像を描画

次に、ボールを描画する部分について、詳しく見てみましょう。

//ボール描画部分

ctx.drawImage(ba.imgPath, ballX, ballY, ballW, ballH); //ボールを描画

ctxはcanvasのコンテキストです。このコンテキストオブジェクト内のdrawImageメソッドを使用して、画像をボールの画像を描画しています。

ctx.drawImageメソッドは3つの異なる引数セットを使って呼び出すことができます。このメソッドは引数の渡し方が幾つかあり、少し複雑です。興味のある方は調べてみてください。

今回は引数を5つ渡す方法で使用いたしました。

drawImage(image, dx, dy, dw, dh)

引数の詳細については下記の通りです。

第1引数・・・image引数は、HTMLImageElement, HTMLCanvasElement, HTMLVideoElement のいずれかのインスタンスです。

第2引数・・・dx　画像を貼り付けるx座標

第3引数・・・dy　画像を貼り付けるy座標

第4引数・・・dw　画像の幅

第5引数・・・dh　画像の高さ

実際にメソッドを使用している部分はこちらです。

let ballX = ba.posX - ba.radius;//ボールを描画する為の基準x座標

let ballY = ba.posY - ba.radius;//ボールを描画する為の基準y座標

let ballW = ba.diameter;//ボール画像幅

let ballH = ba.diameter;//ボール画像高さ

ctx.drawImage(ba.imgPath, ballX, ballY, ballW, ballH); //ボールを描画

ballX変数とballY変数の値については、ボールの中心位置から、x座標・y座標ともにボールの半径分左上にずらすように計算しています。こうすることで、ボールの座標と、画像の中央を合わせています。

■■テキストを描画

canvasにテキストを描画する部分は、util.displayTextメソッド内に処理をまとめました。具体的にメソッド内を確認してみましょう。

let util = {

//~~~~~~~~~~~~~~~

displayText: function(textObj, x, y) {

// || の後ろにデフォルト値を設定

let text = textObj.text;

ctx.font = textObj.font || "14px 'ＭＳ Ｐゴシック'";

ctx.textAlign = textObj.position || 'center';

ctx.fillStyle = textObj.color || '#000';//

ctx.fillText(textObj.text, x, y);//テキストを描画

},

//~~~~~~~~~~

util.displayTextメソッドは引数を3つとります。一つ目はオブジェクトを引数として受け取ります。textObj内に「text」、「font」、「position」,「color」

という4つのkeyを持つようにオブジェクトを記述すると、メソッド内の処理部分でctxのプロパティに値が代入されます。textObjオブジェクト内に該当のプロパティがなかった場合は「||」演算子の後ろの初期値の値が渡されます。引数にオブジェクトの形でデータを渡しているのは、必要なデータがなかった場合に、このように初期値を適用するためです。

二つ目、三つ目の引数は、テキストの位置をx座標、y座標で指定しています。ctx.fillText(textObj.text, x, y)の第二、第三引数にデータが渡されます。ctx.fillTextメソッドに渡される第一引数はテキストの文字列です。

util.displayTexメソッドを実行する際は、

util.displayText({

position: 'right',

text: 'BESTTIME : ' + gs.bestTime

}, canvas.width - 20, 40);

の様にオブジェクトリテラルをそのまま引数に記述しても良いですし、

let obj = {

position: 'right',

text: 'BESTTIME : ' + gs.bestTime

};

util.displayText(obj, canvas.width - 20, 40);

の様に、オブジェクトを作成して、引数に渡す事も可能です。

■■アニメーション(ループ)処理

canvasに図形や画像、テキストを描画する方法を説明してきましたが、それだけではアニメーションさせることができません。ゲームとしてアニメーションさせるには、一度画面を描画させて→画面をクリア→画面に描画、と、処理を繰返させなければなりません。

描画を繰返し実行する部分について解説いたします。

繰返し実行させている部分は以下の記述になります。

window.requestAnimFrame = window.requestAnimationFrame ||

window.webkitRequestAnimationFrame ||

window.mozRequestAnimationFrame ||

window.oRequestAnimationFrame ||

window.msRequestAnimationFrame ||

function(callback) {

window.setTimeout(callback, 1000 / 60);

};

windowオブジェクトのrequestAnimFrameというプロパティに値を代入しています。

WindowオブジェクトのwebkitRequestAnimationFrame

プロパティに関数があれば、それをrequestAnimFrameプロパティへ代入、無ければ、mozRequestAnimationFrame内の関数をチェックし、そこにも関数が無ければ・・・というように、ベンダープリフィックスが付いたrequestAnimFrameプロパティの関数をチェックしていきます。requestAnimFrameなかった場合は、最終的にsetTimeoutメソッドで代用させています。こうすることで、window.requestAnimFrameプロパティに関数を格納しています。

window.requestAnimationFrame() メソッドは、ブラウザに描画させたいアニメーションを指定し、次の再描画の前に、アニメーションを更新する指定した関数を呼び出します。

このメソッドは再描画する前に呼び出されるコールバックメソッドを引数にひとつとります。

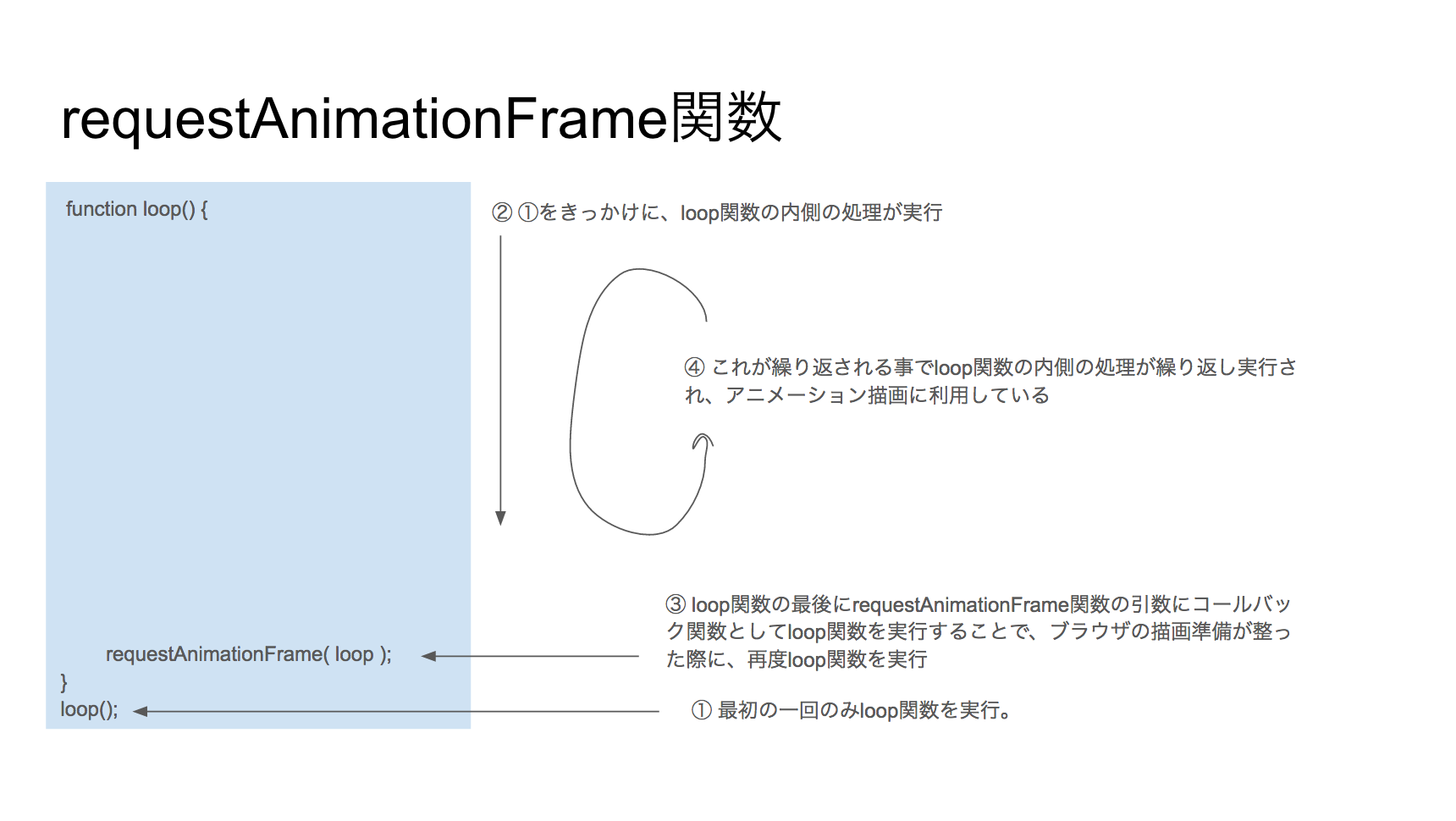
今回は、アニメーションを更新する関数としてloop関数を定義し、コールバックメソッドとして渡します。requestAnimFrameメソッドがこのコールバックメソッドをを繰返し実行することで、アニメーションを行います。

スクリーン上でアニメーションが更新できる準備が整ったとき、ブラウザにアニメーション関数（loop関数）を次の再描画が実行される前に呼び出されます。この再描画はフォアグラウンドのタブでおよそ毎秒60回行われます。バックグラウンドのタブではより少ない割合に減らされる為、メモリの消費を抑えることができます。

この関数を実行している具体的な部分も確認しましょう。

function loop() {

requestAnimFrame(loop);

この部分でrequestAnimFrame関数を実行し、コールバックメソッドとしてloop関数を繰返し実行しています。

【第4節】 イベント

この章では、イベント部分について解説します。

4-1.イベント記述場所

イベントの記述場所は、スクリプトの最下部にまとめて記述しています。各イベントの詳細について解説します。



4-2.各イベント解説

■■keydown部分

pcのキーを押した際に実行されるイベントについて、以下のように定義いたしました。

document.addEventListener('keydown', keyDown);

documentオブジェクトにaddEventListenerというメソッドがあります。こちらの第一引数に対象とするイベントの種類を表す文字列を渡します。第二引数で渡しているのは、指定されたタイプのイベントが発生するときに実行される関数です。

「Keydown」は、KeyboardEventイベントの種類の一つで、PCのキーボードが押されたタイミングで、第二引数で渡している関数を実行するという仕組みになっています。第三引数は、そのイベントリスナ（関数）をイベントキャプチャリングで処理するならtrue、イベントバブリングで処理するならfalseということになっています。この部分については少し難しいので、おまじないだと思ってください。気になる方は調べてみてください。

Keydownイベントが発生した際に実行される関数として、同じ名前のkeydown関数を定義しました。

function keyDown(event) { ①

switch (event.keyCode) { ②

case 37: // ←キー ③ここから

case 65: // Aキー

p.moveLeft = true;

break; ③ここまで

case 39: // →キー④ここから

case 68: // Dキー

p.moveRight = true;

break; ④ここまで

case 13: // enterキー⑤ここから

case 32: // spaceキー

startStop();

break; ⑤ここまで

default:

break;

}

}

addEventListenerの第二引数としてで登録した関数の第一引数（最初の引数）に、イベントオブジェクトが自動的に渡されます（①）。関数を定義する際に、keyDown(event)と第一引数にeventという引数を受け取るように定義すると、その関数内で「event」というオブジェクトから、発火したイベントに関する様々な情報を取得することができます。今回はどのキーが押されたのかを取得したいため、関数内のswitch分の条件部分に「event.keyCode」を記述し（②）、keyCodeによって実行する処理を分けています。keyCode の「37」は「←」キー、「65」は「A」キーとなっています。これらのキーが押されると、pオブジェクトのmoveLeftプロパティにtrueが代入されます。moveLeftがtrueの間は自機が左に移動していきます（③）。同様に、keyCodeが39の「→」キーやkeyCodeが68の「D」キーを押すと、p.moveRightの値がtrueとなり、自機が右に移動していきます。（④）

「13」は「enter」キー、「32」は「space」キーです。これらのキーはゲームをスタートさせる際、またはゲームを中断させる際に使用します。これらのキーを押すと、startStop(); 関数が実行され、ゲーム開始前はゲームがスタートし、ゲーム中はゲームを中断します。（⑤）

■■keyup部分

pcのキーを離した際に実行されるイベントについて、以下のように定義いたしました。

document.addEventListener('keyup', keyUp);

第二引数で渡されている関数は以下の通りです。

function keyUp(event) {

switch (event.keyCode) {①ここから

case 37:

case 65:

p.moveLeft = false;

break; ①ここまで

case 39: ②ここから

case 68:

p.moveRight = false;

break; ②ここまで

default:

break;

}

}

keyupイベントについてはpcのキーを離した際に実行されるイベントとなっています。

先ほどのkyedownイベントで、「←」キーを押すと、p.moveLeft = true;となり、自機が左へ移動していくと説明しましたが、このままでは「←」キー離しても自機が左へ移動し続けてしまします。「←」キーを離した際に自機の移動を止める為、keyupイベントに、p.moveLeft = false;と、falseを代入しています。（①）「→」キーが話されて場合も同様に、p.moveRight = false;を代入し、右方向への移動を停止しています。（②）

■■click部分

canvas部分をクリックした際にも、ゲームを開始したり終了したりできるように、canvas要素にクリックイベントを付加しました。「startStop」と関数名を第二引数へ渡し、canvas要素をクリックするたびにstartStop関数が実行されるようになっています。

■■DOMContentLoaded部分

DOMContentLoadedイベントは、最初のHTMLドキュメントの読み込みが完了した時に発火します。 スタイルシートや画像等の読み込みが終わるのを待ちません。

このイベントリスナー部分の中身は以下のようになっております。

document.addEventListener('DOMContentLoaded', function() {

gs.bestTime = localStorage.getItem('bestTime'); ①ここから

if (!gs.bestTime) {

gs.bestTime = '';

}①ここまで

});

localStorageからbestTimeの取得に成功した場合は、gsオブジェクトのbestTimeプロパティに取得した値を代入します。bestTimeの取得ができなかった場合は「””」と、空の文字列を渡します。（①）

4節 ボールを動かす

5-1.ボールオブジェクトのプロパティ・メソッドについて

5-2.loop関数内での記述について

■■ba.move();メソッドについて

■■ボール描画部分

4節 ボールを動かす

この節では、ボールを動かす部分について解説いたします。

5-1.ボールオブジェクトのプロパティ・メソッドについて

まず、ボールオブジェクトの中身について解説します。

ballオブジェクト部分

let ba = {

speed: 6,

speedX: 0,

speedY: 0,

posX: 0,

posY: 0,

diameter: 25,

radius: 12.5,

imgPath: null,

move: function() {

this.posX += this.speedX;

this.posY += this.speedY;

}

};

let baの部分で、ボールオブジェクトを格納する変数を宣言しました。baは「ball」の略です。

speed: 6,

ボールの絶対速度です。

speedX: 0,

ボールのx方向の速度です。

speedY: 0,

ボールのy方向の速度です。

posX: 0,

ボールのx座標です。

posY: 0,

ボールのy座標です。

diameter: 25,

ボールの直径です。

radius: 12.5,

ボールの半径です。

imgPath: null,

ボールの画像です。

move: function() { //ballを動かすメソッド

this.posX += this.speedX;//x座標をspeedX分移動する

this.posY += this.speedY;//y座標をspeedY分移動する

}

ボールの位置を移動するためのメソッドです。

これらのプロパティ・メソッドを使用して、ボールの描画や、衝突判定の算出に使用しています。

5-2.loop関数内での記述について

先ほどのbaオブジェクトのプロパティやメソッドを使用して、canvasへボールを表示し、アニメーションさせています。その仕組みについて解説いたします。

■■ボール描画部分

ボール描画部分はloop関数内の以下の部分で　行われています。

let ballX = ba.posX - ba.radius;//ボールを描画する為の基準x座標

let ballY = ba.posY - ba.radius;//ボールを描画する為の基準y座標

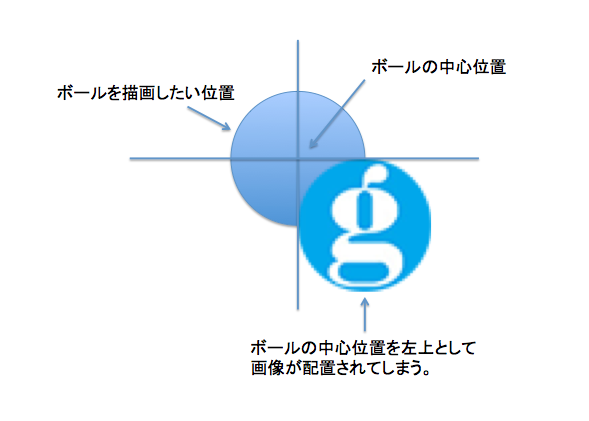
let ballW = ba.diameter;//ボール画像幅

let ballH = ba.diameter;//ボール画像高さ

ctx.drawImage(ba.imgPath, ballX, ballY, ballW, ballH); //ボールを描画

ctx.drawImageメソッドの引数部分に、ボール描画に必要な値を渡し、ボール画像を描画しています。今回は分かりやすくするために、関数の冒頭部分で、ballX（ボールのx座標位置）、ballY（ボールのy座標位置）、ballW（ボールの幅）、ballH（ボールの高さ）という変数を作成しました。ctx.drawImageメソッドの引数として渡しています。

ボール描画のための座標の渡し方については注意が必要です。ボールの座標をそのままctx.drawImageメソッドへ渡してしまうと、以下のような状態になってしまいます。



ボールを描画したい位置になるように、ボールの座標値から、渡すx・y座標をそれぞれボールの半径分（ba.radius）差し引き、引き渡す変数へ代入します。

let ballX = ba.posX - ba.radius;

let ballY = ba.posY - ba.radius;

こうすることで、ボールの中心を基準として画像が描画されます。

■■ba.move();メソッドについて

ボールを描画する方法についてはご説明しましたが、このままではボールは動きません。今度はボールを動かす部分について解説いたします。

ボールを動かす処理については、loop関数内の、

p.move();

と記述している部分で実行しています。具体的な処理は以下のようになっております。

Baオブジェクト内

move: function() {

this.posX += this.speedX;//x座標をspeedX分移動する

this.posY += this.speedY;//y座標をspeedY分移動する

}

this.posXはボールオブジェクト自身のx座標値です。同様にthis.speedX;はボールオブジェクト自身のx方向の移動速度です。

この処理はloop関数内部で実行されるので、繰り返し実行されます。

ba.moveメソッドでボールオブジェクト自身のx座標にspeedX分座標をずらし、ctx.drawImage関数で描画という処理を何度も繰り返し実行することで、スムーズなアニメーションを行っています。

【節5節】 衝突判定

ボールをアニメーション描画する部分についてはご説明しましたが、このままではボールは初期速度の方向へどんどん移動してしまい、あっという間にcanvas要素の外側へ移動して消えてしまします。ゲームとして成立させるためには、ボールが壁やブロックに衝突した際に、ボールを跳ね返らせる必要があります。

この節ではその処理部分について解説します。

6-1.ボールと壁との衝突

ボールをcanvas要素内へ留めておくため、まずは壁と衝突した際の処理について解説します。

■■ballHitsWalls関数

ボールと壁とが接触した際の処理をまとめたものがballHitsWalls関数です。

ballHitsWalls関数は以下のように記述いたしました。

--------------------------------------------- ballHitsWalls関数部分

function ballHitsWalls() {

let ballTop = ba.posY - ba.radius; ①ここから

let ballBottom = ba.posY + ba.radius;

let ballLeft = ba.posX - ba.radius;

let ballRight = ba.posX + ba.radius; ①ここまで

let hitWall = false; ②ここから

if (ballLeft <= 0 && ba.speedX < 0) {

ba.speedX = -ba.speedX;

hitWall = true;

}②ここまで

if (ctx.canvas.width <= ballRight && ba.speedX > 0) {

ba.speedX = -ba.speedX;

hitWall = true;

}

if (ballTop <= 0) {

ba.speedY = -ba.speedY;

hitWall = true;

}

if (hitWall) { ③ここから

sf.kabe.stop();

sf.kabe.play();

}③ここまで

if (ballBottom > ctx.canvas.height) { ④ここから

gameOver();

}④ここまで

}

--------------------------------------------- ballHitsWalls関数部分終了

まず、ボールの上下左右部分の座標を変数に格納し、扱いやすくしました。（①）

これらの変数を使って、ボールの左側の部分が、左の壁と接触したかどうかを判定します。（②）

左側の壁のx座標は0です。ボールの左側の座標が、0より小さい場合、左の壁に衝突したと判定しています。誤動作を起こさないように、ボールの移動方向が左方向のときのみ判定されるよう、ba.speedX < 0という条件もandで付け加えています。

衝突したと判定した場合はボールを跳ね返らせる為、x方向の速度を反転させます。

衝突したと判定した場合、hitWall変数にtrueを代入します。この値がtrueかどうかを関数の下部で判定し、衝突音を再生しています。（③）

ボールの上部と壁の上部の衝突、ボールの右部分と右の壁との衝突部分についても同様の処理を行っています。

ボールの下部が下の壁と衝突した場合は、ボールを跳ね返らせる処理ではなく、ゲームを終了させる為、gameOver関数を実行します。（④）

このballHitsWalls関数を、loop関数内で実行されるように

ballHitsWalls();

と記述すると、フレーム毎にボールと壁とが接触していないかどうかを確認し、接触していた場合に処理を行ってくれます。

6-2.ボールと自機との衝突

ボールと自機の衝突について解説します。

■■ballHitsPlayer関数

ボールと自機とが接触した際の処理をまとめたものが、以下のballHitsPlayer関数です。

-------------------------------- ballHitsPlayer関数部分

function ballHitsPlayer() {

let ballTop = ba.posY - ba.radius;

let ballBottom = ba.posY + ba.radius;

let ballLeft = ba.posX - ba.radius;

let ballRight = ba.posX + ba.radius;

let pTop = p.posY; ①ここから

let pBottom = p.posY + p.height;

let pLeft = p.posX;

let pRight = p.posX + p.width; ①ここまで

let hit = false;

if (pTop < ballBottom &&②ここから

pBottom > ballTop &&

ba.speedY > 0) {②ここまで

if (ba.posX >= pLeft && ba.posX <= pRight) {③

if (p.moveLeft) {④ここから

ba.speedX -= 1;

} else if (p.moveRight){

ba.speedX += 1;

}④ここまで

if (ba.speedX >= 6) {

ba.speedX = 5.4;

} else if (ba.speedX <= -6) {

ba.speedX = -5.4;

}

hit = true;

} else if (ba.posX < pLeft && ballRight > pLeft) {

ba.speedX = -5.4;

hit = true;

} else if (ba.posX > pRight && ballLeft < pRight) {

ba.speedX = 5.4;

hit = true;

}

if (hit) {

ba.speedY = Math.sqrt(Math.pow(ba.speed, 2) - Math.pow(ba.speedX, 2));

ba.speedY = -ba.speedY;

sf.ballHitPr.stop();

sf.ballHitPr.play();

}

}

}

-------------------------------- ballHitsPlayer関数部分終わり

ボールの上下左右部分を変数化した理由と同様に、自機の上下左右の座標についても変数化しました（①）。

続いて衝突の判定部分です。厳密に判定する場合は、より複雑な計算が必要になるかと思いますが、今回は簡易的なゲームですので、大雑把に判定しています。

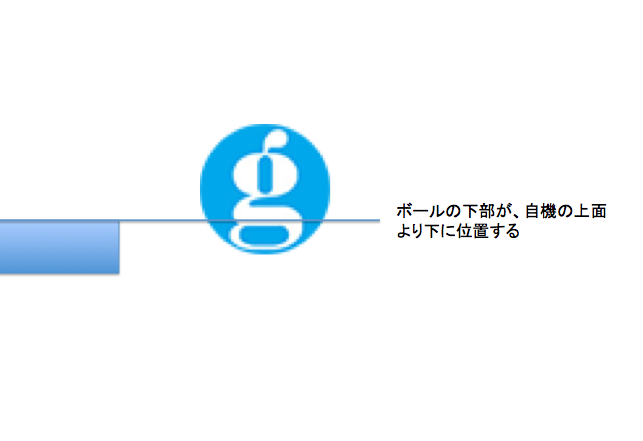
まずY座標部分の判定を行っている条件分岐部分（②）について解説します。

ここで設定している条件は以下の３つです。

・ボールの下部のy座標値が自機の上面のy座標値を超えている

・ボールの上部のy座標値が自機の下面のy座標値を超えていない

・ボールが下方向へ移動している





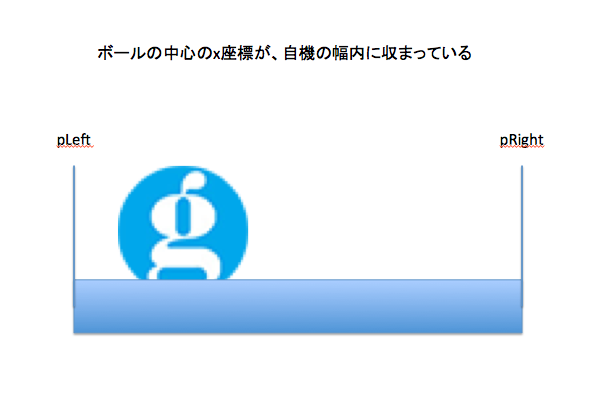


以上の３つの条件を満たしている場合、ボールと自機が衝突する可能性があると判断しています。

この条件を満たした場合、今度はx座標の判定を行います。

ここでは次のように判定しています（③）。

・ボールの中心のx座標が、自機の幅内に収まっている。



この条件を満たした際に、衝突したとみなしています。

衝突したと判定した後は、ボールの速度を変更し、跳ね返り処理を記述していますが、さらにここで、衝突時に自機が移動中であったのか否かで、ボールのx方向の速度に変化をつけています（④）。

p.moveLeftの値がtrueとなっている場合、自機は左に移動しているはずです。自機が左に移動しながらボールに当たった場合、ボールのx方向の速度を左方向へ加速させるようにするため、x方向の速度成分に1を減算しました。自機が右方向へ移動中の場合は逆にx方向の速度成分に1を加算しています。厳密な物理演算とは言い難い処理ですが、こうすることにより、ボールの方向をユーザーが若干コントロールできるようにしました。

6-3.ブロック

ここではブロック部分について解説します。

■■blオブジェクト

ブロックに関する変数をまとめたblオブジェクトを確認します。

//------------------------------------------------------ブロックオブジェクト

let bl = {

map: [],①

cols: 4,

rows: 8,

colorMap: [],

height: 20,

BLOCK\_MARGIN\_TOP: 60,

width: null,

initMap: function() {②ここから

for (let i = 0; i < this.cols; i++) {

this.map[i] = [];

for (let j = 0; j < this.rows; j++) {

let num = util.rand(4);

this.map[i][j] = num;

}

}

this.initblocksColor();

},②ここまで

initblocksColor: function() {③ここから

for (let i = 0; i < this.cols; i++) {

this.colorMap[i] = [];

for (let j = 0; j < this.rows; j++) {

let r = util.rand(255);

let g = util.rand(255);

let b = util.rand(255);

this.colorMap[i][j] = [r, g, b];

}

}

},③ここまで

//ブロックを描画する関数

blockDrow: function() {④ここから

for (let i = 0; i < bl.cols; i++) {

for (let j = 0; j < bl.rows; j++) {

if (bl.map.length && bl.map[i][j]) {//map配列にデータがあれば and ブロックがあれば

util.drawRect(//ブロックを描画

bl.width \* j,//x座標を指定

bl.height \* i + bl.BLOCK\_MARGIN\_TOP,//y座標を指定

bl.width,//ブロックの幅

bl.height,//ブロックの高さ

'rgb(' + bl.colorMap[i][j][0] + ', ' + bl.colorMap[i][j][1] + ', ' + bl.colorMap[i][j][2] + ')'//ブロックの色を指定

);

}

}

}

}④ここまで

};

bl.width = ctx.canvas.width / bl.rows;

//------------------------------------------------------ブロックオブジェクト部分終了

blオブジェクト内のmapプロパティは、ブロックのmap情報を書くのするための配列を格納しています（①）。初期状態では空の配列となっておりますが、blオブジェクト内のinitMapメソッド（②）を実行すると、blockの配置情報を二次元配列で格納します。



完成系は下記のような感じ

//------------------------- initMapメソッドを実行した後の配列イメージ

map: [

[1, 1, 1, 1, 1, 1, 1, 1],

[1, 1, 1, 1, 1, 1, 1, 1],

[1, 1, 1, 1, 1, 1, 1, 1],

[1, 1, 1, 1, 1, 1, 1, 1]

],

//-------------------------

■■initMapメソッドについて

initMapメソッドの処理部分について解説します。

ブロックは4行8列として配置します。まずは１行ずつ４回処理を記述できるように、for文を回します。this.colsプロパティに4の値が入っているのでそれを使用します。このfor文の内側で、this.map[i] = [];と記述し、map配列の内側に配列を生成し、２次元配列にします。

生成した内側の配列内に、8個のブロックを生成し、追加していきます。for文を入れ子にして、記述し、this.rowsプロパティに格納されている８の値を使用して、８回繰り返し処理していきます。util.randメソッドで０から３までの4つの数字を生成し、this.map[i][j]へ代入します。代入した値は、１節の定数部分で説明した、ブロックを描画する際に使用する定数の意味合い通りに、後ほどブロックを配置します。

let NO\_BLOCK = 0;//ブロックがない

let EXIST\_BLOCK = 1;//ブロックがある

let BLOCK\_WITH\_GREEN\_ITEM = 2;//ブロックがあり、緑のアイテムが入っている

let BLOCK\_WITH\_PINK\_ITEM = 3;//ブロックがあり、ピンクのアイテムが入っている

これでブロックを配置するための4x8のmapが出来上がりました。

■■initblocksColorメソッドについて

4x8のmap配列は生成できましたが、ブロックには色をつけており、map配列にはいろの情報が含まれておりません。ブロックの色の情報を保有しておくcolorMap配列の生成部分についても触れていきます。

colorMap配列は先ほど生成したmap配列と同じサイズのものを生成する必要がありますので、initblocksColorメソッドで、同じようにcols・rowsプロパティの値を使用し、２次元配列を生成します。外側のfor文については、initMapメソッドとほぼ同じですが、内側のfor文内の処理が異なります。色の情報はrgbの3つの値（カラーコード）で表現できます。r,g,bの3つの変数を用意し、それぞれに、util.randメソッドで0~255までのランダムな値を代入します。この生成された値を使って新たな配列を生成し、this.colorMap[i][j]へ代入しています。つまり、３次元配列を生成していることになります。

■■blockDrowメソッドでブロック描画

blockDrowはブロックを描画するためのメソッドです。メソッド内の具体的な処理について解説いたします。

mapプロパティは２次元配列となっており、この配列内の情報を一つづつ取り出してブロックを生成する処理を行いますので、for文を入れ子にして処理します。内側のfor文内で、まずthis.map[i][j]が0でないか（NO\_BLOCKステータスでないか）をif文でチェックし、0でなければブロックを描画していきます。ブロックの描画はutil.drawRectメソッドにブロックを描画する位置、ブロックの幅高さなどの情報、描画する色の情報を渡し行います。色の情報については文字列を渡す必要があるため、

'rgb(' + this.colorMap[i][j][0] + ', ' + this.colorMap[i][j][1] + ', ' + this.colorMap[i][j][2] + ')'

のように値を文字列に連結し、”rgb(xxx,xxx,xxx)”のような文字列を生成し、引数として渡しています。

このメソッドはloop関数内に記述しており、フレーム毎に何度も実行させています。

6-4.ブロックとの衝突

ブロックの配置し、ブロックの場所が決まると、ボールとブロックの衝突判定を行う必要があります。その処理について、解説いたします。

■■ballHitsBlocks関数

この関数でボールとブロックとの衝突を判定しています。ブロックとボールの衝突判定部分をピックアップして解説します。

function　ballHitsBlocks() {

let countBlock = 0;

let hit = false;

for (let i = 0; i < bl.map.length; i++) {①

for (let j = 0; j < bl.map[i].length; j++) {②

if (bl.map[i][j]) {

countBlock++;

let ballTop = ba.posY - ba.radius;

let ballBottom = ba.posY + ba.radius;

let ballLeft = ba.posX - ba.radius;

let ballRight = ba.posX + ba.radius;

let blockTop = (bl.height \* i) + bl.BLOCK\_MARGIN\_TOP;

let blockBottom = (bl.height \* (i + 1)) + bl.BLOCK\_MARGIN\_TOP;

let blockLeft = bl.width \* j;

let blockRight = bl.width \* (j + 1);

//ブロック下辺との当たり判定

if (ballTop < blockBottom && ③ここから

ballTop > blockTop &&

ba.speedY < 0) {

if (blockLeft < ba.posX && ba.posX < blockRight) {

ba.speedY = -ba.speedY;

hit = true;

}

}③ここまで

//ブロック上辺との当たり判定

if (blockTop < ballBottom &&

blockBottom > ballBottom &&

0 < ba.speedY) {

if (blockLeft < ba.posX && ba.posX < blockRight) {

ba.speedY = -ba.speedY;

hit = true;

}

}

//ブロック左辺との当たり判定

if (blockTop < ba.posY &&

ba.posY < blockBottom) {

if (blockLeft < ballRight &&

blockRight > ballRight &&

ba.speedX > 0) {

ba.speedX = -ba.speedX;

hit = true;

}

}

// ブロック右辺との当たり判定

if (blockTop < ba.posY &&

ba.posY < blockBottom) {

if (ballLeft < blockRight &&

ballLeft > blockLeft &&

ba.speedX < 0) {

ba.speedX = -ba.speedX;

hit = true;

}

}

if (hit) { ④ここから

//アイテムを表示させる

let posX = blockLeft + bl.width / 2;

let posY = bl.height \* i + bl.height / 2 + bl.BLOCK\_MARGIN\_TOP;

if (bl.map[i][j] === BLOCK\_WITH\_GREEN\_ITEM) {

itemsAry.push(new Item(posX, posY, 1, LOGO\_GREEN\_PATH, GREEN\_ITEM));

} else if (bl.map[i][j] === BLOCK\_WITH\_PINK\_ITEM) {

itemsAry.push(new Item(posX, posY, 2, LOGO\_PINK\_PATH, PINK\_ITEM));

}

bl.map[i][j] = NO\_BLOCK; ⑤

sf.hitBlock.stop();

sf.hitBlock.play();

hit = false;

}④ここまで

}

}

}

if (!countBlock && gs.isGameStarted) {

gameClearFunc();

}

}

①②の部分でfor文を入れ子にしてブロックを一つづつに衝突判定を行っています。

ブロックには辺が上下左右合わせて4つずつあります。すべてのブロックの4つの辺に対して衝突したかどうかの処理を行っています。

ブロック下辺との当たり判定部分（③）を一例に説明します。

ブロックの一片とボールとの当たり判定方法は自機の上辺とボールとの当たり判定とかなり似ています。

・ボールの上部がブロックの下辺よりも上にある

・ボールの上部がブロックの上辺よりも下にある

・ボールの速度が上向きである

以上の3つの条件を満たせば、衝突する可能性があるとみなし、今度はx座標部分での次の条件判定を行います。

・ボール中心のx座標がのブロックの幅内に収まっている

この条件を満たした場合、ボールとブロックとが衝突したと判断し、衝突時の処理を行います。ボールの速度を逆向き（下向き）へ変化させ、hit変数へtrueを代入します。

■■アイテム表示とブロック削除

hit変数は衝突音を再生させる為に使用しますが、今回はブロックを消す処理と、アイテムを表示させる処理にも使用します。

まずは衝突したブロックにアイテムが格納されているかを判定します。

衝突したblockを表すbl.map[i][j]の値にBLOCK\_WITH\_GREEN\_ITEM定数（つまり2）が格納されている場合、そのブロックは緑のアイテムを保有している事とし、BLOCK\_WITH\_PINK\_ITEM定数（つまり3）が格納されている場合、ピンクのアイテムを保有している事としています。

どちらも処理は同じようなものになるので、緑のアイテム処理のみ解説いたします。

条件分岐でBLOCK\_WITH\_GREEN\_ITEM定数が格納されていた場合、以下の処理が実行されます。

itemsAry.push(new Item(posX, posY, 1, LOGO\_GREEN\_PATH, GREEN\_ITEM));

itemsAryはitemを格納する配列です。この配列に.pushメソッドで引数に渡したオブジェクトを追加格納しています。引数部分ではitemというクラスのインスタンスを新たに生成しています。itemのインスタンスをitemsAryへ格納すると、itemが画面に描画され、上から降ってくるようにアニメーションされます。この部分については第７節で解説いたします。

ボールと衝突したブロックについては、画面から消す必要があります。画面からブロックを消すには、bl.map[i][j] の値へ NO\_BLOCK定数（つまり0）を代入することで実行することができます（⑤）。

【第6節】 スマホジャイロセンサー

このゲームはスマホからも傾き操作でプレイできるようにしてあります。その部分について、解説いたします。

7-1.window. ondeviceorientationについて

■■eventオブジェクト内のgammaプロパティを利用する

スマホからの操作は、window.ondeviceorientationメソッドを使用することで実現ができます。

window.ondeviceorientationメソッド部分は以下の通りとなっています。

window.ondeviceorientation = function(event) {

let gamma = event.gamma; ①

if (gamma < -5) {②ここから

p.moveLeft = true;

} else if(gamma > 5){

p.moveRight = true;

}else{

p.moveLeft = false;

p.moveRight = false;

}②ここまで

};

引数で渡されるeventオブジェクトのgammaプロパティから y 軸を中心にしたデバイスの動きを表す値を取得することができます。

gammaの値が一定以上を超えた場合にp.moveLeft・p.moveRightプロパティへtrueを代入し、時期が傾けた方向に移動します。傾きを戻すとp.moveLeft・p.moveRightプロパティへfalseが代入され、自機の移動が停止します。このようにスマホの傾きの角度によって、自機の操作を可能にしました。

ソースコードの主な部分については解説いたしました。

解説しきれていない部分について気になる方は、ご自身で紐解いてみてください。