Javascriptコース

目次

1. 導入

1.1. まず、Javascriptコースでは何をするの？

このコースでは、Javascriptで簡単なゲームを作っていきます。今回は「CreateJS」というライブラリを使います。

１日目は、Javascriptの基礎知識やCreateJSの使い方、簡単なゲームの作り方を学びます。

２、３日目は、初日で学んだ知識をもとにゲームを作っていきます。何人かのチームに分かれてゲームを作っていきましょう。

最終日は、作ったゲームをみんなでプレイしましょう！

1.2. Javascriptってなあに？

Javascript（ジャバスクリプト）とは、プログラミング言語の１つで、Webゲームを作るときに使われる言語です。Javaと名前は似ていますが全く違う言語です。

1.3. CreateJSってなあに？

CreateJSは、Webゲームをより簡単に作るためのものです。ゲームを作る際にとても便利です。

1.4. Javascriptって何に使われているの？

Webページの多くは、Javascriptで作られています。

2. WebStromを使おう

2.1. WebStormとは

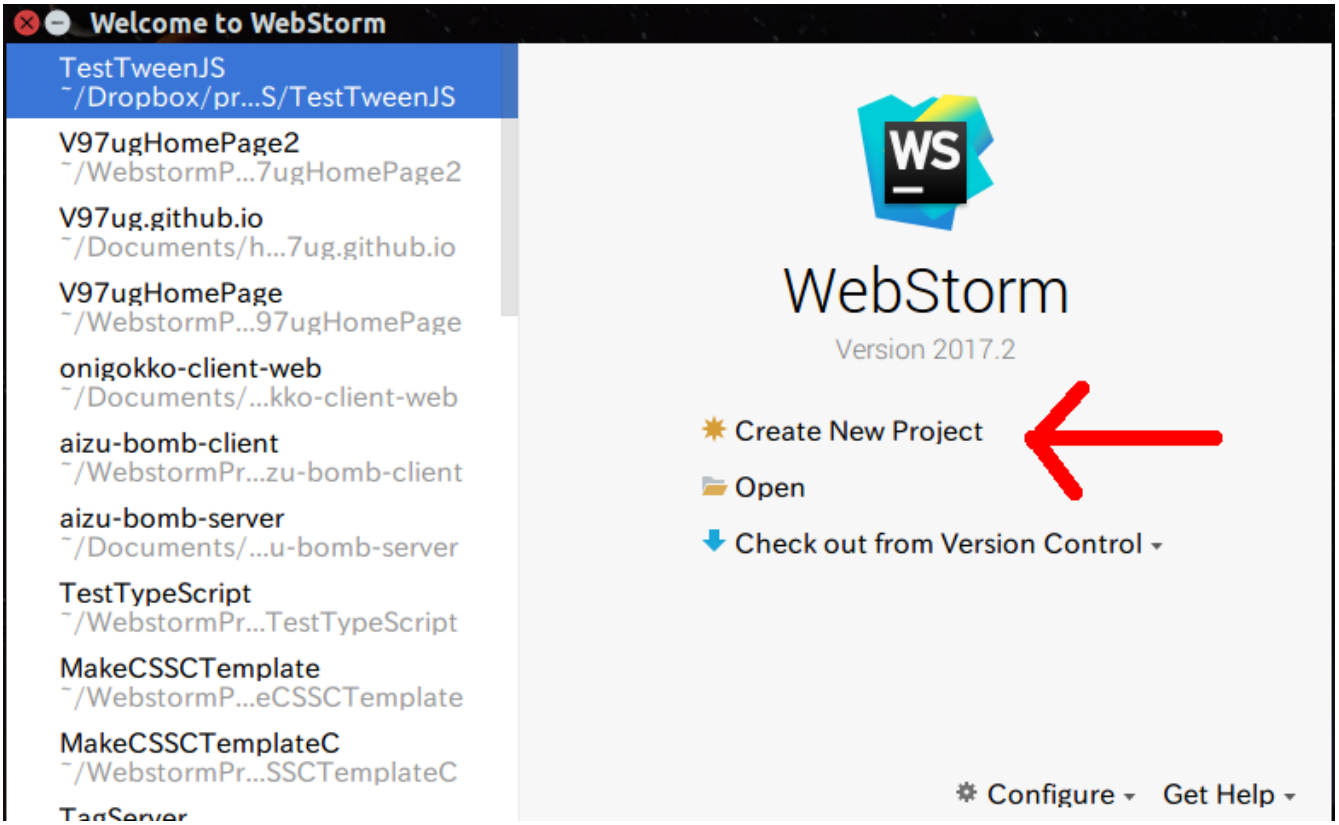
WebStormとは、HTMLやJavascriptのための統合開発環境です。簡単に言うと、これ一つで、WebJavascriptのコーディング、実行までできる便利なものです。

2.2. WebStormの使い方

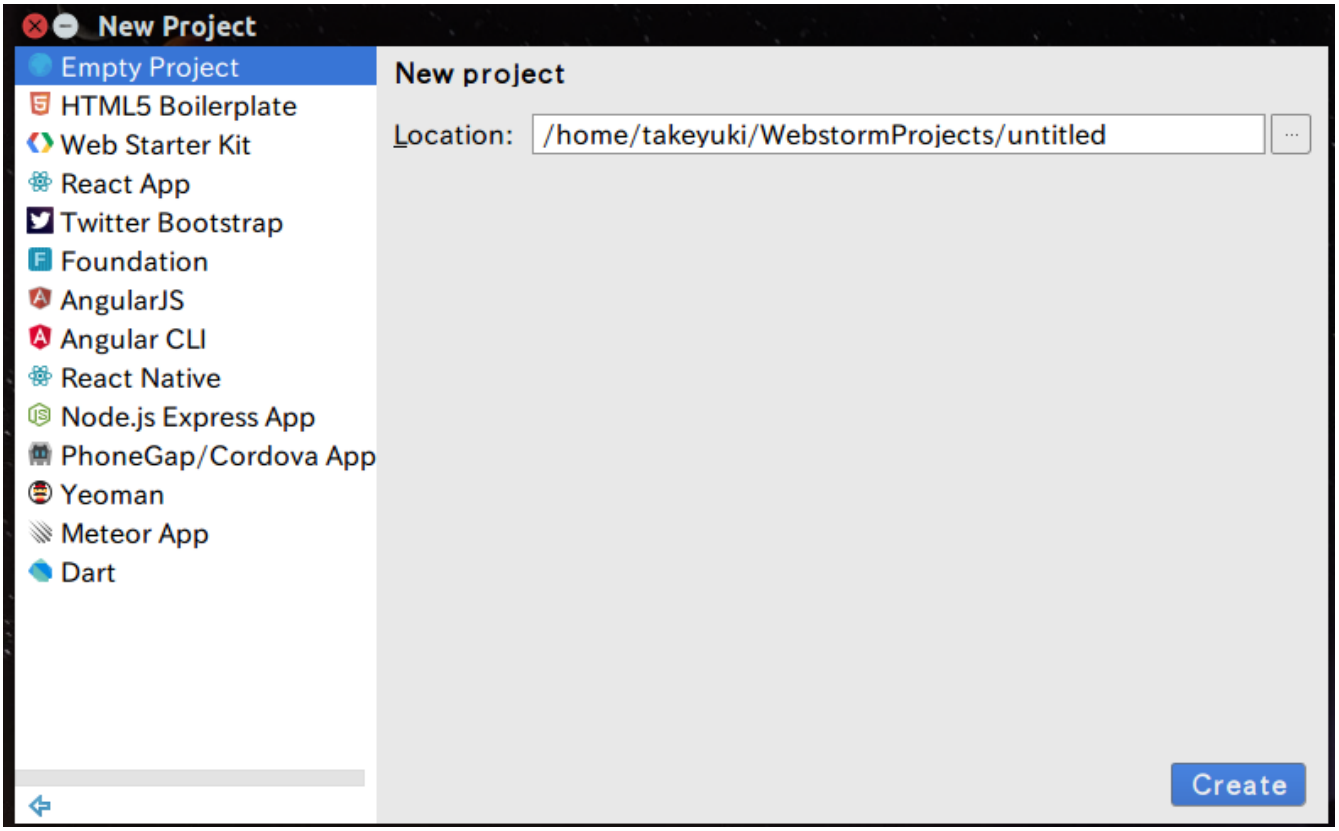
まず、WebSormのアイコンをクリックして、WebStormを起動させます。

2.2.1. 新しいプロジェクトを作る

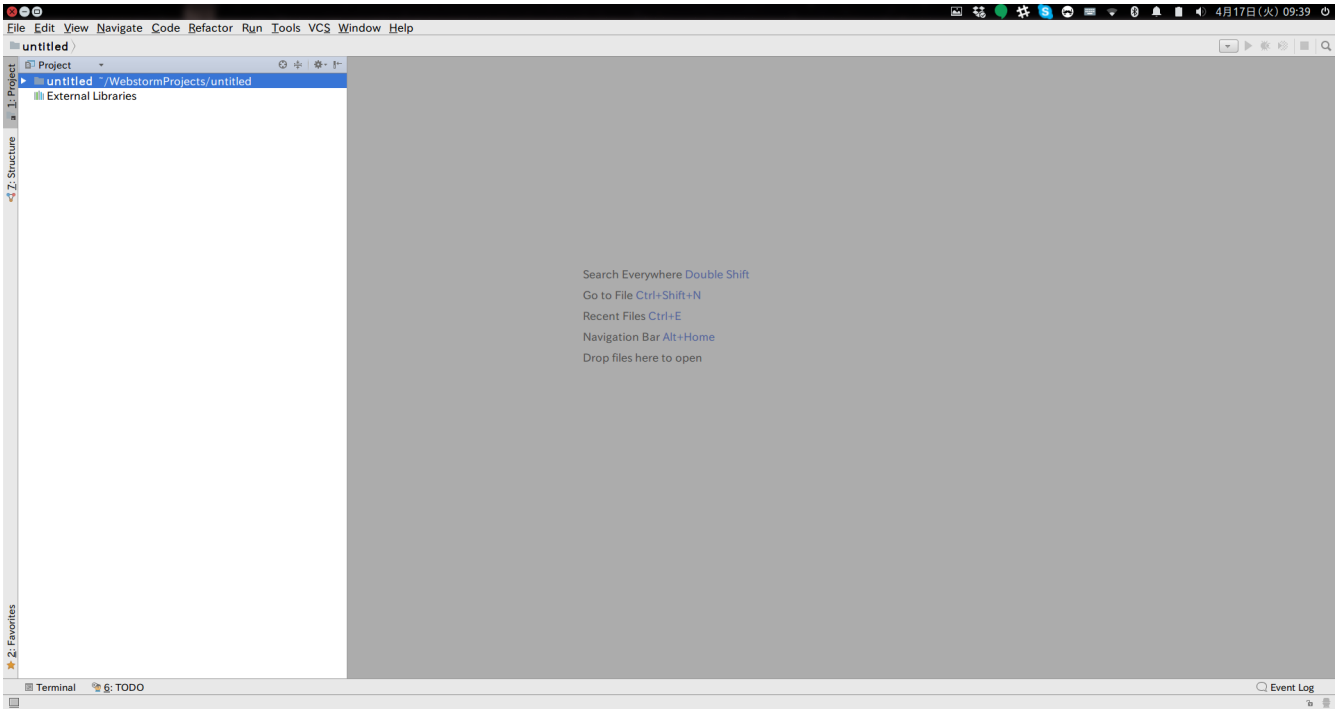
WebStormが起動したら、`Create New Project` をクリックして、新しいプロジェクトを作ります。



次に、プロジェクトを作る場所を指定します。入力されている文字の途中までは変えなくてもいいですが、最後のスラッシュ以降の文字は変えましょう。名前はなんでもいいですが、ここではuntitledにしておきましょう。

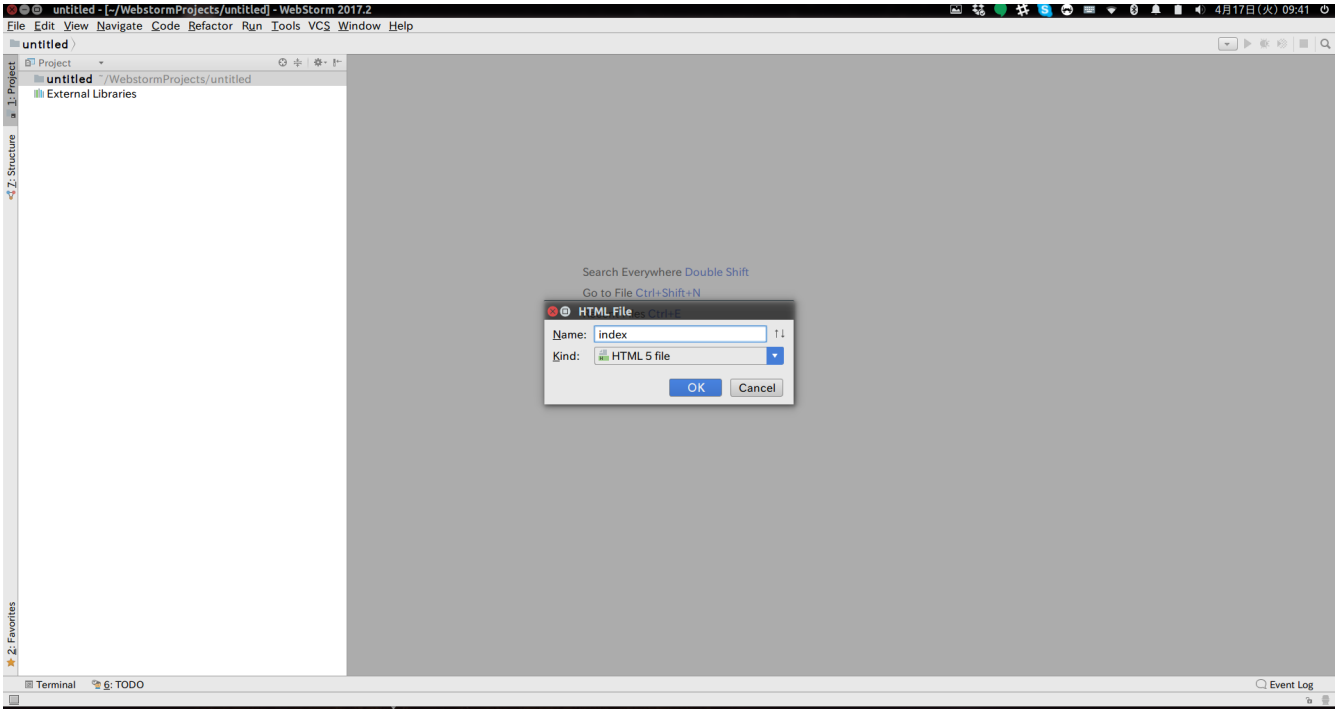


入力ができたら、右下の `Create` ボタンをクリックします。すると、以下の画面が出できたと思います。

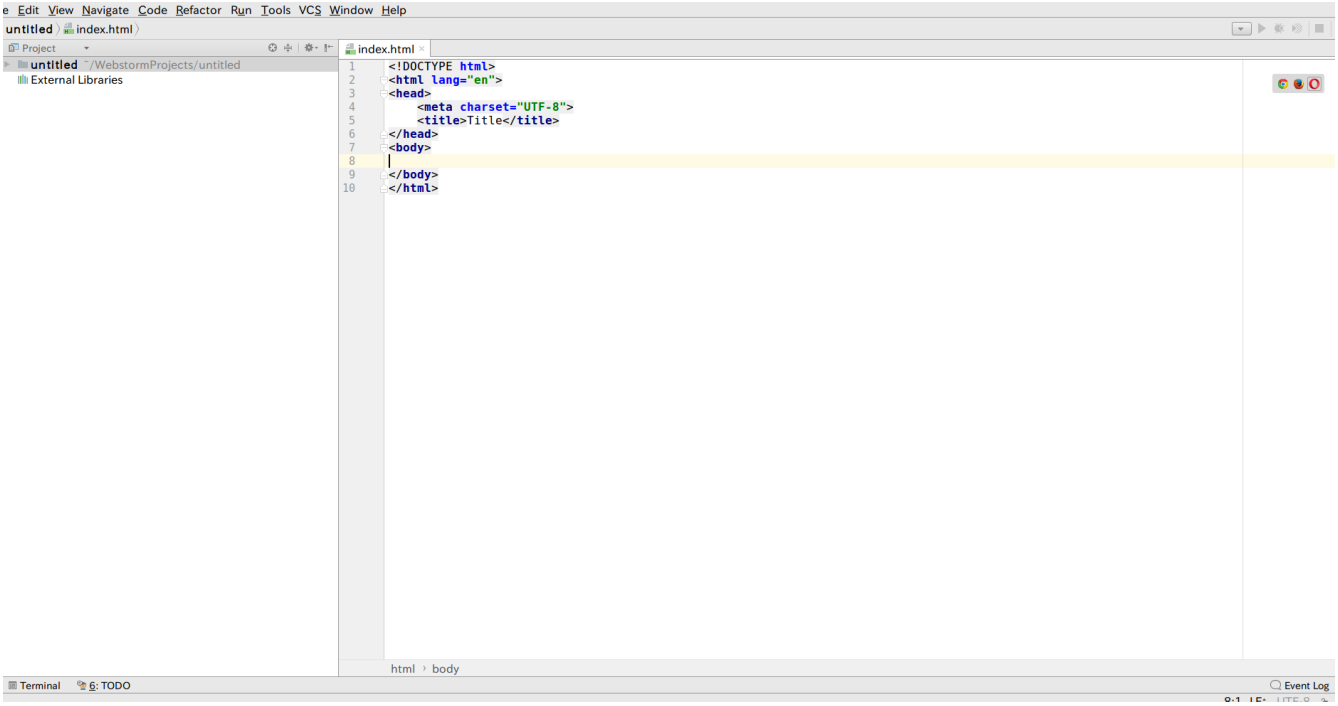


2.2.2. HTMLファイルを作成する

次にHTMLファイルを作成してみましょう。左のエリアのフォルダを右クリックして、New → HTML Fileを選択します。すると、小さいウィンドウが出てくるので、そこの Name という欄にファイル名を入力しましょう。HTMLのファイル名はなんでもいいですが、ここでは名前をindex.htmlとしておきます。

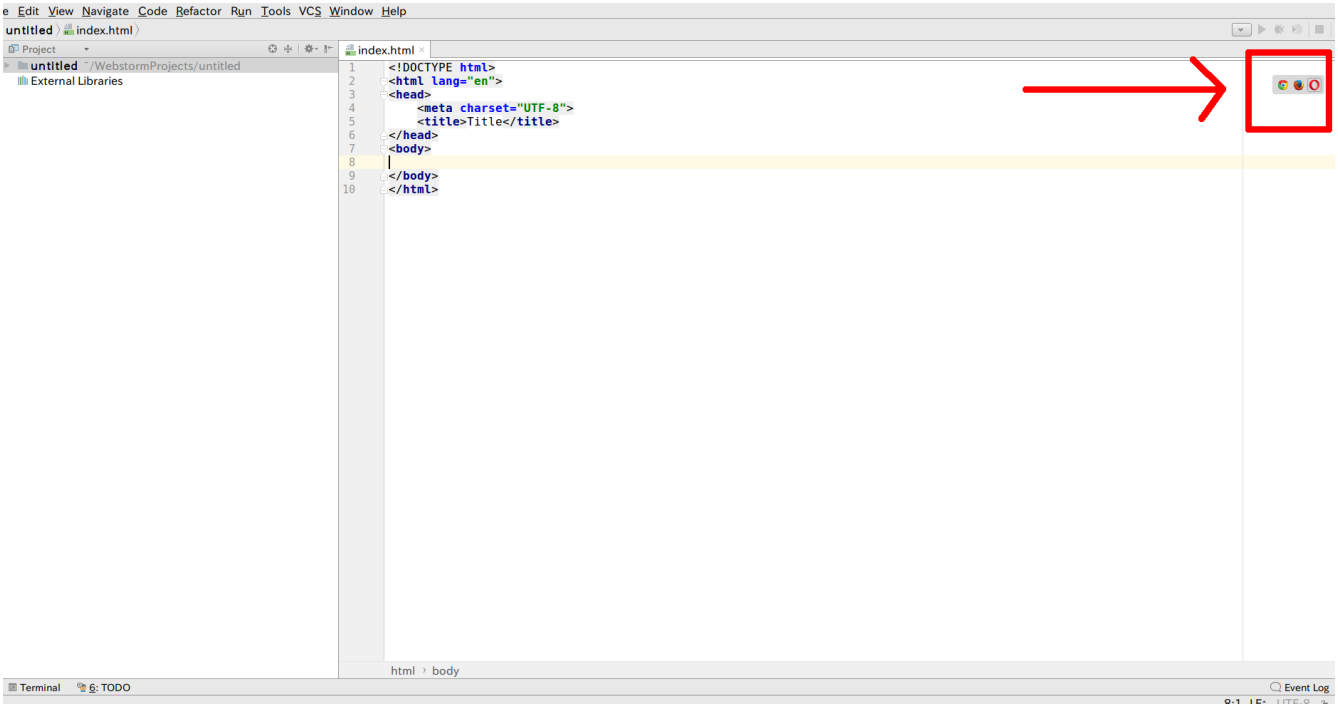


名前が入力できたら、 OK ボタンをクリックします。すると以下のようにHTMLファイルが作られます。



2.2.3. ブラウザ上で表示してみよう

それでは、このHTMLファイルをブラウザ上で表示してみましょう。ブラウザで表示するためには、左上のほうにある、小さいアイコンたちがあるのがわかりますか。そのアイコンのうち1つをクリックしてください。



すると、真っ白いページが表示されたと思います。上のバーのところに `Title` と表示されていればOKです。

ここからゲーム画面などを追加していくので、今は真っ白いページで大丈夫です。

2.2.4. タイトルを変えて、ブラウザで表示してみよう

titleタグで囲まれたところを変更しましょう。こうすることで、ブラウザで表示したときに、ページのタイトルが変わります。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>タイトル！</title>
</head>
<body>

</body>
</html>
```

3. CreateJSで作られたゲームを改造してみよう

この賞では、CreateJSを深く学ぶ前に、CreateJSで作られたゲームのサンプルを使って遊んでみましょう。遊んだら、このゲームを少し改造してみましょう。

3.1. CreateJSのゲームのサンプル

以下にサンプルのソースコードの載せます。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>shooting</title>
    <script src="https://code.createjs.com/1.0.0/createjs.min.js"></script>
    <script>
        //createJSの読み込みが終わってから、init関数をよぶ。
        window.addEventListener("load", init);

        function init() {
            let stage = new createjs.Stage("myCanvas");
            let count = 0;
            let enemyList = [];
            let bulletList = [];
            let scene = 0;

            let bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0, 0, 960, 540);
            stage.addChild(bg);

            let player = new createjs.Shape();
```

```
player.graphics.beginFill("white").drawCircle(0, 0, 10);
stage.addChild(player);

let titleText = new createjs.Text("Title", "40px sans-serif", "white");
titleText.x = 480;
titleText.y = 50;
titleText.textAlign = "center";
stage.addChild(titleText);

stage.addEventListener("click", handleClick);

createjs.Ticker.setFPS(60);
createjs.Ticker.addEventListener("tick", handleTick);

function handleClick() {
    if (scene === 0) {
        scene = 1;
        stage.removeChild(titleText)
    } else {

        let bullet = new createjs.Shape();
        bullet.graphics.beginFill("white").drawCircle(0, 0, 3);
        bullet.x = player.x;
        bullet.y = player.y;

        bulletList.push(bullet);
        stage.addChild(bullet);
    }
}

function handleTick() {
    if(scene === 0){
        stage.update();
    }

    if(scene === 1) {

        player.x = stage.mouseX;
        player.y = stage.mouseY;

        if (count % 100 === 0) {
            let enemy = new createjs.Shape();
            enemy.graphics.beginFill("red").drawCircle(0, 0, 10);

            enemy.x = 960;
            enemy.y = 540 * Math.random();

            stage.addChild(enemy);
            enemyList.push(enemy);
        }
        count = count + 1;
```

```
                    for (let i = 0; i < enemyList.length; i++) {
                        enemyList[i].x -= 2;
                    }

                    for (let i = 0; i < bulletList.length; i++) {
                        bulletList[i].x += 10;
                    }

                    for (let i = 0; i < enemyList.length; i++) {
                        let enemyLocal = enemyList[i].localToLocal(0, 0, player);
                        if (player.hitTest(enemyLocal.x, enemyLocal.y)) {
                            gameOver();
                        }
                    }

                    for (let i = 0; i < bulletList.length; i++) {
                        for (let j = 0; j < enemyList.length; j++) {
                            let localPoint = bulletList[i].localToLocal(0, 0,
enemyList[j]);
                            if (enemyList[j].hitTest(localPoint.x, localPoint.y)) {
                                stage.removeChild(bulletList[i]);
                                bulletList.splice(i, 1);

                                stage.removeChild(enemyList[j]);
                                enemyList.splice(j, 1);
                            }
                        }
                    }

                    stage.update()
                }
            }

//            function title() {
//                let titleText = new createjs.Text("", "24px sans-serif", "white");
//                stage.addChild(titleText);
//            }
//
//            function titleClick() {
//                scene = 1;
//            }

            function gameOver() {
                alert("ゲームオーバー");

                createjs.Ticker.removeAllEventListeners();
                stage.removeAllEventListeners();
            }
        }
```

```
    </script>
</head>
<body>
<canvas id="myCanvas" width="960" height="540"></canvas>
</body>
</html>
```

3.2. 少し改造してみる。

背景の色、敵の数、プレイヤーの色、プレイヤーの動きやすさなどを変えてみましょう。

4. CreateJSでシューティングゲームを作ってみる。

さて、前章でシューティングゲームを少し作り変えてみました。この章では、そのゲームがどのような過程でつくられているのかを詳しく見ていきましょう。

4.1. CreateJSを使うための下準備

4.1.1. HTMLファイルの作成

まずは、 `New → HTML File` でHTMLファイルを作成します。この中にゲームを作るためのコードを記述していきます。HTMLファイルを作ると、以下のようなものが出来上がると思います。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>shooting</title>
</head>
<body>

</body>
</html>
```

titleはshootingにしておきました。自分で好きなタイトルに変えて構いません。

4.1.2. canvasタグの追加

次は、ゲーム画面をどのくらいの大きさにするかを決めます。 `<canvas id="myCanvas" width="960" height="540"></canvas>` というコードを以下のようにbodyタグ内に追加します。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>shooting</title>
</head>
<body>
    <canvas id="myCanvas" width="960" height="540"></canvas>
</body>
</html>
```

canvasタグは、文字通り、キャンバスのごとく自由に絵をかくことができます。そのcanvasの中にゲームを描いていくような感じです。

canvasタグのidはここでは、myCanvasとしておきます。

また、width属性で横幅を調整できて、height属性で高さを調整できます。ここでは、横が960、高さが540となっていますね。

4.1.3. CreateJSを使えるようにする

CreateJSを使うためには、それを読み込む必要があります。どう読み込むかといいますと、以下のようにscriptタグを追加して読み込みます。6行目に追加してある、`<script src="https://code.createjs.com/1.0.0/createjs.min.js"></script>` ですね。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>shooting</title>
    <script src="https://code.createjs.com/1.0.0/createjs.min.js"></script>
</head>
<body>
<canvas id="myCanvas" width="960" height="540"></canvas>
</body>
</html>
```

4.1.4. 関数を定義する

次に、ゲームのプログラムを記述する関数を定義します。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>shooting</title>
    <script src="https://code.createjs.com/1.0.0/createjs.min.js"></script>
    <script>
        //createJSの読み込みが終わってから、init関数をよぶ。
        window.addEventListener("load", init);

        function init() {

        }
    </script>
</head>
<body>
<canvas id="myCanvas" width="960" height="540"></canvas>
</body>
</html>
```

`window.addEventListener("load", init);` は、CreateJSを読み込んでから、init関数を呼び出します。つまり、CreateJSを完全に読み込んでから、ゲームが始まります。もしCreateJSを読み込む前にゲームが始まってしまうと大変ですから、そういうことがないようにしています。

4.2. ゲーム画面の表示

4.2.1. Stageの作成

CreateJSでは、まずStageという生地をベースに他のオブジェクトを追加します。ですのでまず最初はStageを作ります。

```
        function init() {
            var stage = createjs.Stage("myCanvas");
        }
```

（ここからは、init関数の中だけを変更していくので、その他のHTMLは省略します。）

ここで注意したいのが、new　createjs.Stageの中の文字。これは、canvasタグで指定したidと同じにしなければいけません。この場合だと、 `myCanvas` とする必要があります。

4.2.2. 背景の表示

このままでは、Stageに何も追加されていないので、次はゲーム画面の背景を追加してみましょう。背景の色はここでは黒色にしています。

```
        function init() {
            var stage = new createjs.Stage("myCanvas");

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            stage.update()
        }
```

`new createjs.Shape()` で、シェイプを作ります。ここで注意してほしいのは、**new**をつけ忘れないようにしてくださいね。それで、これを変数として保存しておきたいので、 `var stage = new createjs.Stage("myCanvas");` というふうに変数に格納します。

`bg.graphics.beginFill("black").drawRect(0,0,960,540);` は、シェイプの特徴を詳しく記述しています。ここでは背景を指しますね。ではどんなシェイプかといいますと、背景が黒で、座標(0,0)が始点の幅960・高さ540の長方形。これで背景を表しています。

あとはその背景をstageに追加します。 `stage.addChild(bg);` でステージに背景を追加できます。これをしないと、背景が表示されないことになります。

また、最後に `stage.update()` で毎回背景を描画してくれます。

これで実行してみて、黒い四角形が表示されていればOKです。

4.3. プレイヤーの表示

```
        function init() {
            var stage = new createjs.Stage("myCanvas");

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            var player = new createjs.Shape();
            player.graphics.beginFill("white").drawCircle(100,100,10);
            stage.addChild(player);

            stage.update()
        }
```

背景と同じ要領で、プレイヤーもシェイプで作っていきます。プレイヤーの色は白にして、形は丸にでもしておきましょう。

この `drawCircle(100,100,10)` の意味は、中心の座標が(100,100)で、半径が10の円という意味です。

実行してみると、丸い円が表示されます。

4.4. プレイヤーがマウスで動けるようにする

では、このプレイヤーを動かしてみましょう。

4.4.1. tickイベントを作る

まず、プレイヤーを動かすために、更新処理をする必要があって、それをするために、createjs.Tickerクラスのtickイベントを使います。

```
        function init() {
            var stage = new createjs.Stage("myCanvas");

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            var player = new createjs.Shape();
            player.graphics.beginFill("white").drawCircle(100,100,10);
            stage.addChild(player);

            createjs.Ticker.setFPS(60);
            createjs.Ticker.addEventListener("tick", handleTick);

            function handleTick(){
                stage.update()
            }
        }
```

`createjs.Ticker.setFPS(60);` とすることで、１秒間に60回の頻度で画面が更新されていきます。また、`handleTick` という関数を作って、その中に、 `stage.update()` を入れましょう。

4.4.2. プレイヤーとマウスの動きを同期させる

次に、プレイヤーがマウスの動きと同じになるようにしてみましょう。

```
        function init() {
            var stage = new createjs.Stage("myCanvas");

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            var player = new createjs.Shape();
            player.graphics.beginFill("white").drawCircle(100,100,10);
            stage.addChild(player);

            createjs.Ticker.setFPS(60);
            createjs.Ticker.addEventListener("tick", handleTick);

            function handleTick(){
                player.x = stage.mouseX;
                player.y = stage.mouseY;

                stage.update()
            }
        }
```

`player.x = stage.mouseX;` で、プレイヤーのx座標をマウスのx座標に変えています。y座標も同様です。

こうすることで、マウスについてくるようになりますが、、、

何かおかしいですよね。マウスの位置とプレイヤーの位置がずれているような。。 実は、プレイヤーを円で描画するときに、 `player.graphics.beginFill("white").drawCircle(100,100,10);` としましたよね。注目してほしいのが 、drawCircleのところ。中心座標が（100,100）となっていますが、実はこれ、相対的な座標になっています。つまり、マウスで動かしても、 マウスの位置+100 の位置に円が描画されることになってしまうのです。

これを修正するには、drawCircleのところを `drawCircle(0,0,10)` にしてしまいましょう。

```
        function init() {
            var stage = new createjs.Stage("myCanvas");

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            var player = new createjs.Shape();
            player.graphics.beginFill("white").drawCircle(0,0,10);
            stage.addChild(player);

            createjs.Ticker.setFPS(60);
            createjs.Ticker.addEventListener("tick", handleTick);

            function handleTick(){
                player.x = stage.mouseX;
                player.y = stage.mouseY;

                stage.update()
            }
        }
```

こうすれば、ちゃんとついてきますね！

4.5. 敵を出現させる

次は敵を出現させてみましょう。今回は、100フレームに１体、ランダムな位置から敵を出現させてみます。

4.5.1. 100フレームに１体敵を出現させる

まずは、フレームを数える必要がありますね。そこで、 `count` という変数を作って、フレームを数えまし

ょう。フレーム数を数えるには、毎回 count を１ずつ足していく必要がありますね。

```
        function init() {
            var stage = new createjs.Stage("myCanvas");
            var count = 0;

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            var player = new createjs.Shape();
            player.graphics.beginFill("white").drawCircle(0,0,10);
            stage.addChild(player);

            createjs.Ticker.setFPS(60);
            createjs.Ticker.addEventListener("tick", handleTick);

            function handleTick(){
                player.x = stage.mouseX;
                player.y = stage.mouseY;

                if(count % 100 === 0){

                }
                count = count + 1;


                stage.update()
            }
        }
```

count の変数宣言は、init関数の最初の方で行います。handleTick関数内で count を宣言してしまうと、ローカル変数となってしまい、 count が毎回0で初期化されてしまいます。毎回0になってしまってはしょうがないので、init関数の最初の方で宣言することによって、countの情報が無くならずに、countが増え続けます。

if文は、丸括弧内の式の条件が正しければその下の処理を実行するので、ここでは count % 100 === 0 であれば、下の中括弧の処理を実行します。では、count % 100 === 0 とは何なのでしょうか。

まず、 count % 100 の意味は、「 count を100で割った余り」です。例えば、103 % 100 なら答えは3だし、 200 % 100 なら答えは0です。

そうすると、count % 100 === 0 の意味は、「count を100で割った余りが0になるかどうか」です。つまり、 count が100の倍数になったときに、敵が出現します。

4.5.2. 右端から敵を出現させる

次に、プレイヤーを作った要領で敵を作ってみましょう。

```
            function handleTick(){
                player.x = stage.mouseX;
                player.y = stage.mouseY;

                if(count % 100 === 0){
                    var enemy = new createjs.Shape();
                    enemy.graphics.beginFill("red").drawCircle(0,0,10);

                    enemy.x = 960;
                    enemy.y = 540 * Math.random();

                    stage.addChild(enemy);
                }
                count = count + 1;


                stage.update()
            }
```

敵を作る時、位置を決めてしまいます。ここでは、 `enemy.x = 960;` でx座標を960(右端)、 `enemy.y = 540 * Math.random();` でy座標をランダムにしています。

`Math.random()` は、0以上1未満の小数を返します。よって、 `540 * Math.random();` は、0以上540未満の数になります。

4.5.3. 敵を保存しておく「配列」

ここで、敵を作るだけだと、後に敵を動かすことができなくなるので、敵を保存しておく「配列」用意します。この配列もcountと同じように、init関数の最初の方に宣言します。

```
        function init() {
            var stage = new createjs.Stage("myCanvas");
            var count = 0;
            var enemyList = [];

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            var player = new createjs.Shape();
            player.graphics.beginFill("white").drawCircle(0,0,10);
            stage.addChild(player);

            createjs.Ticker.setFPS(60);
            createjs.Ticker.addEventListener("tick", handleTick);

            function handleTick(){
                player.x = stage.mouseX;
                player.y = stage.mouseY;

                if(count % 100 === 0){
                    var enemy = new createjs.Shape();
                    enemy.graphics.beginFill("red").drawCircle(0,0,10);

                    enemy.x = 960;
                    enemy.y = 540 * Math.random();

                    stage.addChild(enemy);
                    enemyList.push(enemy);
                }
                count = count + 1;

                stage.update()
            }
        }
```

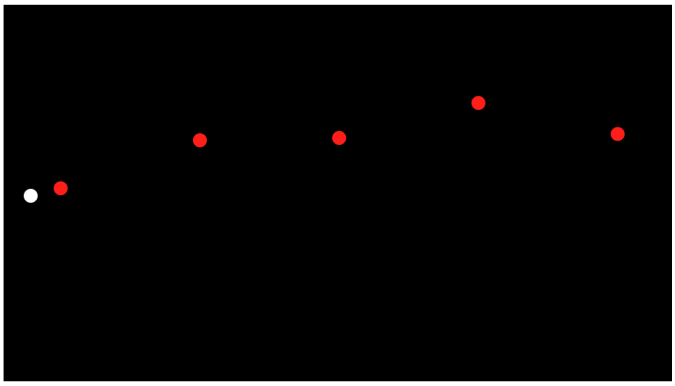
if文の最後のほうに、`enemyList.push(enemy)` と書いてありますね。これは、 `enemyList` という配列に `enemy` を追加していく処理です。

4.5.4. 全ての敵を動かす

敵１つずつのx座標を左にずらせば大丈夫そうですね。for文を使って1つずつ配列にアクセスしていって、x座標の値を1ずつ減らしていきましょう。

```
function handleTick(){
    player.x = stage.mouseX;
    player.y = stage.mouseY;

    if(count % 100 === 0){
        var enemy = new createjs.Shape();
        enemy.graphics.beginFill("red").drawCircle(0,0,10);

        enemy.x = 960;
        enemy.y = 540 * Math.random();

        stage.addChild(enemy);
        enemyList.push(enemy);
    }
    count = count + 1;

    for(var i = 0; i < enemyList.length; i++){
        enemyList[i].x -= 2;
    }

    stage.update()
}
```

これで敵もちゃんと動くと思います。



4.6. 敵とプレイヤーの当たり判定

敵とプレイヤーの当たり判定をつけるために、CreateJSで用意されている `hitTest` を使います。これを使うと、点とシェイプの当たり判定がわかります。

```
        function handleTick(){
            player.x = stage.mouseX;
            player.y = stage.mouseY;

            if(count % 100 === 0){
                var enemy = new createjs.Shape();
                enemy.graphics.beginFill("red").drawCircle(0,0,10);

                enemy.x = 960;
                enemy.y = 540 * Math.random();

                stage.addChild(enemy);
                enemyList.push(enemy);
            }
            count = count + 1;

            for(var i = 0; i < enemyList.length; i++){
                enemyList[i].x -= 2;
            }

            for(var i = 0; i < enemyList.length; i++){
                var enemyLocal = enemyList[i].localToLocal(0,0, player);
                if(enemyList[i].hitTest(player.x, player.y)){

                }
            }

            stage.update()
        }
```

for文で敵全てとプレイヤーの当たり判定を確認します。 `hitTest` の括弧の中は、プレイヤーのx座標とy座標をそれぞれ入れましょう。

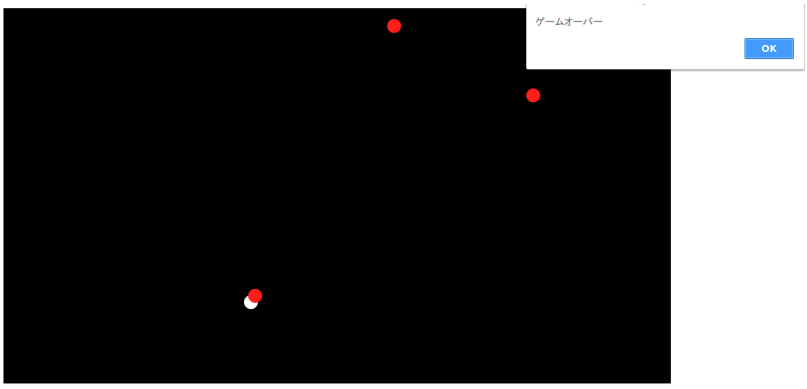
また、`enemyList[i].localToLocal(0,0, player);` は何かと言いますと、enemyをローカル座標に変えています。こうすることで、hitTestができるようになります。

4.7. ゲームオーバー画面に移動する

敵と接触したら、今度はゲームオーバー画面に移動してみましょう。 `gameOver` 関数を作って、当たり判定をしたif文の中に `gameOver` 関数を呼び出す処理をしてみましょう。

```
function handleTick(){
    player.x = stage.mouseX;
    player.y = stage.mouseY;

    if(count % 100 === 0){
        var enemy = new createjs.Shape();
        enemy.graphics.beginFill("red").drawCircle(0,0,10);

        enemy.x = 960;
        enemy.y = 540 * Math.random();

        stage.addChild(enemy);
        enemyList.push(enemy);
    }
    count = count + 1;

    for(var i = 0; i < enemyList.length; i++){
        enemyList[i].x -= 2;
    }

    for(var i = 0; i < enemyList.length; i++){
        var enemyLocal = enemyList[i].localToLocal(0,0, player);
        if(player.hitTest(enemyLocal.x, enemyLocal.y)){
            gameOver();
        }
    }

    stage.update()
}

function gameOver(){
    alert("ゲームオーバー");

    createjs.Ticker.removeAllEventListeners();
    stage.removeAllEventListeners();
}
```

`gameOver`　関数内では、ゲームオーバーというポップアップを表示して、Tickerとstageの全てのイベントリスナーを消しています。つまり、ゲームを完全に止めるという処理です。



4.8. プレイヤーが弾を発射できるようにする

次に、プレイヤーがクリックで弾を発射できるようにしましょう。

4.8.1. マウスイベントの登録

クリックをして何かをするためには、stageにマウスイベントの登録が必要となります。

```
        function init() {
            var stage = new createjs.Stage("myCanvas");
            var count = 0;
            var enemyList = [];

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            var player = new createjs.Shape();
            player.graphics.beginFill("white").drawCircle(0,0,10);
            stage.addChild(player);

            stage.addEventListener("click", handleClick);

            createjs.Ticker.setFPS(60);
            createjs.Ticker.addEventListener("tick", handleTick);

            function handleClick() {

            }
```

`stage.addEventListener("click", handleClick);` をすることで、stageにclickのイベントが登録されます。つまり、クリックをした時の処理がかけるようになります。その処理は、新たに作った `handleClick`

関数で行います。

4.8.2. クリックで弾を出現させる

`handleClick` 関数内で、弾を出現させる処理をかいてみましょう。

```
        function init() {
            var stage = new createjs.Stage("myCanvas");
            var count = 0;
            var enemyList = [];
            var bulletList = [];

            var bg = new createjs.Shape();
            bg.graphics.beginFill("black").drawRect(0,0,960,540);
            stage.addChild(bg);

            var player = new createjs.Shape();
            player.graphics.beginFill("white").drawCircle(0,0,10);
            stage.addChild(player);

            stage.addEventListener("click", handleClick);

            createjs.Ticker.setFPS(60);
            createjs.Ticker.addEventListener("tick", handleTick);

            function handleClick() {
                var bullet = new createjs.Shape();
                bullet.graphics.beginFill("white").drawCircle(0, 0, 3);
                bullet.x = player.x;
                bullet.y = player.y;

                bulletList.push(bullet);
                stage.addChild(bullet);
            }
```

敵を作ったのと同じように、弾も作っていきましょう。座標はプレイヤーの座標と同じでいいですね。敵で `enemyList` を作ったように、弾も `bulletList` という配列をinitの最初の方に宣言して、その配列に順次格納していくようにしましょう。

4.8.3. 弾を動かす

敵を動かしたように、弾も同じ方法で動かしましょう。今度はhandleTick内を修正します。

```
        function handleTick() {
            player.x = stage.mouseX;
            player.y = stage.mouseY;

            if(count % 100 === 0){
                var enemy = new createjs.Shape();
                enemy.graphics.beginFill("red").drawCircle(0,0,10);

                enemy.x = 960;
                enemy.y = 540 * Math.random();

                stage.addChild(enemy);
                enemyList.push(enemy);
            }
            count = count + 1;

            for(var i = 0; i < enemyList.length; i++){
                enemyList[i].x -= 2;
            }

            for(var i = 0; i < bulletList.length; i++){
                bulletList[i].x += 10;
            }

            for(var i = 0; i < enemyList.length; i++){
                var enemyLocal = enemyList[i].localToLocal(0,0,player);
                if(player.hitTest(enemyLocal.x, enemyLocal.y)){
                    gameOver();
                }
            }

            stage.update()
        }
```

4.8.4. 弾と敵の当たり判定をつける

プレイヤーと敵の当たり判定をつけたやり方と同じように、全ての弾と全ての敵との当たり判定をします。そのために、2重のfor文を使います。

```
            function handleTick() {
                player.x = stage.mouseX;
                player.y = stage.mouseY;

                if(count % 100 === 0){
                    var enemy = new createjs.Shape();
                    enemy.graphics.beginFill("red").drawCircle(0,0,10);

                    enemy.x = 960;
                    enemy.y = 540 * Math.random();

                    stage.addChild(enemy);
                    enemyList.push(enemy);
                }
                count = count + 1;

                for(var i = 0; i < enemyList.length; i++){
                    enemyList[i].x -= 2;
                }

                for(var i = 0; i < bulletList.length; i++){
                    bulletList[i].x += 10;
                }

                for(var i = 0; i < enemyList.length; i++){
                    var enemyLocal = enemyList[i].localToLocal(0,0, player);
                    if(player.hitTest(enemyLocal.x, enemyLocal.y)){
                        gameOver();
                    }
                }

                for(var i = 0; i < bulletList.length; i++){
                    for(var j = 0; j < enemyList.length; j++){
                        var localPoint = bulletList[i].localToLocal(0,0,enemyList[j]);
                        if(enemyList[j].hitTest(localPoint.x, localPoint.y)){

                        }
                    }
                }

                stage.update()
            }
```

4.8.5. 弾が当たったら敵を消す

あとは、 `stage.removeChild` で、stageから弾と敵を削除します。配列からも、`spice` で削除します。

```
            function handleTick() {
                player.x = stage.mouseX;
                player.y = stage.mouseY;

                if(count % 100 === 0){
                    var enemy = new createjs.Shape();
                    enemy.graphics.beginFill("red").drawCircle(0,0,10);

                    enemy.x = 960;
                    enemy.y = 540 * Math.random();

                    stage.addChild(enemy);
                    enemyList.push(enemy);
                }
                count = count + 1;

                for(var i = 0; i < enemyList.length; i++){
                    enemyList[i].x -= 2;
                }

                for(var i = 0; i < bulletList.length; i++){
                    bulletList[i].x += 10;
                }

                for(var i = 0; i < enemyList.length; i++){
                    var enemyLocal = enemyList[i].localToLocal(0,0, player);
                    if(player.hitTest(enemyLocal.x, enemyLocal.y)){
                        gameOver();
                    }
                }

                for(var i = 0; i < bulletList.length; i++){
                    for(var j = 0; j < enemyList.length; j++){
                        var localPoint = bulletList[i].localToLocal(0,0,enemyList[j]);
                        if(enemyList[j].hitTest(localPoint.x, localPoint.y)){
                            stage.removeChild(bulletList[i]);
                            bulletList.splice(i, 1);

                            stage.removeChild(enemyList[j]);
                            enemyList.splice(j, 1);
                        }
                    }
                }

                stage.update()
            }
```

`bulletList.splice(i, 1);` は、配列のi番目の要素から1つだけを削除します。ここでは、敵に当たった弾が削除されますね。ちなみに、`bulletList.splice(i, 2);` にすると、配列のi番目の要素とその次の要素

の2つの要素が削除されてしまいます。

5. キー入力でプレイヤーを操作してみよう

ここまではマウス操作でプレイヤーを動かしましたが、今度はキーボードで操作できるようにしてみましょう。

5.1. プレイヤーを動かす

まず、init関数の中に以下のものを追加します。

```
window.addEventListener("keydown", handleKeydown);
window.addEventListener("keyup", handleKeyUp);

function handleKeydown(event){
  // キーが押された時の処理
}

function handleKeyUp(event) {
  // キーを離した時の処理
}
```

ここから以下のように、キーが押された時にtrue、キーを離したときにfalseになる変数をそれぞれ用意して、処理をかきます。

```
            window.addEventListener("keydown", handleKeyDown);
            window.addEventListener("keyup", handleKeyUp);

            const SPACE = 32;
            const LEFT = 37;
            const UP = 38;
            const RIGHT = 39;
            const DOWN = 40;

            let isSpace = false;
            let isLeft = false;
            let isUp = false;
            let isRight = false;
            let isDown = false;

            function handleKeyDown(event) {
                let keyCode = event.keyCode;

                if (keyCode === SPACE) {
                    isSpace = true;
                }
                else if (keyCode === LEFT) {
                    isLeft = true;
```

```
            }
            else if (keyCode === UP) {
                isUp = true;
            }
            else if (keyCode === RIGHT) {
                isRight = true;
            }
            else if (keyCode === DOWN) {
                isDown = true;
            }
        }

        function handleKeyUp(event) {
            let keyCode = event.keyCode;

            if (keyCode === SPACE) {
                isSpace = false;
            }
            else if (keyCode === LEFT) {
                isLeft = false;
            }
            else if (keyCode === UP) {
                isUp = false;
            }
            else if (keyCode === RIGHT) {
                isRight = false;
            }
            else if (keyCode === DOWN) {
                isDown = false;
            }
        }
```

プログラムが長く見えますが、５回同じようなことをしているだけなので、１個ずつみていきましょう。

まず、handleKeyDown関数内の `let keyCode = event.keyCode;` は、押されたキーが何であるかを取得して（厳密にはキーコードをとってきます）、それをkeyCode変数に格納します。

スペースキーのキーコードは32らしいので、 `const SPACE = 32` で、SPACEという 定数 を作ります。定数というのは、あとから値を変更できないもののことです。これでSPACEという定数を使う限り、スペースキーのキーコードが32であることが保証されるので、何か変更したくない値を作りたいときは定数を使いましょう。

それから、`if (keyCode === SPACE) {}` でキーコードがスペースキーであるかを判定して、もしそうであれば、isSpace変数をtrueにします。こうすることで、スペースキーが押されたという状態を表すことができます。キーを離した場合も同様です。

他のキーについても同様のことが言えます。

あとは、isSpace, isUpなどの変数をどう使うかです。以下のように使います。前の章で

```
player.x = stage.mouseX;
player.y = stage.mouseY;
```

としていたところを、

```
if(isLeft === true) {
    player.x -= 3;
}
if(isUp === true){
    player.y -= 3;
}
if(isRight === true){
    player.x += 3;
}
if(isDown === true) {
    player.y += 3;
}
```

のようにします。isLeftがtrueなら（左キーが押されているなら）、プレイヤーのx座標を-3します（プレイヤーを左に移動させます）。他の方向でも同じようにすれば、キー入力でプレイヤーを動かすことができます。

5.2. スペースキーを押して弾を発射させる

スペースキーを押したら、弾を出現させる処理を書けばいいので、さっき書いたコードの下あたりに以下のコードを追加しましょう。

```
if(isSpace === true) {
    let bullet = new createjs.Shape();
    bullet.graphics.beginFill("white").drawCircle(0, 0, 3);
    bullet.x = player.x;
    bullet.y = player.y;

    bulletList.push(bullet);
    stage.addChild(bullet);
}
```

これは、handleClick関数内で書いた処理をそっくりそのまま書けばOKですね。 これでプレイヤーから弾をキー入力で発射できるようになったと思います。

6. Javascriptの基本

この章では、Javascriptという言語の基本的な構文や使い方について学んでいきます。

6.1. 変数について

みなさんは、「変数」という言葉は聞いたことがあるでしょうか？他のプログラミング言語にも変数はあるので知っているかも知れませんが、変数とは、数字や文字などを保存しておく箱のようなものです。

変数を使うためには、ここから変数を使いますよというようにコンピュータに教える必要があります。このことを、「変数を宣言する」といいます。Javascriptでは以下のように変数を宣言できます。

```
let i = 1;
```

こうすることで、iという名前の変数を宣言し、その中身を1という数字にすることができます。

また変数は、その名の通り「変わる数」です。つまり、変数は中身に何かを入れたり、書き換えたりすることができます。変数を書き換えることを、「代入」といいます。

では、変数を宣言してから何かを代入してみましょう。

```
let i = 1;
i = 3;
```

こうすることで、変数iの中身が1から3に変わります。

代入に関して、以下のようなかき方もできます。

```
let i = 1;
i = i + 1;
```

さて、代入した式の右側を見てみると、 `i + 1` のようになっていて、それをiに代入していますね。これは、`i + 1`した結果（ここでは2）をiに代入しています。最終的にiの中身は2になっているはずです。

6.2. 変数の中身がどうなっているか知りたい時

また、Javascriptで変数の中身が何なのかを表示したいときは、 `console.log()` という関数を使いましょう。例えば変数iの中身を知りたい時は、

```
let i = 1;
i = i + 1;
console.log(i);
```

のようにして、出力結果はブラウザのデベロッパーツールのConsoleタブなどで見るといいでしょう。（ブラウザの画面を右クリックして、「検証」を押せばデベロッパーツールが開けます。）

6.3. if文

次はif文について説明します。if文は、ある特定の条件のときだけ処理を実行したいときや、条件を分岐したいときに使います。

```
if(条件文){
    条件が正しい時に実行;
}
else {
    条件が正しくない時;
}
```

例えば、プレイヤーのx座標が入っている変数 `playerX` があったとして、もしプレイヤーのx座標が0から500までなら「画面内です」と表示して、それ以外なら「画面外です」と表示してみます。

```
if(0 <= playerX && playerX <= 500) {
    console.log("画面内です");
}
else {
    console.log("画面外です");
}
```

まずif文の条件文を見てみましょう。 `0 ⇐ playerX` はplayerXが0以上という条件で、 `playerX ⇐ 500` はplayerXが500以下という条件ですね。その２つの条件を `&&` で結んでいます。 `A && B` は、AかつBという意味です。つまり、 `0 ⇐ playerX && playerX ⇐ 500` という条件文は、「playerXが0以上 かつ playerXが500以下」ということです。

（ここでは、⇐ は「以下」という意味で、< だと「未満」となります。）

実際のゲームでは、プレイヤーが画面外に行かないように、「もしプレイヤーの移動先が画面外なら、移動をしない」というふうにすれば、画面外に行かないように実装できます。

6.4. for文

for文は繰り返す処理をするときに使う構文です。かき方は以下のようになります。

```
for(初期値; 実行し続ける条件; 増加（減少）分){
    繰り返す処理;
}
```

for文は初期化する変数が終了条件を満たすまで処理を繰り返します。実際に書いてみると以下のようになります。

```
for(let i = 0; i < 10; i++){
    console.log(i);
}
```

まず変数iに0を入れて、iが10未満なら `console.log(i);` を実行し続けます。そして一回実行するごとに、`i` をします。 `i` は、 `i = i + 1` と同じ意味で、iに1ずつ足していきます。

上のfor文を実行すると以下のような結果が出るはずです。

```
0
1
2
3
4
5
6
7
8
9
```

6.5. 配列

配列とは、変数が複数集まったもののことです。変数を箱に例えると、箱が複数連なっているのが配列です。Javascriptで配列をかく時は、 `let a = [1,2,3,4]` のようになります。

配列がどこで使われるかというと、例えば弾を10個作って、それら全てを動かしたい時、(ここではx座標だけを動かすことにします)

```
let bulletX1 = 10;
let bulletX2 = 5;
let bulletX3 = 13;
let bulletX4 = 19;
let bulletX5 = 2;
let bulletX6 = 20;
let bulletX7 = 14;
let bulletX8 = 0;
let bulletX9 = 4;
let bulletX10 = 21;

bulletX1 = bulletX1 + 3;
bulletX2 = bulletX2 + 3;
// ...途中省略
bulletX10 = bulletX10 + 3;
```

のように、配列を使わないと変数を10個宣言して、１０回移動の処理を書かなければいけなくなります。

では配列を使って表現してみます。

```
let bullets = [10, 5, 13, 19, 2, 20, 14, 0, 4, 21];
for(let i = 0; i < bullets.length; i++) {
    bullets[i] += 3;
}
```

まず、10個の値を配列にいれます。

次に、iが0から bullets.length 未満なら繰り返します。 bullets.length は配列の中身が何個あるのかを表しています（ここでは10）。

次に、bullets[i] += 3; については、i番目のbulletsの値を+3します。例えば、bullets[0]は最初10でそこから+3するので、最終的にbullets[0]は13になります。

こうすることで、配列の中身の値全てを+3することができます。

6.6. 関数について

関数は、同じ処理をまとめて定義して、使い回せるようにしたものです。

Javascriptの関数は以下のように定義します。関数名にはわかりやすい名前をつけるようにしましょう。

```
function 関数名() {
    処理1
   処理2
   ...
}
```

また、関数には引数と返り値を設けることもできます。

```
function 関数名(引数1,引数2,…) {
  処理1
  処理2
  ...
  return 返り値（戻り値）;
}
```

何かのデータを使って関数を実行したい時があります。その時に関数へ渡す値のことを　引数　と言います。

返り値 は、関数を実行して最終的に返す値のことです。

ここでは簡単に、四則演算をする関数を作ってみましょう。

```
<script>
function add(x, y) {
    return x + y;
}

function minus(x, y) {
    return x - y;
}

function multiply(x, y) {
    return x * y;
}

function divide(x, y) {
    return x / y;
}

console.log(add(1, 3));
console.log(minus(5, 3));
console.log(multiply(4, 5));
console.log(divide(9, 3));
</script>
```

それぞれの関数内にx,yという変数がありますが、これらは全部別ものです。その関数内でしか使われません。また、関数の使い方についてですが、`add(1, 3)` のように関数を呼び出します。

6.7. クラスについて

クラスとは、データをひとまとめにしたもの、いわば設計図のようなものです。実例を見せたほうがわかりやすいと思うので、ここではPlayerクラスを作っていきます。

まず、プレイヤーを作るにあたって必要なものは何か考えてみましょう。プレイヤーのx座標、y座標、プレイヤーの画像などが必要ですね。

それらをクラスで定義すると以下のようになります。

```
class Player {
    constructor(x, y, img){
        this.x = x;
        this.y = y;
        this.img = img;
    }
}
```

このクラスはまだ設計図なので、実際にプレイヤーをつくってみましょう。

```
var player = new Player(0,0,"player.png");
```

このようにPlayerクラスから実際のplayerを作ることを、「インスタンスを生成する」と言います。インスタンスを作る際、 `new クラス名(constructorに渡す引数１,引数２,…)` というようにします。

インスタンスを生成したことで、playerの情報がまとめて管理することができます。例えば、playerのx座標とy座標が知りたくなった時、

```
console.log(player.x);
console.log(player.y);
```

のようにすれば、playerのx座標とy座標がわかります。

クラスには、x,yなどのデータ（フィールドと言います）だけではなく、動作も定義できます。例えば、playerが上に動く時は、

```
class Player {
    constructor(x, y, img){
        this.x = x;
        this.y = y;
        this.img = img;
    }

    moveUp(){
        this.y -= 3; // thisをつける
    }
}
```

とクラスを定義して、

```
player.moveUp();
```

とすれば、playerのy座標は-3されます。