1 PeerJS で P2P オンライン対戦ゲームを楽に作ろう

ashiato45

1. これは何？

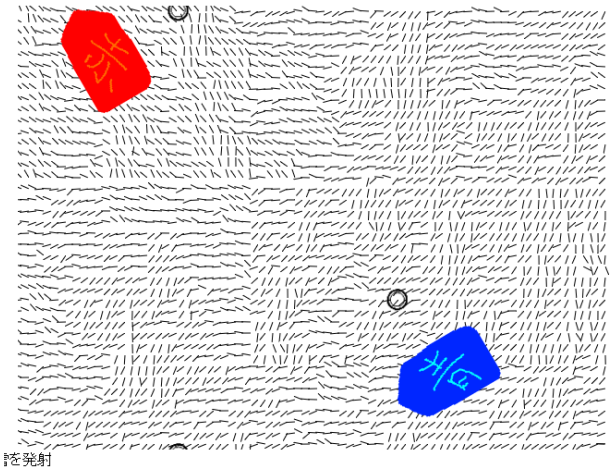
ネット対戦でみんなでできるゲームを作ろう，というのはゲームを作る人は一度はやってみたいと思うものですし，あるいはお気に入りのボードゲームやカードゲームをネット対戦できるようにしたいと考えたことがある人というのもたくさんいると思います．

ところで最近は Javascript がポピュラーになって，JS+Canvas や JS+WebGL などでのゲーム開発というのはライトなゲーム開発者にとってよい選択肢となっています．

さて，Javascript はそもそもブラウザ上で動くものなので，Javascript から通信を扱うのは普通のクライアントアプリから扱うのより楽なのではないか [*1] という気がしますが果たしてその通りで，P2P[*2] を扱う規格が HTML5 にも用意され，それを楽に P2P 扱う PeerJS というライブラリが存在し，さらに P2P をするのに必要なサーバーが無料で使えるという至れり尽くせりな状況になっています．この夏コミに出した「FLOW」はそれを使って数日で作ったもので，その楽さに感動したので今回ここでそれについて報告します．

2. FLOW

夏コミで出したゲームの「FLOW」は，海で海流を読みながら 2 隻の船が爆弾を投げながら戦うゲームで，キーボードの右側と左側で戦う版と PeerJS を用いたネット対戦版があります．今はオンラインで私のサイトで公開しているのでよかったら遊んでみてください．[*3]



[*1] 時前でやろうとすると NAT 越えがどうのとかで繋らなかったりするし，ユーザの側も結構面倒くさい

[*2] Peer to Peer の略．Winny 事件で有名になりましたね．2 台のコンピュータ間で通信するときに，そのコンピュータ同士で通信するような方法のこと．親となるコンピュータを媒介して通信するサーバー・クライアント型とよく対にされる．

[*3] `http:/ashiato45.github.io//`

3. WebRTC と PeerJS

WebRTC というのはブラウザでリアルタイム通信 (Real-Time Connection で RTC) を Javascript で提供しようとする API のことだそうで，すでに最新の Google Chorme，Firefox，Opera では実装されているようです．実際これを使ったブラウザのみによるビデオチャットの Demo などもあり，[*4] なかなかこれはなかなかエキサイティングです．これで提供される機能にリアルタイムのデータ通信も含まれているので，ゲームの対戦データをやりとりすることも可能です．

そして，それをより扱いやすくラップしてくれたのが PeerJS です．PeerJS の Demo としてチャットが紹介されていますが，このソースはページ生成のための HTML とチャットのメインのコードを両方含んでいるにも関わらず 300 行程度になっています．そういうわけで，このチャットを参考にテキストの通信方法をさらって，その上にゲーム情報を流すことにしました．

4. PeerJS の通信の基本

ほとんど Tutorial まんまなのですが，書いてみます．

4.1 PeerServer Cloud service への登録

PeerJS で通信するアプリケーションを作るには PeerServer という接続の確立を媒介するサーバが必要なのですが，これを自分で立てなくても小規模なものなら Peer Server Cloud service[*5] がこれを提供してくれるのでこれを使いました．

サイトにアクセスして「Developer-Free」をクリックしてメールアドレスを打てば終了です．そのアプリケーションのためのキーが発行されるので，これをコード内に埋めこむことになります．

4.2 Peer オブジェクトを作る

```
1 var peer = new Peer({key: 'ここに取得したキーを入れる'});
```

として，通信に使う Peer オブジェクトを作ります．以降 PeerJS の操作はこれで行うことになります．

4.3 こちら側の ID をとる

```
1 peer.on('open', function(id){
2   //idを使った処理
3 })
```

で，通信に使うこちら側の ID を取得します．PeerJS は自分の ID を相手に送り，その相手が互いの ID を使って通信を開始とすることで通信できます．この辺に非対称性があるので，ゲームのようなものを作るのに少し細工をしました．

今回 FLOW では，その ID をテキストボックスに表示しておきました．即ち，“input_mine” という name を持ったテキストボックスを input タグで作り，

[*4] http:/www.webrtc.org/demo/
[*5] http:/peerjs.com/peerserver/

```
var mine;
peer.on('open', function(id){
  mine = id;
  document.mainform.input_mine.value = id;
})
```

となります.

4.4　相手の ID が分かっている状態で相手へ接続する

```
var conn = peer.connect('相手のID');
```

これで ID への接続を作ります．これで相手が接続受けつけをしていれば接続が作られて Connection オブジェクトが conn に入りますが，していない場合はどうなるかよく分からないです．今回は起動時に FLOW が接続受付を開始するようにしてあって，起動してはじめて ID が表示されるようにしてあったので，そういうのを想定していませんでした.

4.5　相手からの接続を受け付ける

```
peer.on('connection', Connectionオブジェクトをうけとるコールバック関数);
```

とすると, 相手からの接続を受けつけて, それがあり次第コールバック関数 [*6] が呼ばれ, その引数は Connection オブジェクトとなっています．つまり，Connection オブジェクトを作る方法は二通りある訳で，接続する側とされる側とで非対称になっています.

4.6　接続が利用可能になったときに処理をする

```
conn.on('open', コールバック関数);
```

こちらから相手に接続したとき，接続が利用可能になったときの処理を書きます.

4.7　相手からのデータを受けとる

```
conn.on('data', 相手からのデータを受けとるコールバック関数);
```

これで相手からのデータを受けとれます．データはすでにデシリアライズされているので，普通の JS の値として利用することができます．例えばこれで相手の位置が飛んできたらそれを反映すればいいわけです.

4.8　相手にデータを送る

```
conn.send(データ);
```

データを送ります．データは普通の JS の値を入れればよく，オブジェクトリテラルだって使えます．シリアライズは勝手に PeerJS がやってくれるので，どう送られているかを気にする必要はありません．ただ，オブジェクトが画像など不要なものを含んでいるならばそれを外したオブジェクトを作って送るべきでしょう.

[*6] 何か事があったときに，場合によってはそのイベントの情報を引数に渡されて呼ばれる関数

5. 今回どのように作ったか

まず，Connection オブジェクトが利用可能になったときのために connect 関数を用意しておきます．これは peer.on('connection', ...) で相手から接続が飛んできたときや，conn.on('open', ...) で接続が利用可能になったときに，conn.on('data', ...) して相手からの情報を受けとれるようにします．かなり行儀は悪いのですが，飛んできたデータはそのままグローバル変数に流してしまいました．また，受けとった Connection オブジェクトはグローバル変数 conn に入れ，送信などに使えるようにしてあります．

```
function connect(c){
   conn = c;
   conn.on("data", function(data){
  receipt = data;
  if(state_game == STATES_GAME.COUNTING){
  if(receipt.type == "map"){
      console.log("map get");
      if(!is_owner()){
    perlinmap[receipt.x][receipt.y] = receipt.t;
    console.log(receipt.x, receipt.y, receipt.t);
      }
  }
    }
    });
}
```

ただ，map に関しては即座に処理するようになっています (なぜだろう)

peer.on('connection',...) に関しては起動時にすぐ行なって，相手からの接続はいつでも受けつけるようにしています．スペースキーで接続を開始したおきに相手への接続を行ないます．ですから，先に接続要求をしたほうは相手もスペースを押して接続開始するまで待機するわけですが，いざ相手が押したら接続要求の受けとりでまた Connection を作りなおすことになります．つまり，先に接続要求をして，相手から接続要求を後で受ける側はおそらく peer.on で作られた Connection オブジェクトを，後に接続要求をしたほうは，おそらく peer.connect で作った Connection オブジェクトを conn として使うことになります．無駄っぽいですが，とりあえずこれで動いてます．

このゲームでは海流が両プレイヤーで共通していなければいけないので，それをカウントダウン中に送っています．一方をもう一方に送ればいいのでどちらかが送ればいいのですが，どちらが送るかはプレイヤー ID の辞書的な意味で若くないほうを owner と呼び，owner が送ることになっています．マップのデータは二次元配列をそのまま peer.send で送って，先の connect で反映させています．地図送信は

```
function send_map(){
   for(xi=0; xi < MAP_WIDTH; xi++){
  for(eta=0; eta < MAP_HEIGHT; eta++){
      send({type:"map",
      x:xi, y:eta,
      t:perlinmap[xi][eta]});
  }
    }
}
```

こんな感じです．

プレイ中は敵の位置，爆弾の位置，あるいは試合が終了したか否かを送ることになります．今試合中であれば，メインループの終わりで自分の状況を相手に知らせます．

```
function send_state(){
   bi = new Array();
   for(l=0; l < bombs.length; l++){
  bi.push(bombs[l].sprite.pos);
   }

   send({type:"gameinfo",
        ship:{pos:ship_red.sprite.pos,
```

```
          angle:ship_red.sprite.angle,
          cursorpos:ship_red.cursor.pos,
          waiting:ship_red.waiting},
          bombinfo: bi});
}
```

となっています．送るデータの send にデータの種類を持たせています (それは先の map にせよ同じですね．) 爆弾のオブジェクトは画像データを持ってしまっているので，send_state の冒頭で位置情報だけいしています．このゲームでは互いに爆弾を投げることができますが，相手に送るのは自分が投げた爆弾だけです．

相手から飛んできたデータは，メインループの終わりに，

```
 function apply_receipt(){
    if(receipt == undefined || conn == undefined){
  return;
    }
    if(state_game == STATES_GAME.HANDSHAKING){
  console.log(receipt);
  if(receipt.type == "handshake"){
      console.log("handshake end");
      state_game = STATES_GAME.COUNTING;
  }
    }

    if(state_game == STATES_GAME.NO){
  if(receipt.type == "gameinfo"){
      ship_blue.sprite.pos = receipt.ship.pos;
      ship_blue.sprite.angle = receipt.ship.angle;
      ship_blue.cursor.pos = receipt.ship.cursorpos;
      ship_blue.waiting = receipt.ship.waiting;
      foebombs = new Array();
      for(l=0; l < receipt.bombinfo.length; l++){
    foebombs.push(new Bomb(receipt.bombinfo[l]));
    console.log(receipt.bombinfo[l]);
      }
  }
  if(receipt.type == "winner"){
      if(receipt.winner == mine){
    state_game = STATES_GAME.WIN_RED;
      }else{
    state_game = STATES_GAME.WIN_BLUE;
      }
  }
    }
}
```

で処理しています．これはゲームの今のフェーズと相手から飛んできたデータの種類で場合分けして処理しています．type が winner というのは，戦闘が決着したということです．

6. おわりに

なんとなく概略を説明してみました．データの通信の基本は大したことはないのですが，互いに準備できたことをどう処理するかがポイントだったような気がして，それについて結構汚らしくなってしまったように思います．

データの送信は普通にオブジェクトが送れるので何も困らないでしょう．普通のどの言語よりもゲームの本質に注目してネット対戦が作れる素晴らしいライブラリだと私は思いました．