WebSocket を用いた Universe フレームワークのブラウザ対応

上田結実 (指導教員：浅井 健一)

1 はじめに

プログラミング教育の現場では、プログラミングが嫌いになる学生等が見られる。そこで、楽しくプログラミングが学習できる題材としてゲームプログラミングが着目されている。

Universe フレームワーク [4, 5, 7] は、初学者向けにゲームプログラミングをサポートするフレームワークである。この環境は Unix モジュールの通信を利用した対戦ゲームの作成が可能だが、初学者にとってより身近なブラウザ上で動くゲームを作成できると良いと考えた。また、プログラミング初学者向けのツールの 1 つにビジュアルプログラミング言語がある。これは Web ブラウザ上で動くものが多く、Universe フレームワークのブラウザ対応をすれば、ビジュアルプログラミングの環境での通信ゲーム作成に活かせると考えた。

このような理由から、本研究は、WebSocket を利用した双方向通信を実装して Universe フレームワークをブラウザに対応させ、通信ゲームを可能にすることを目的とする。

2 クライアント・サーバ

複数人で対戦するためには、サーバとクライアントを作ることが必要である。Universe フレームワークでは、プレイヤーのプログラムはすべてクライアントである。クライアントとは別にサーバを 1 つ作り、クライアントとサーバが情報を送受信することで通信を行う。以後、このやり取りする情報をメッセージと呼ぶ。

3 Universe フレームワーク

Universe フレームワークには、簡単にゲームを作るための基本的な関数が一通り定義されている。そのため、ゲームの作成者はゲームの内容を作ることだけに専念できる。

3.1 world

Universe フレームワークで定義されている world の概要を説明する。world は「各クライアントの現在のゲームの状況」のことである。

world 内で使用する、ゲームを開始する関数を big-bang 関数という。ここでは第一引数に world の初期値を受け取り、第二引数以降にゲームの動作に必要な関数を受け取る。big-bang 関数の中で使用している draw 関数は、world を受け取ったらそれに基づいた画像を返す。big-bang 関数の動き方は以下である。

1. 登録された draw 関数を使って、初期のゲーム画面を描写する
2. 一定時間ごとに world を変化させる on_tick 関数を呼んで、world を更新する
3. キーイベントやマウスイベントが起きたときも、対応する on_key 関数、on_mouse 関数を呼んで world を更新する
4. world が更新されるたびに、新しい world を元に draw 関数を使ってゲーム画面を描写する

また通信のためには、on_receive 関数を big-bang に登録する必要がある。on_receive はサーバから新しいメッセージが送られてくる度に呼ばれる関数である。ユーザは on_receive 関数を利用して、クライアントの world を最新の状態にすることができる。

3.2 universe

次に universe という関数について説明する。universe はサーバを作る時の関数で、big-bang と同じように関数を登録することでサーバとしての役割を果たし、通信を可能にする。サーバの役割は各クライアントへ最新のゲームの状況を配信することである。ここでは画面の描写はせず、通信するクライアント間のデータのやり取りを中継する役割を果たす。そのため全クライアントの状況を保持する必要があり、universe では各クライアントごとに、ゲームの状況と相手の状況をリストにして扱う。

通信ゲームでは on_tick 関数、on_new 関数、on_msg 関数を universe に登録する。on_tick 関数は定めた時間ごとに呼ばれ、その度に変化したゲーム状況を返す。on_new 関数は新たなクライアントがゲームに参加した時、on_msg 関数はあるクライアントからメッセージを受信した時に呼び出され、各クライアントのゲーム状況を更新する。

4 通信方法

既存の Universe フレームワークでは、Unix モジュールを使ったプロセス間通信によって通信ゲームを行っていた。

本研究では Universe フレームワークをブラウザに乗せたいと考えたがこのままでは通信ができないので、socket.io を使用した WebSocket 通信を行った。

この実装のために用いた環境の説明をする。

Node.js サーバサイドで JavaScript を実行可能にする [2]

socket.io WebSocket などの非同期双方向通信を Node.js から利用できるようにしたモジュール [3]

Express web アプリ開発に有用なフレームワーク [1]

これらの環境を利用してサーバとクライアントが通信を始める手順を考える。

1. ポート番号を指定して HTTP と socket.io サーバの待ち受け (サーバ側)

   サーバ側では socket.io モジュールを読み込み、createServer によってサーバを生成する。また、listen 関数を実行することによってソケットの紐付けを行い、クライアントからの接続の待ち受けを行う。

2. コネクションを作成する (クライアント側)

クライアント側では、socket.io のクライアント用ライブラリを読み込み、サーバが立てた URL にアクセスする。

3. コネクション時にイベントを送信する (サーバ側)

   サーバ側ではクライアントの接続を受け付けると、コネクションイベントが発生する。ここで最初のゲームの状況をクライアントに送信する。

4. ゲーム画面を描写する (クライアント側)

   クライアント側ではサーバから送られたゲームの状況を画面に描写する。

socket.io の on 関数はデータの受信を処理するものであり、emit 関数はデータの送信を処理するものである。この on 関数、emit 関数を使用してメッセージの送受信を行なった。

socket.io でアクセスしたクライアントには必ず一意の ID が割り当てられ、この ID を利用することで、特定のクライアントに対してのみのデータをサーバから送信できる。

また、Unix モジュールの使用時と同様に、サーバ・クライアント間のやり取りの際、メッセージの型を string 型にする必要があり、そのままの状態では送れないことやデータが壊れてしまうことがある。そのため、メッセージを marshaling して string 型に変換して文字列にし、更に読める文字列にするために base64 で encode した上で、送信している。同様に、受信時は base64 で decode して unmarshaling している。なお、ユーザはゲーム製作時にメッセージを送る側と受け取る側で同じ型を扱うように設計する必要がある。

5 通信ゲームの実装

実際に socket.io を使用した通信ゲームを作る。今回は、Unix モジュールで作られたボールゲームを参考に、複数人で通信するボールゲームを作った。

ゲーム画面ではクライアント 1 つにつきボールが 3 個動き、そのボールはクリックすると小さくなる。また、ボールの色はクライアントによって異なる。ここで、相手の 3 個のボール全てを決められた大きさよりも小さくすると、その相手は Game over となる。

Unix モジュールの使用時と同様、プログラムのファイルはライブラリ部分とゲーム構成部分で分けている。ライブラリはどんな通信ゲームを作る時にも共通するプログラムの部分であり、この部分を切り離しておくことによって、ゲームの製作時にゲームの内容のみに専念してプログラムを考えることができる。

本研究のブラウザ対応は、Unix モジュールの Universe フレームワークで作成されたボールゲームのファイルの、ライブラリ部分のみの変更で実行することができた。そのため、ゲーム製作者はその作成時には大きな変更がないまま、ブラウザで通信ゲームを行うことができる。

ライブラリ部分の変更では、socket.io が Node.js から利用できるモジュールであるため、一旦 JavaScript で書いたコードを OCaml に変換していく。socket.io のモジュールは OCaml では未定義であるため、socket や io を関数として使用する前に、型を定義する所から取り入れなければならず、その際には Js.Unsafe.eval_string

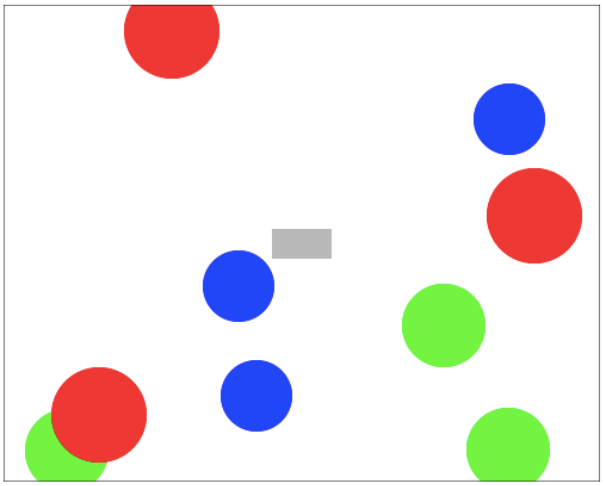

図 1: 3 人でのボールゲームの様子

という、文字列をそのまま JavaScript として評価する関数など、Unsafe モジュールを使用し、動作を確認しながら段階的に OCaml に変更していく。

6 現状と実使用に向けて

本研究では WebSocket 通信を利用し、OCaml の Universe フレームワークで作成した通信ゲームの実行・動作確認が出来た。

しかし、ボールゲームの作成と動作確認しかできていないので、実使用に向け、更に複数の様々なゲームを作成して動作確認を行いたい。また、よりわかりやすくプレイヤーの識別をすることや、ゲームスタート時やゲームオーバー時の動きを工夫できるようにしたい。

7 まとめ

本研究では、WebSocket 通信を使用した Universe フレームワークのブラウザ対応を行い、実際に作ったゲームの動作確認を行なった。今後は更に様々な複数の通信ゲームを作成して動作確認を行っていく。そして将来的には OCaml のビジュアルプログラミングの環境 [6] で使用できるようにし、ユーザ実験を行いながら扱いやすくしていきたい。

参考文献

[1] Express. `https://expressjs.com/`.

[2] Node.js. `https://nodejs.org/ja/`.

[3] socket.io. `https://socket.io/`.

[4] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. A Functional I/O System, or, Fun for Freshman Kids. In *ACM International Conference on Functional Programming (ICFP 2009)*, pp. 47–58, 2009.

[5] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. How to Design Programs, Second Edition. 2014.

[6] 松本晴香, 浅井健一. Blockly をベースにした OCaml ビジュアルプログラミングエディタ. 第 2 1 回プログラミングおよびプログラミング言語ワークショップ論文集, 11 pages, March 2019.

[7] 上原千裕, 浅井健一. LablGtk2 を用いた universe teachpack の実装. 日本ソフトウェア科学会第 31 回大会 (2014 年度) 講演論文集, 15 pages, September 2014.