WebSocket 通信を用いた Universe フレームワークの拡張

理学専攻　情報科学コース　2040633　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。

本研究では初学者向けの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、Universe フレームワーク [4, 5, 7] を使用した。既存の環境は 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、通信部分を変更してブラウザ対応をすると、初学者にとってさらに身近に感じるものとなるだけでなく、遠方の様々な機器との通信が可能になるなど利点が多いと考えた。

そのため、本研究は、WebSocket 通信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のプログラムを作り、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 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 関数といい、その中で使用している関数は次のような動きをする。・初期のゲーム画面を描画する・一定時間ごとに world を更新する・キーイベントやマウスイベントを受け取って world を更新する・world の更新のたびにゲーム画面を描画する

また通信のためにはサーバからのメッセージを受け取る関数を big-bang 関数に登録し、クライアントの world を最新の状態にする必要がある。

3.2　universe

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。

通信のために登録する関数は以下のような動きをする。・定めた時間ごとに変化したゲーム状況を返す・新たなクライアントがゲームに参加したときにゲームの状況を更新する・あるクライアントからメッセージを受信したときにゲームの状況を更新する

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 socket.io を使用した WebSocket 通信に通信部分のプログラムを変換した。

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)

   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受を行う。

2. コネクションを作成する (クライアント:ゲーム作成者)

   ゲーム作成者は、ゲームプログラムを保持した状態でサーバが立てた URL にアクセスする。そこで socket.io のクライアント用ライブラリの読込を行い、universe 関数の呼出によりゲームサーバが起動される。

3. コネクション時にイベントを送信する (サーバ)

   サーバ側ではゲーム作成者からの接続を受け付けると、任意の通路名を発行し URL 内にパラメータとして追加した上でゲーム作成者に通知する。また、ゲームプログラムを受け取りプレイヤーからの接続の待受を行う。

4. コネクションを作成する (クライアント:ゲームプレイヤー)

   ゲームプレイヤーは、ゲーム作成者から受け取った URL にアクセスする。そこで socket.io のクライアント用ライブラリの読込を行い、パラメータから通路名を取得して big-bang 関数の呼出が行われる。

5. コネクション時にイベントを送信する (サーバ)

   サーバ側ではクライアントの接続を受け付けると、ゲームサーバへその情報が渡り、ゲームサーバから最初のゲームの状況を配信する。

6. ゲーム画面を描写する (クライアント)

   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。

このように通信を行うことで、サーバ側にあらかじめゲームのプログラムを置かなくて良い形になっており、サーバ側のあらゆる場所へアクセスできない仕組みやそのファイル自体をサーバ内で実行しない仕組みによって、サーバ側のセキュリティ面での安全性がある程度保つことができると考える。

また、ゲーム作成者が接続を行うときにユニークなチャネル名を生成しているため、同時に複数のゲームサーバを立てることができ、それぞれ独立にゲームをすることが可能である。

socket.io では on 関数がデータの受信を処理するものであり、emit 関数が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である。

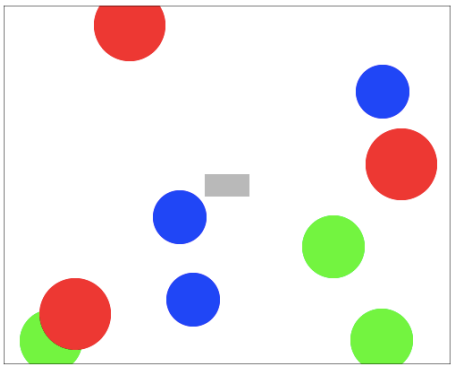

図 1: 3 人でのボールゲームの様子

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。

実使用に向け、数人に実際に使ってもらってヒアリングを行い、インターフェースなどを整えたいと考える。また、通信時の接続方法について、現状ではプログラムファイルの送受信が必要だが、この部分でより安全性を高めることができる方法を模索する。

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。今後はさらに安全性を高める工夫を模索する。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.