UniverseJs ライブラリを用いたゲームプログラミングのデバッグ環境の改善

茅根 珠来 (指導教員：浅井 健一)

1 はじめに

UniverseJs ライブラリ [1] を利用したゲームプログラミングでは、「世界」と呼ばれるゲームの状態を変化させていくことでゲームを進めていく。しかし、世界の情報は私たちには見えていないため、世界がどのように変化しているのかを知ることは難しく、このことはデバッグのやりづらさに影響している。そのため、世界の情報の可視化をはじめとした実装をすることでデバッグ環境を改善したいと考えた。そこで、ppx_deriving[1] を使用し、単独ゲームプログラミングでは世界の情報を、対戦ゲームプログラミング [2] では世界と相互メッセージの表示を行えるようにした。加えて、単独ゲームプログラミングではゲームの一時停止などの機能を追加した。

2 UniverseJs ライブラリを用いたゲームプログラミング方法

UniverseJs ライブラリは、ユーザが定義した世界やゲーム画面を描く draw 関数、ゲーム画面でのマウスの動きやクリックに対する世界の変化を決める on_mouse 関数、ゲーム画面でのキー入力に対する世界の変化を決める on_key 関数、時間経過による世界の変化を決める on_tick 関数といった関数を big_bang 関数に登録することで、実際にゲームとして動かすことを可能にするライブラリである。

単独ゲームプログラミングでは、ユーザがゲームの状態を定義する世界の情報に加えて、draw 関数、on_mouse 関数、on_key 関数、on_tick 関数といった関数の中から作ろうとするゲームに必要な関数を定義する。それらを big_bang 関数に登録することで単独ゲームが作られる。

対戦ゲームプログラミングでは、クライアントとサーバがメッセージでやり取りすることによってゲームが進行していく。そのため、クライアントの世界についてのプログラムとサーバの世界についてのプログラムをユーザが定義する必要がある。クライアントのプログラムは、クライアントの世界の情報、draw 関数、on_mouse 関数、on_key 関数、on_tick 関数以外に、メッセージを受け取ったことによる世界の変化を決める on_receive 関数を定義する。サーバの世界についてのプログラムは、サーバの世界の情報、on_tick 関数、受け取ったメッセージの処理した後の状態を返す on_msg 関数といった関数を定義する。これらのプログラムをそれぞれの big_bang 関数を登録することで対戦ゲームが作られる。

3 ppx_deriving

本研究では、ユーザが定義した世界の情報をコンソール上に表示するために ppx_deriving という OCaml の ppx の機能を使用している。ppx とは OCaml のプリプロセッサ方式の一つで、OCaml の構文解析木 (AST) を受け取って AST を返すプログラムである。ユーザが定義した世界の情報である world_t を表示するには、ユーザが定義した world_t 型の値を文字列にしなくてはならない。しかし、そのような関数をあらかじめ UniberseJs ライブラリの中に作っておくことはできないため、型定義に [@@deriving show] をつけて、world_t がユーザによって定義された時点で定義を生成させている。例として以下のように型定義の後ろに [@@deriving show] を付け加えると、show_world_t という world_t 型の情報を string 型にするための関数ができ、これを world_t 型のレコードに適用することで文字列として表示することができる。

```
type world_t = {
  x : int;
  y : int;
}
[@@deriving show]
```

4 デバッグ環境の改善目的と内容

4.1 一時停止、1 ステップ実行、連続実行

一時停止機能の実装以前は、1tick ごとに世界の情報は変化していくため、ゲーム画面のある特定の状況を再現するためには何度も実行してその状態になるまでゲーム画面を進行させる必要があった。また、1 ステップ実行機能の実装以前は、1tick ごとの細かい動作確認をすることができず、バグの原因を探るためにはゲーム画面の視覚情報と書いたプログラムからの両面から見た推測が必要であった。このように、視覚情報からデバッグを行う場面は非常に多いため、それを補助することを目的として、一時停止、1 ステップ実行、連続実行をキー操作で行える機能を実装した。一時停止中は tick イベントが起こらないため、多くのイベントが 1tick の間に起こることになる。特に、マウスイベントはマウスを動かしただけで発生するため、1tick の間に非現実的なイベントが発生してしまう。これを避けるため、連続実行時以外での on_mouse 関数の実行は行わないこととした。

4.2 世界の情報と押されたキーの表示

視覚情報からのデバッグでは細かな不具合が残ってしまう場合や、バグがあった時に具体的にプログラムの書き直すべき部分が浮かばない場合がある。また、押されたキーに対して世界の情報がどのように更新されたかが明確に分からない場合がある。そこで、世界の情報の変化とゲーム画面の両方を確認しながらのデバッグを可能にすることで、より短い時間でデバッグができるようにすることを目的として、世界の更新された情報と押したキーがコンソールに表示される機能を実装した。このとき、世界の情報が更新されない場合にはコンソールに表示を行わないことで、世界の情報の変化を見やすくしている。

4.3 クライアントとサーバの世界の情報と相互メッセージの表示

[1] https://github.com/ocaml-ppx/ppx_deriving

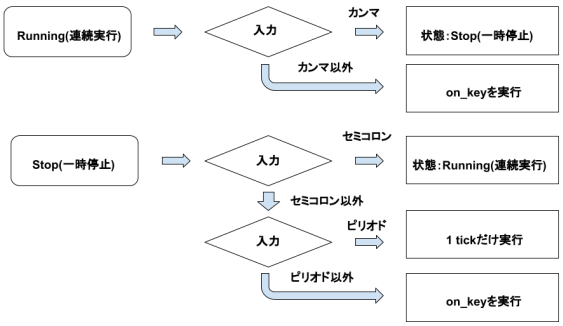

図 1: 各状態からの遷移

対戦ゲームプログラミングでは、各クライアントとサーバ間でのメッセージのやり取りによって世界を更新していく。クライアントからサーバへのメッセージ内容とそれによるサーバの世界の情報の変化、サーバからクライアントへのメッセージ内容とそれによるクライアントの世界の情報の変化をそれぞれ可視化することで、自分のプログラムに対する理解を深めるとともに、バグの原因がクライアントやサーバのどこにあるのかを分かりやすくすることを目的としてクライアントとサーバの世界と相互メッセージの表示を行う機能を実装した。

5 具体的な手法と変更点

5.1 一時停止、1 ステップ実行、連続実行

実行の状態の管理をするため、状態を表す値として連続実行を表す Running と一時停止を表す Stop の導入を行い、各状態での on_key 関数や on_mouse 関数の処理を行うように変更した (図 1)。

図 1 から分かるように、on_key 関数の処理について、各状態で特殊文字として扱う文字入力された場合以外には、ユーザが定義した on_key 関数が実行されるようになっている。特殊文字が入力された場合の状態の変化とゲームプログラムの進行は、以下のようになっている。状態が Running である場合においては “ ,(カンマ) ” を特殊文字として扱い、“ , ” が入力された場合には状態を Stop へ変更し、プログラムを一時停止させる。一方で、状態が Stop である場合には、“ ;(セミコロン) ”、“ .(ピリオド) ” の二つを特殊文字文字として扱い、“ ; ” が入力された場合は状態を Running へ変更し、プログラムを連続実行させ、“ . ” が入力された場合は on_tick 関数を一度だけ実行することで、1 ステップ実行を行うようにした。また、on_mouse 関数は状態が Running であるときのみに実行が行われるようになっている。

5.2 世界の情報と入力キーの表示

ユーザが定義した世界の型の後ろに [@@deriving show] を追加し、世界の型の情報を string 型にするための関数をつくり、これを big_bang 関数へ付け加え、それを処理するプログラムへ引き渡すことで、ユーザが定義した世界の情報の表示をすることができるようになる。今回 UniverseJs ライブラリを書き換えたことにより、ユーザ側は、定義した世界の型の後ろに [@@deriving show] を追加するだけで世界の情報の表示が可能になった。

また、世界の情報に変化があった場合にのみ世界の情報の表示を行うため、1 つ前の世界の情報と現在の世界の情報を保持し、世界の情報が更新されるとともに, 保持している 2 つの情報も更新していく。1 回更新していくごとに、1 つ前の世界の情報と現在の世界の情報を比較し、世界の情報に変化があった場合はユーザが定義した型の通りの現在の世界の情報がコンソールに表示されるように変更した。しかし、このままでは最初の世界の情報は表示されなくなってしまうため、最初の世界を表示する命令を追加し、これを一度だけ呼び出すことで、1 つ前の世界の情報が存在しない、世界の最初の情報の表示を可能にした。

また、入力キーは on_key 関数に入力キーを出力する命令を追加することで表示されるようにした。

5.3 クライアントとサーバの世界の情報と相互メッセージの表示

ユーザが定義したクライアントの世界の型定義、サーバの世界の型定義、サーバへのメッセージの型定義とクライアントへのメッセージの型定義の後ろに [@@deriving show] を追加し、世界とメッセージの型の情報を string 型にするための関数をつくり、これを big_bang 関数へ付け加える。それぞれの big_bang 関数を処理するプログラムへ引き渡し、サーバへのメッセージとクライアントへのメッセージ、ユーザが定義した各世界の情報の表示をすることができるようになる。今回 UniverseJs ライブラリを書き換えたことにより、ユーザ側は、定義した世界の型とメッセージの型の後ろに [@@deriving show] を追加するだけで世界の情報と相互メッセージの表示が可能になった。

単独ゲームプログラミングの世界の情報の表示時と同様にして、世界の情報に変化があった時のみコンソールに表示されるようにした。

6 まとめと今後の展望

本研究では、実際にゲームプログラミングを通して感じたデバッグの効率の悪さを改善するため、単独ゲームプログラミングでは一時停止、1 ステップ実行、連続実行といった機能や世界などの表示機能、対戦ゲームプログラミングではクライアントとサーバの各世界の情報と相互メッセージの表示機能を実装した。実際に UniverseJs ライブラリを用いて作成した単独ゲームや対戦ゲームに対して実装した機能を試したところ、想定通りに動作することが確認できた。

以上の実装が実際のデバッグにどのような影響を与えることができるのかについての実証実験を現在は行えていない。そのため、今後は実証実験を行い、その結果をもとに UniverseJs ライブラリを用いたゲームプログラミングのデバッグ環境をより良いものにしたい。

参考文献

[1] C. Uehara and K. Asai. Cross validation of the universe teachpack of Racket in OCaml. *The 4th International Workshop on Trends in Functional Programming in Education (TFPIE 2015)*, pp. 1–14, June 2015.

[2] 上田結実. WebSocket 通信を用いた Universe フレームワークの拡張. お茶の水女子大学修士論文, 2022.