Julien Moumné Software Engineer, Paris, France

Introducing Rx Training Games

An introduction to Rx Training Games, a coding playground to learn and practice Reactive Extensions

22 Nov 2015

Rx Training Games is a coding playground that can be used to learn and practice Reactive Extensions coding grid-based games.

See it in action here.

The project ambitions are to :

- offer a way to learn new technology while having fun - with as little hardware and software requirements as possible
- demonstrate how leveraging current technology makes it easy to build such training platforms

The idea came out of several sources of inspiration :

- A Playful Introduction to Rx by Erik Meijer
- Online coding playgrounds such as JS Bin and CodingGame

Why Reactive Extensions

If you have developed software that deals with asynchronous data streams such as user inputs, web service requests and I/O, you have probably faced issues related to :

- thread safety
- synchronization
- exception management

Reactive Extensions offer a way of looking at asynchronous data streams as objects that can be queried and composed while abstracting low-level constructs such as threads.

They can be useful in a wide range of applications :

- Composing microservices : NetflixOSS Meetup - Season 2 Episode 2 - Reactive / Async
- Developing games : A Playful Introduction to Rx by Erik Meijer
- GUI components such as an Autocomplete feature : RxJS-DOM Autocomplete Tutorial

For a complete introduction to Rx I suggest The introduction to Reactive Programming you've been missing.

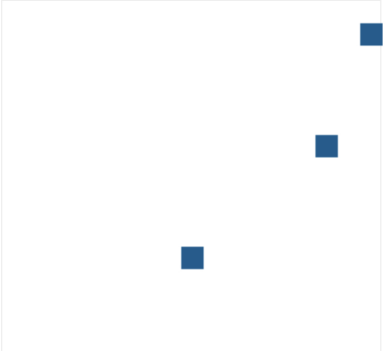
Rx Training Games follows the footsteps of A Playful Introduction to Rx by Erik Meijer and takes a close look at how Reactive Extensions can be used to build games.

Meteorites

Games published on *Rx Training Games* can be embedded in an iframe such as the one bellow.

Press start, then use the Left and Right arrows of your keyboard to aim and the Space bar to fire.

Feel free to experiment with the code by editing it.



```
    .subscribe(meteoriteLayer.fill);

// meteorite updates
Rx.Observable.interval(fallPulse)
    .flatMap(() =>
Rx.Observable.from(meteoriteLayer.getActiveSquares()))
    .do(meteoriteLayer.clear)
    .map(api.directions.Down)
    .do(meteoriteLayer.fill)
    .where(api.isOffLimits)
    .subscribe(api.gameOver);
```

Engine Off ▶ Start or, hit 'Ctrl + Return'.

```
// bullet spawns
api.keyboard.where(keyCode => keyCode == 32)
    .map(() => spaceshipLayer.getActiveSquares()[0])
    .subscribe(bulletLayer.fill);

// bullet updates
Rx.Observable.interval(firingPulse)
    .flatMap(() =>
Rx.Observable.from(bulletLayer.getActiveSquares()))
    .do(bulletLayer.clear)
    .map(api.directions.Up)
    .where(api.isWithinLimits)
    .subscribe(bulletLayer.fill);

// spaceship moves
api.keyboard.where(keyCode => _.contains([37, 39], keyCode))
```

Built using Rx Training Games

A few snippets have also been published to take a closer look at specific parts of the game.

Find below one approach of generating the falling meteorites :

Engine Off ▶ Start or, hit 'Ctrl + Return'.

```
var spawnPulse = 200;
var fallPulse = 100;

api.initGrid({squareSize: 6});

var layer = api.addLayer({color: '#337ab7'});

// droplets spawns
Rx.Observable.interval(spawnPulse)
    .map(() => ({x: api.randomCoord(), y: 0}))
    .subscribe(layer.fill);

// droplets updates
Rx.Observable.interval(fallPulse)
    .flatMap(() => Rx.Observable.from(layer.getActiveSquares()))
    .do(layer.clear)
    .map(api.directions.Down)
    .where(api.isWithinLimits)
    .subscribe(layer.fill);
```

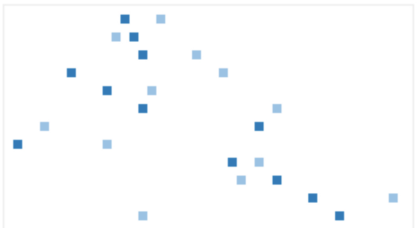
Built using Rx Training Games

More games and snippets are available in the app.

API

Rx Training Games provides a basic API for building games.

Squares of different colors are switched on and off in a graphical square grid :



The grid is first instantiated with the size of the activable squares : `api.initGrid({squareSize: 15});`

Multiple layers with different colors can be added with `var layer = api.addLayer({color: '#337ab7'});`

A layer exposes the following methods :

- `layer.fill({x: 42, y: 42})` : activate a square
- `layer.clear({x: 42, y: 42})` : clear a square
- `layer.getActiveSquares()` : retrieve a list of active squares

The layer is also implemented as an Observable Collection and provides two Observables :

```
// square activations
layer.activations.subscribe(coord =>
    console.log(coord.x + ',' + coord.y + ' has been activated')
);

// square removals
layer.removals.subscribe(coord =>
    console.log(coord.x + ',' + coord.y + ' has been cleared')
);
```

Keyboard events are accessible using an Observable :

```
// keyboard events
api.keyboard.subscribe(keyCode =>
    console.log(keyCode)
);
```

Please consult the complete documentation for more details.

Contributing

Do you think you can invent games with these simple elements? See how to contribute.

Please comment on this page if you think the project or the blog can be improved.