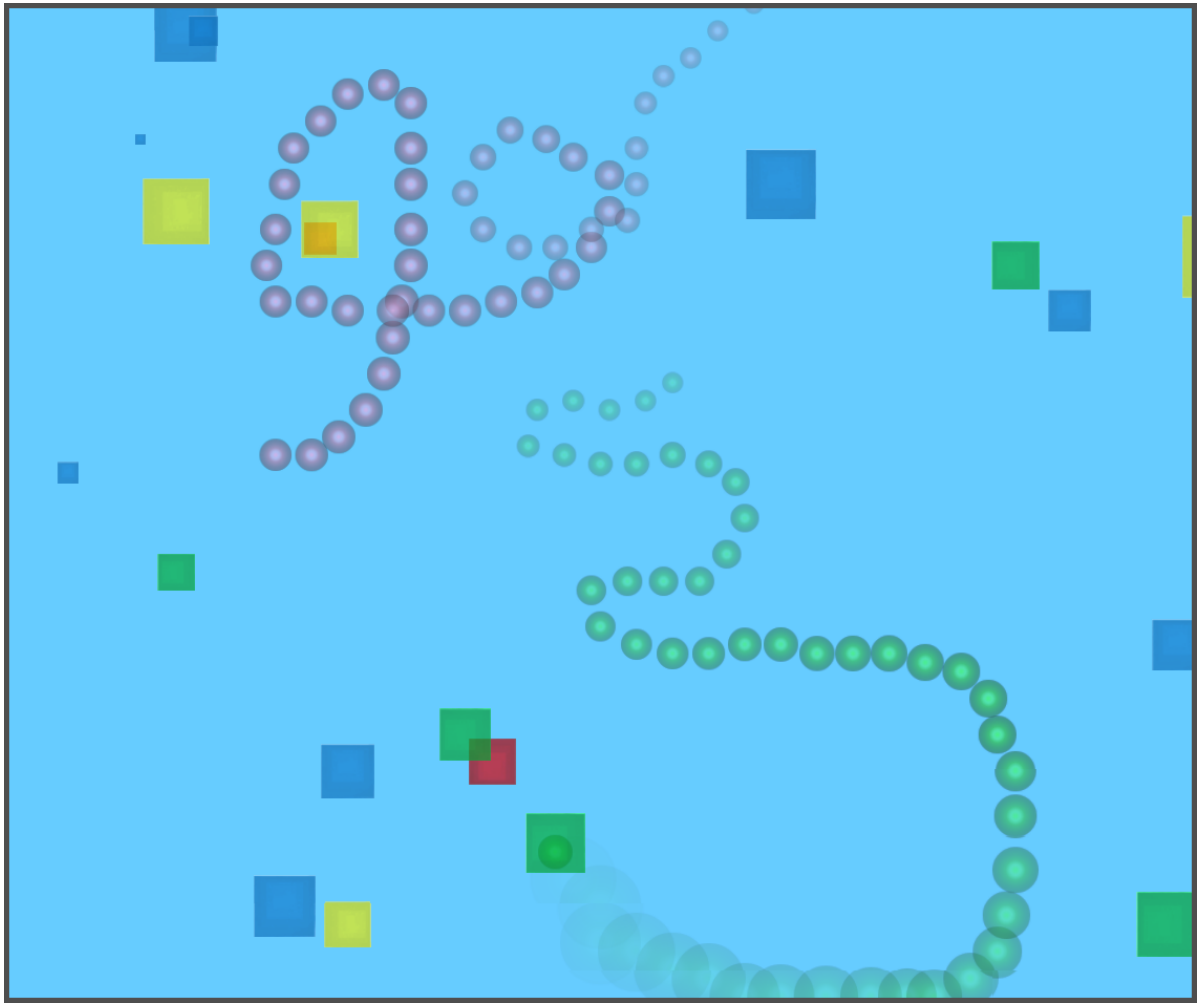
Browser-based MMO Game with Akka and Play!

Piotr Kukielka

March 20, 2013



Source code: https://github.com/pkukielka/QuickPath

Why Play?

- Nice integration with Akka

Why Play?

- Nice integration with Akka
- Out of the box support for WebSockets and streams

Why Play?

- Nice integration with Akka
- Out of the box support for WebSockets and streams
- Embedded support for various assets (i.e. CoffeeScript, LESS)

Why Play?

- Nice integration with Akka
- Out of the box support for WebSockets and streams
- Embedded support for various assets (i.e. CoffeeScript, LESS)
- Simplified change-compile-run cycle

Why Play?

- Nice integration with Akka
- Out of the box support for WebSockets and streams
- Embedded support for various assets (i.e. CoffeeScript, LESS)
- Simplified change-compile-run cycle
- Easy to work with Json

Build system

- Play console, sbt

Project structure - standard application layout

Build system

- Play console, sbt

Project structure - standard application layout

- MVC architecture (app/ directory)

Build system

- Play console, sbt

Project structure - standard application layout

- MVC architecture (app/ directory)
- Resources (public/ directory)

Build system

- Play console, sbt

Project structure - standard application layout

- MVC architecture (app/ directory)
- Resources (public/ directory)
- Configuration (conf/ directory)

Build system

- Play console, sbt

Project structure - standard application layout

- MVC architecture (app/ directory)
- Resources (public/ directory)
- Configuration (conf/ directory)
- Unmanaged library dependencies (lib/ directory)

Build system

- Play console, sbt

Project structure - standard application layout

- MVC architecture (app/ directory)
- Resources (public/ directory)
- Configuration (conf/ directory)
- Unmanaged library dependencies (lib/ directory)
- Build definitions (project/ directory)

Build system

- Play console, sbt

Project structure - standard application layout

- MVC architecture (app/ directory)
- Resources (public/ directory)
- Configuration (conf/ directory)
- Unmanaged library dependencies (lib/ directory)
- Build definitions (project/ directory)
- Output files (target/ directory)

Build system

- Play console, sbt

Project structure - standard application layout

- MVC architecture (app/ directory)
- Resources (public/ directory)
- Configuration (conf/ directory)
- Unmanaged library dependencies (lib/ directory)
- Build definitions (project/ directory)
- Output files (target/ directory)
- Tests (test/ directory)

Build system

- Play console, sbt

Project structure - standard application layout

- MVC architecture (app/ directory)
- Resources (public/ directory)
- Configuration (conf/ directory)
- Unmanaged library dependencies (lib/ directory)
- Build definitions (project/ directory)
- Output files (target/ directory)
- Tests (test/ directory)
- Logs (log/ directory)

Expensive tasks should be performed in separate threads using Future[Result]. Async { } is an helper method that builds an AsyncResult from a Future[Result].

```
def index = Action {
  val futureInt = scala.concurrent.Future {
      intensiveComputation() }
  Async {
    futureInt.map(i => Ok("Got result: " + i))
  }
}
```

Use WebSockets instead of Comet when possible!

```
def index = WebSocket.using[String] { request =>
  // Log events to the console
  val in = Iteratee.foreach[String](println).mapDone {
    _ => println("Disconnected")
  }

  // Send a single 'Hello!' message
  val out = Enumerator("Hello!")

  (in, out)
}
```

Use WebSockets instead of Comet when possible!

```
def index = WebSocket.using[String] { request =>
  // Log events to the console
  val in = Iteratee.foreach[String](println).mapDone {
    _ => println("Disconnected")
  }

  // Send a single 'Hello!' message
  val out = Enumerator("Hello!")

  (in, out)
}
```

Let's see it in practice!

What iteratee do?

What iteratee do?

- iterate over an Enumerator (iteratee is consumer)

What iteratee do?

- iterate over an Enumerator (iteratee is consumer)
- accept static typed chunks and compute a static typed result (chunk by chunk)

What iteratee do?

- iterate over an Enumerator (iteratee is consumer)
- accept static typed chunks and compute a static typed result (chunk by chunk)
- can propagate the immutable context and state over iterations

What iteratee do?

- iterate over an Enumerator (iteratee is consumer)
- accept static typed chunks and compute a static typed result (chunk by chunk)
- can propagate the immutable context and state over iterations

```
val sum: Iteratee[Int,Int] =
    Iteratee.fold[Int,Int](0) { (s, e) => s + e }
```

What iteratee do?

- iterate over an Enumerator (iteratee is consumer)
- accept static typed chunks and compute a static typed result (chunk by chunk)
- can propagate the immutable context and state over iterations

```
val sum: Iteratee[Int,Int] =
    Iteratee.fold[Int,Int](0) { (s, e) => s + e }
```

Different look..

Iteratee is just a state machine in charge of looping over state Cont until it detects conditions to switch to terminal states Done or Error.

What iteratee do?

What iteratee do?

- produces statically typed chunks of data

What iteratee do?

- produces statically typed chunks of data
- requires a consumer to produce

What iteratee do?

- produces statically typed chunks of data
- requires a consumer to produce

```
val integerEnumerator: Enumerator[Int] =
  Enumerate(45, 33, 25, 45)

val fileEnumerator: Enumerator[Array[Byte]] =
  Enumerator.fromFile("some_file.txt")

val dateGenerator: Enumerator[String] =
Enumerator.generateM(
  play.api.libs.concurrent.Promise.timeout(
    Some(12345),
    500
  )
)
```

Enumeratee is kind of pipe between between Enumerator and Iteratee.

Enumeratee is kind of pipe between between Enumerator and Iteratee.

- it can be applied to an Enumerator without Iteratee

Enumeratee is kind of pipe between between Enumerator and Iteratee.

- it can be applied to an Enumerator without Iteratee
- it can transform an Iteratee

Enumeratee is kind of pipe between between Enumerator and Iteratee.

- it can be applied to an Enumerator without Iteratee
- it can transform an Iteratee
- it can be composed with other Enumeratee

Enumeratee is kind of pipe between between Enumerator and Iteratee.

- it can be applied to an Enumerator without Iteratee
- it can transform an Iteratee
- it can be composed with other Enumeratee

```
val enumerator = Enumerator(123, 345, 456)
val iteratee: Iteratee[String, List[String]] = ???

val list: List[String] =
  enumerator through Enumeratee.map(_.toString) run
      iteratee
```

What now?

What now?

- Iteratees with WebSockets in practice

What now?

- Iteratees with WebSockets in practice
- Communication between multiple actors with Akka

What now?

- Iteratees with WebSockets in practice
- Communication between multiple actors with Akka
- Json serialization/deserialization