MOG in Clojure
Attack of the clones
(Episode II)

Hello!

I am **Mey Beisaron**

- Software engineer
- Infra Developer at FORTER
- Writing in : Clojure, Nodejs, Groovy, Python
- Public speaker & Mentor
- Sworn Star Wars fan

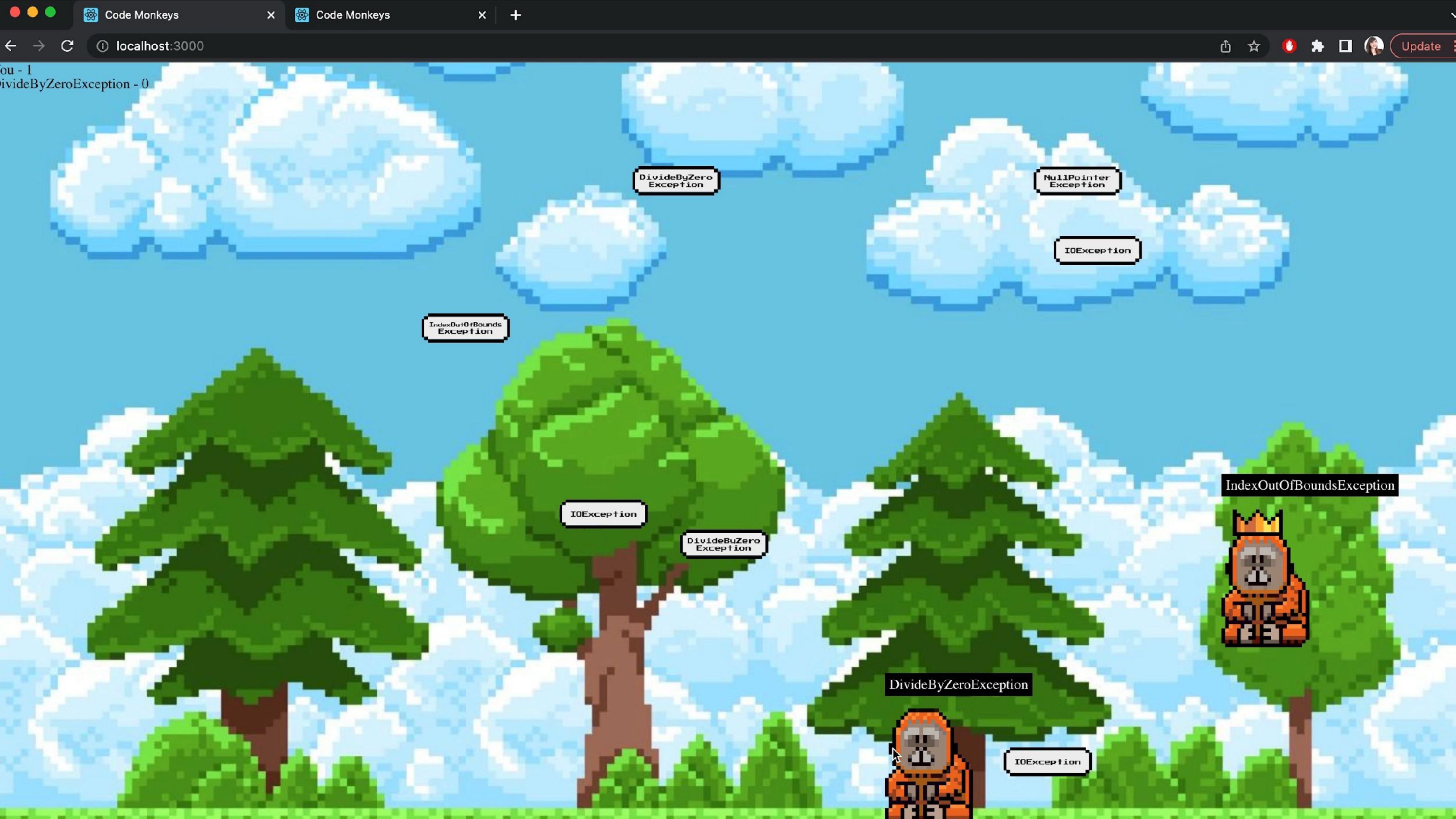
I know why you're here…

@Ladymey

You want to develop a MOG!

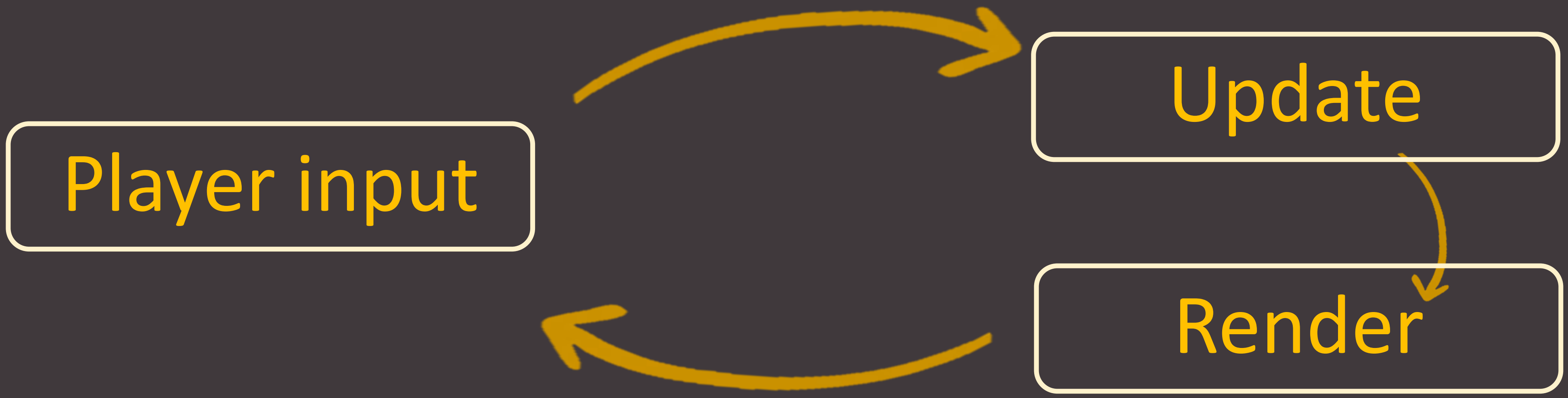
@Ladymey

it won’t be
Star Wars Battlefront II



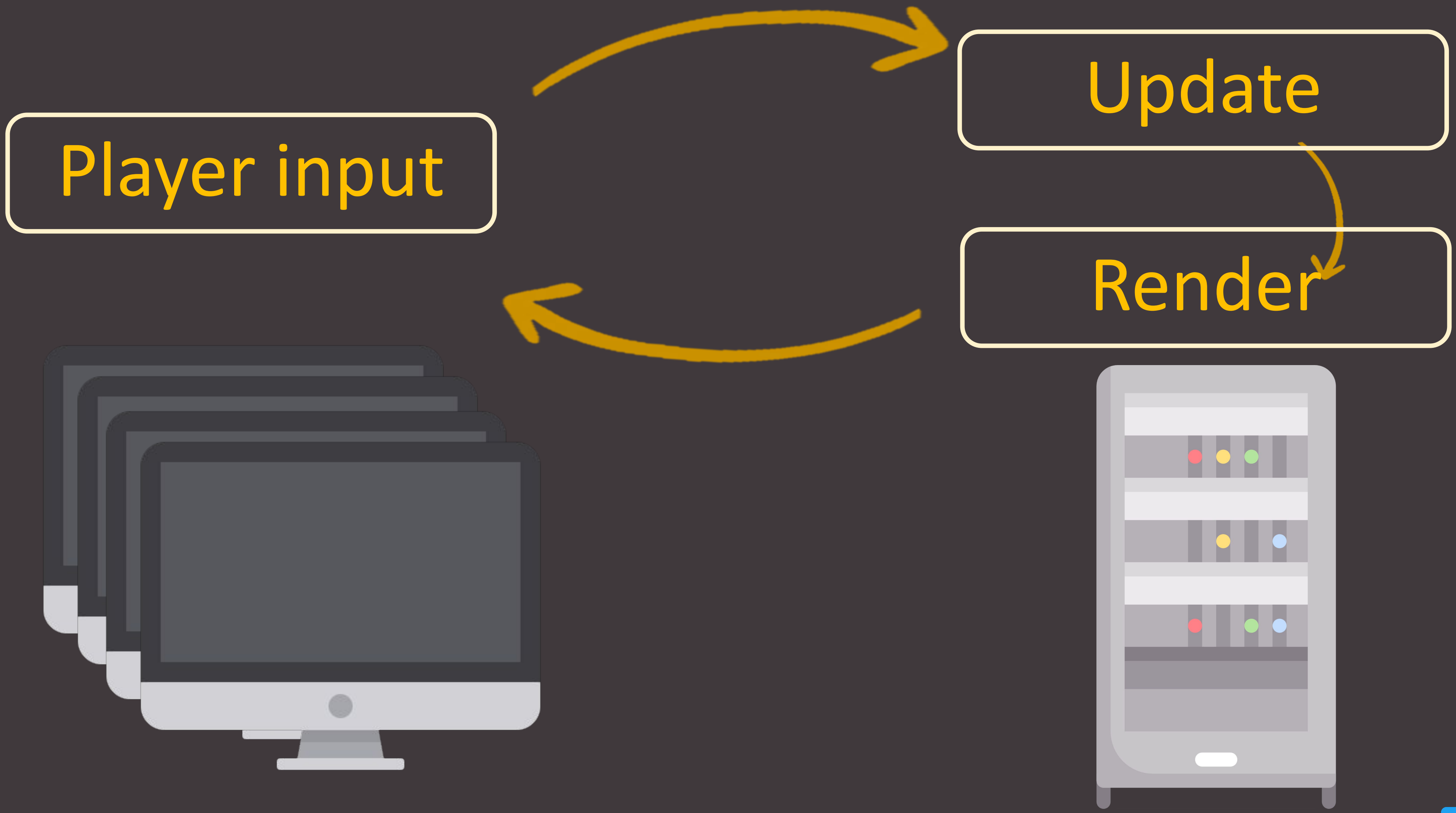
```
ou - 1
ivideByZeroException - 0
```

The Game Loop

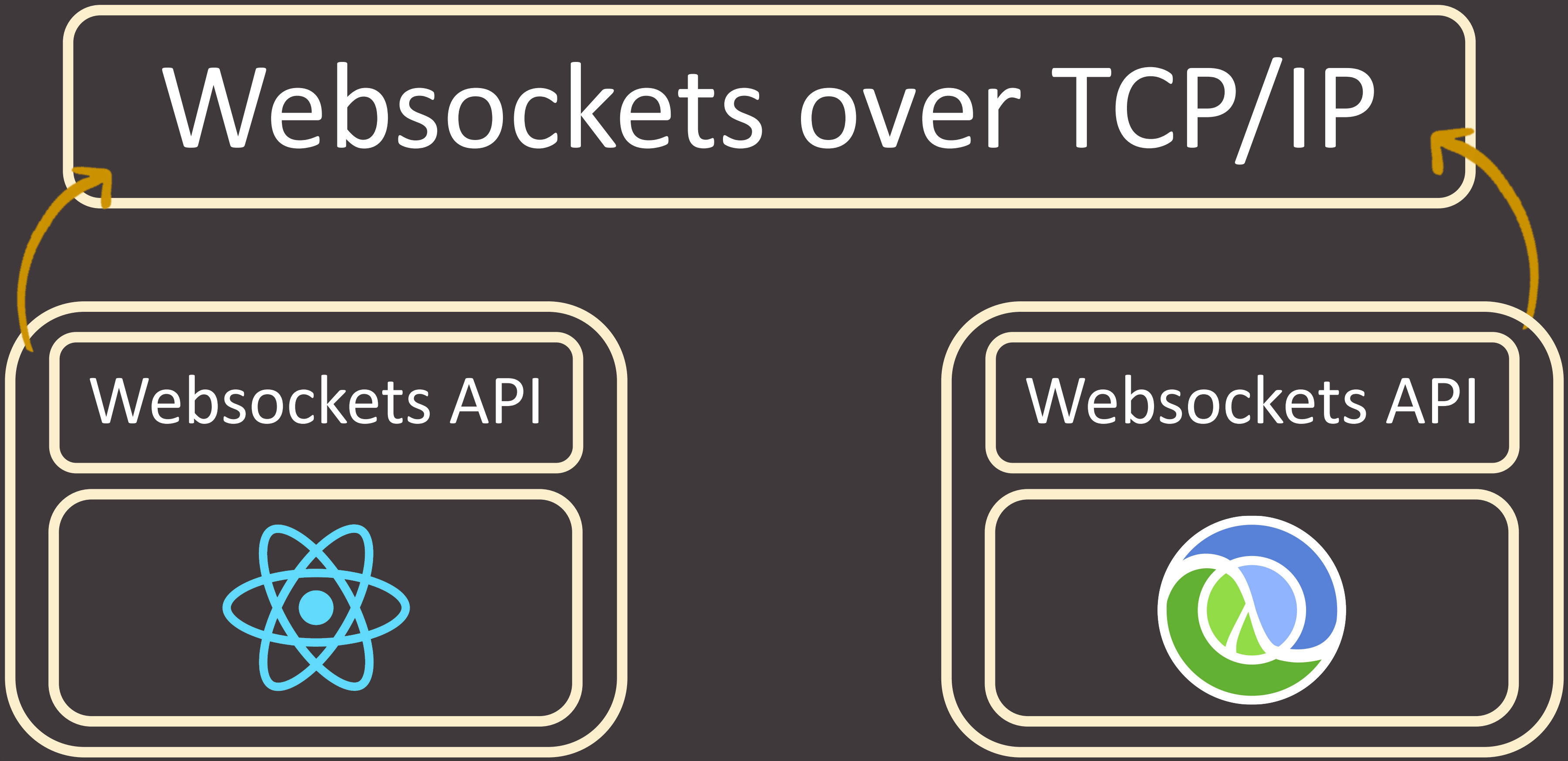


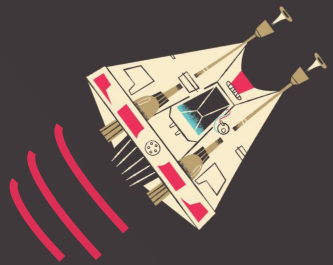
@Ladymey

The Game Loop (online)



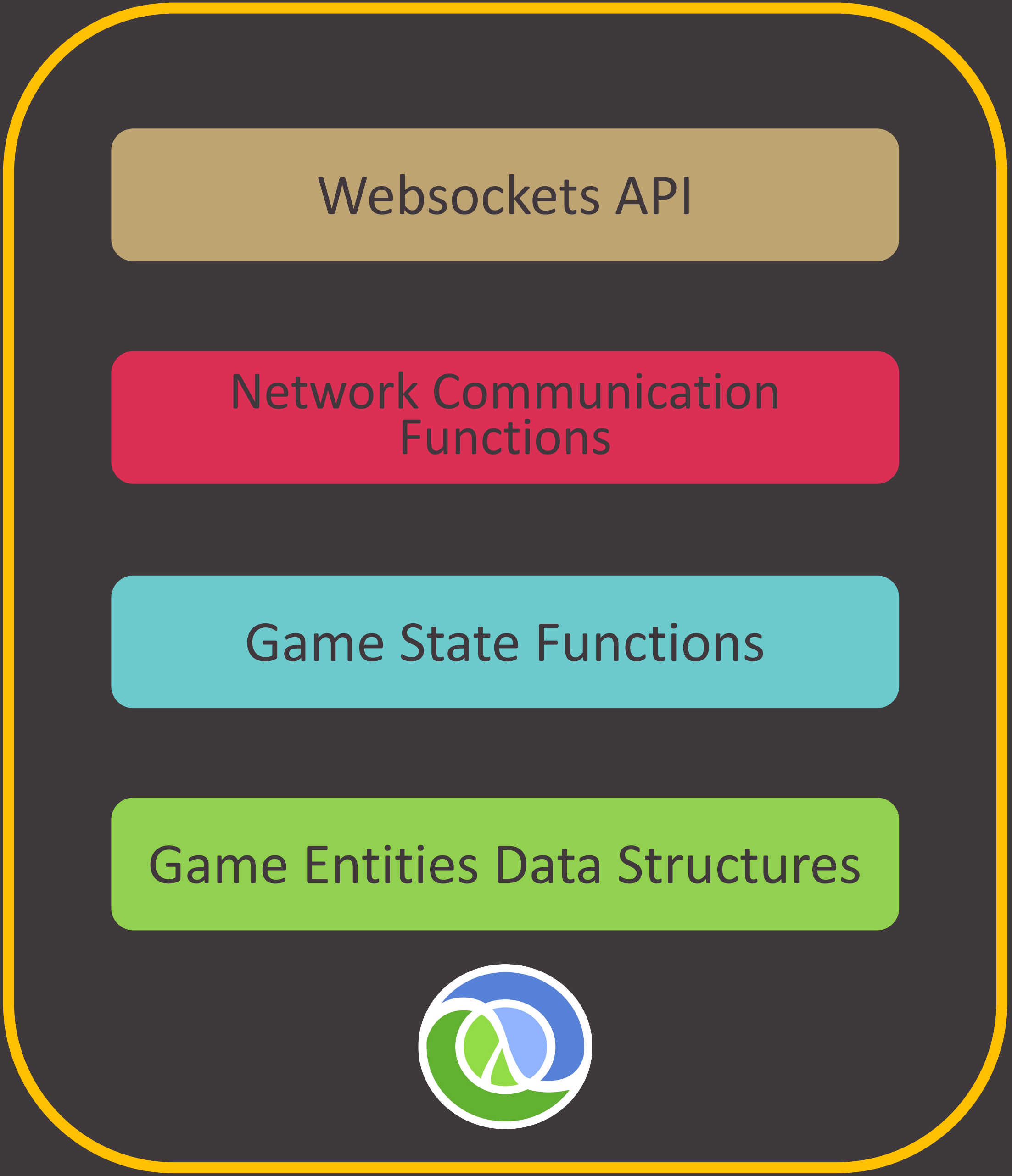
Game Architecture





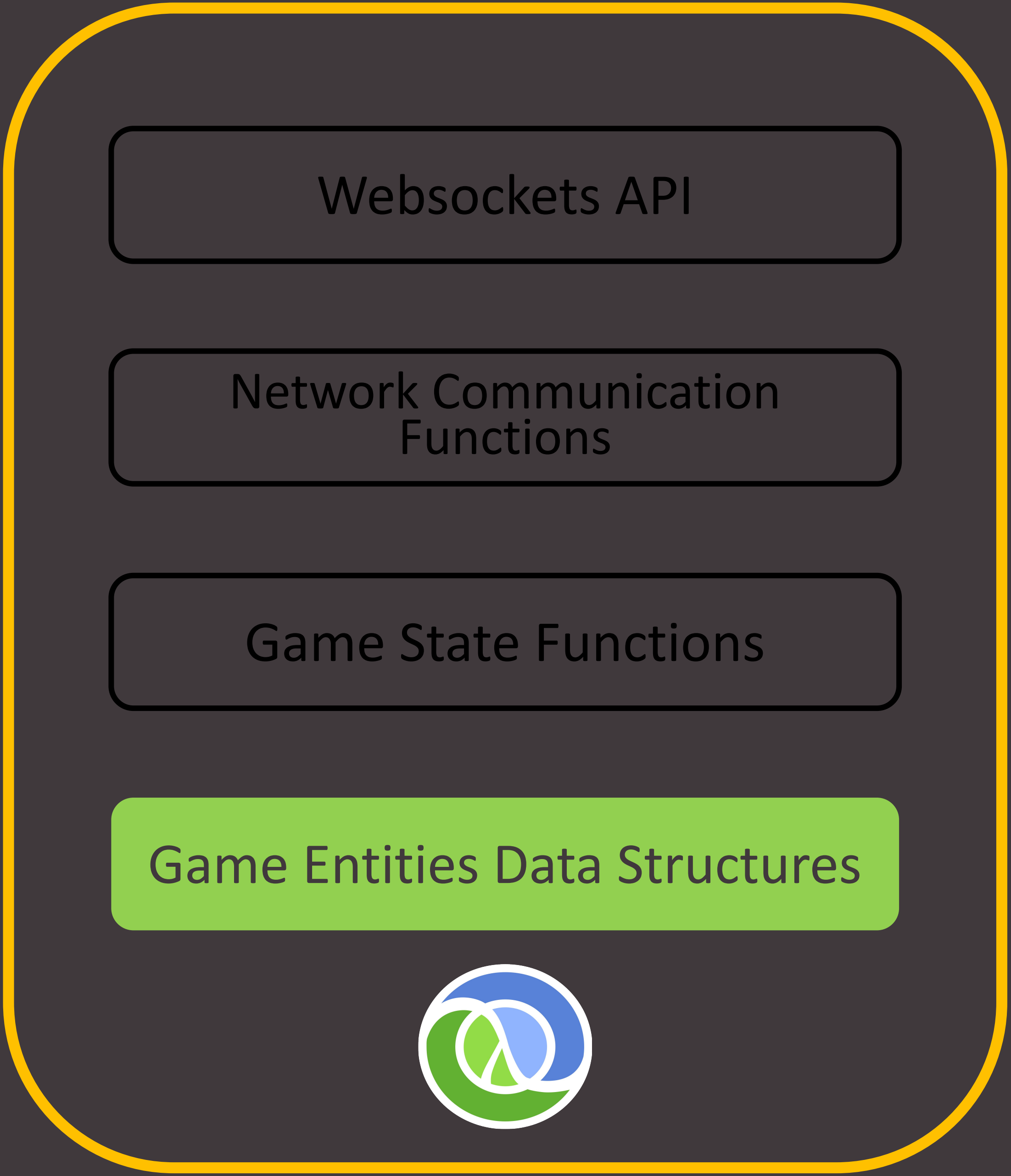
@Ladymey

Server Architecture


Websockets API
Network Communication Functions
Game State Functions
Game Entities Data Structures

@Ladymey

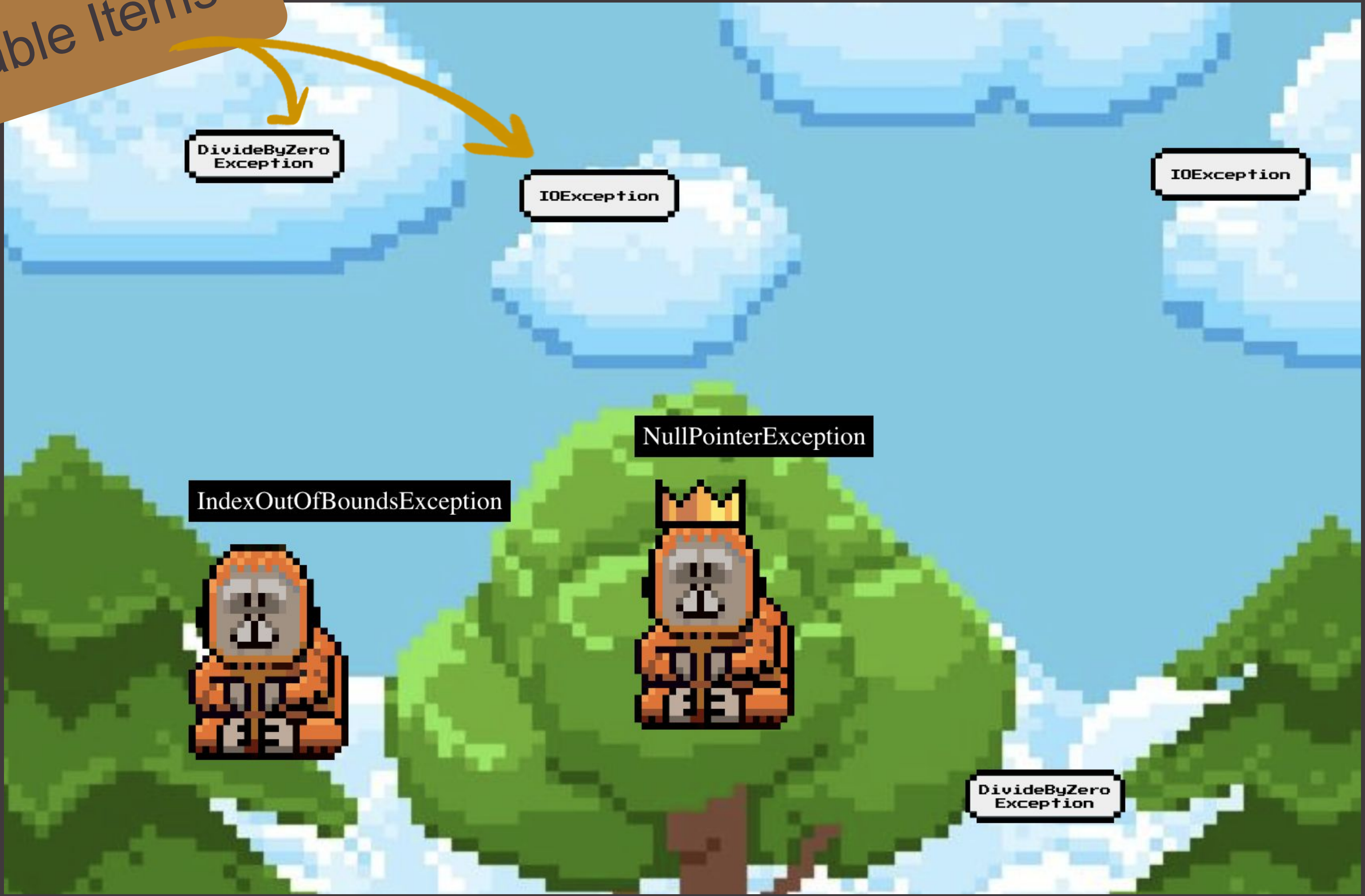
Server Architecture



Websockets API
Network Communication Functions
Game State Functions
Game Entities Data Structures

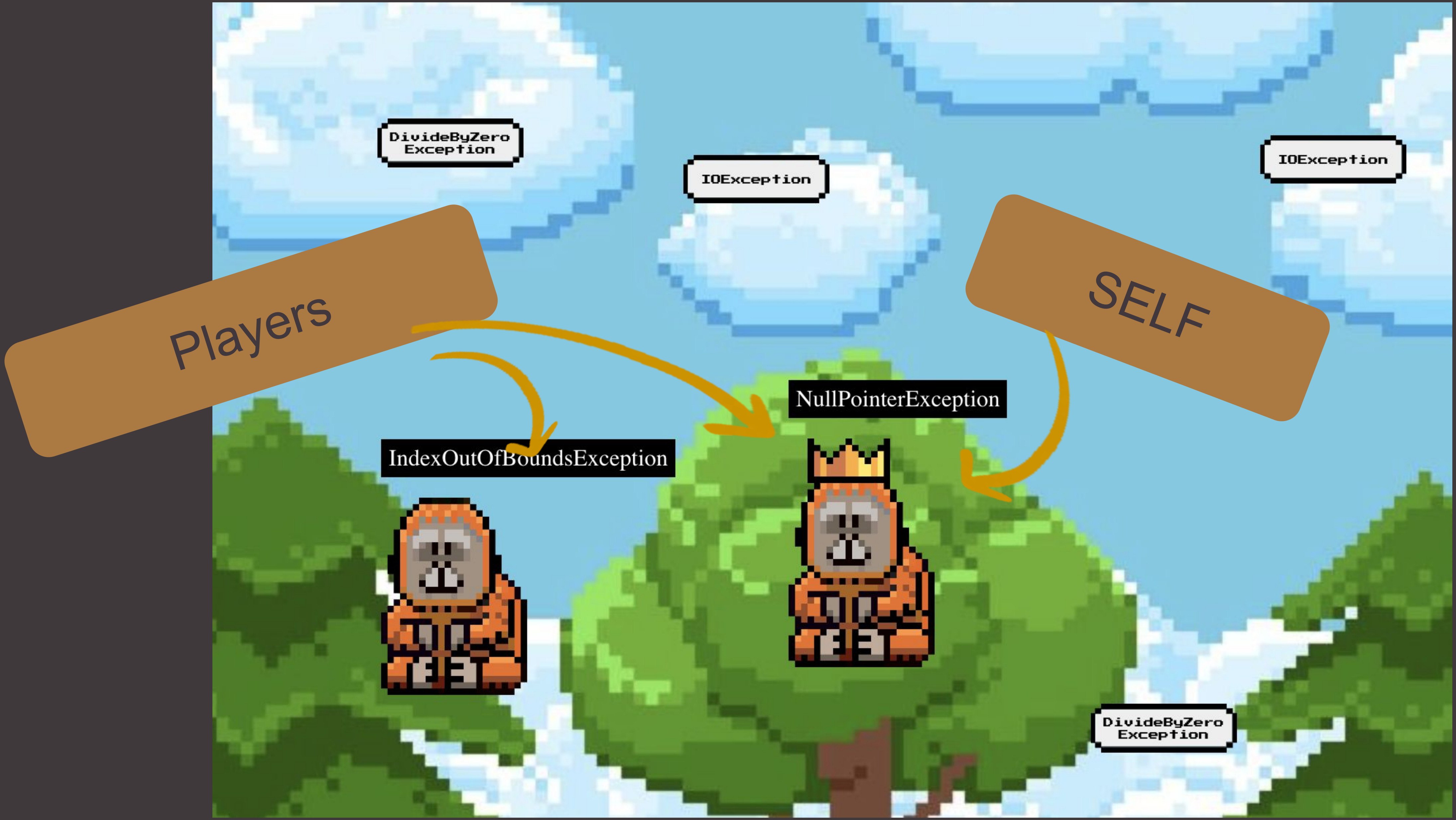
@Ladymey

Collectable Items



DivideByZero Exception
IOException
IOException
Players
NullPointerException
IndexOutOfBoundsException
DivideByZero Exception

@Ladymey


DivideByZero Exception
IOException
IOException
Players
SELF
NullPointerException
IndexOutOfBoundsException
DivideByZero Exception

Game Entities Data Structures

```clojure
(def exceptionTypes ["IOException", "DivideByZeroException",
                     "NullPointerException", "IndexOutOfBoundsException"])
(def items (atom {}))
(def players (atom {}))
```

@Ladymey

Game Entities Data Structures

```
(def exceptionTypes ["IOException", "DivideByZeroException",
                     "NullPointerException", "IndexOutOfBoundsException"])
(def items (atom {}))
(def players (atom {}))
```

@Ladymey

Game Entities Data Structures

```
(def exceptionTypes ["IOException", "DivideByZeroException",
                     "NullPointerException", "IndexOutOfBoundsException"])
(def items (atom {}))
(def players (atom {}))
```

@Ladymey

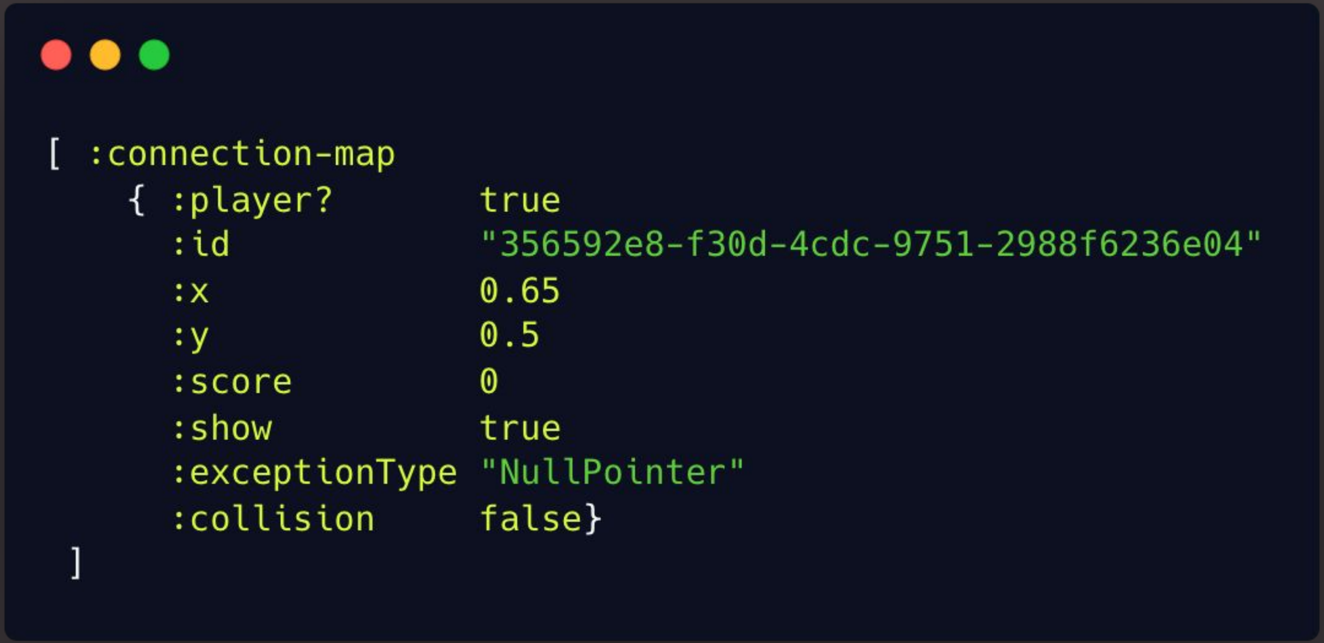
Game Entities Data Structures

```
(def exceptionTypes ["IOException", "DivideByZeroException",
                     "NullPointerException", "IndexOutOfBoundsException"])
(def items (atom {}))
(def players (atom {}))
```

@Ladymey

Game Entities Data Structures

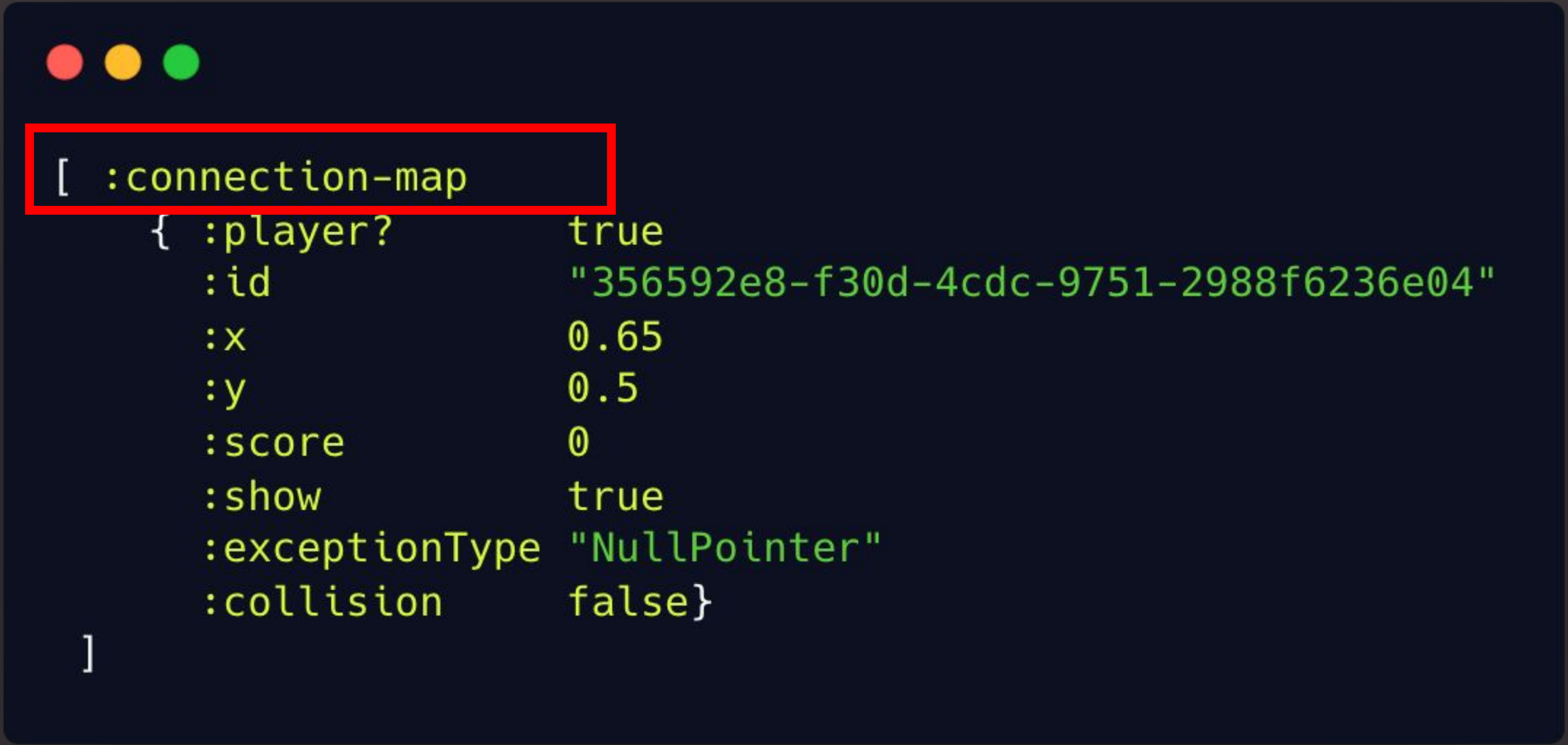
```clojure
(def exceptionTypes ["IOException", "DivideByZeroException",
                     "NullPointerException", "IndexOutOfBoundsException"])
(def items (atom {}))
(def players (atom {}))
```



```
[ :connection-map
    { :player?        true
      :id             "356592e8-f30d-4cdc-9751-2988f6236e04"
      :x              0.65
      :y              0.5
      :score          0
      :show           true
      :exceptionType "NullPointer"
      :collision      false}
 ]
```

@Ladymey

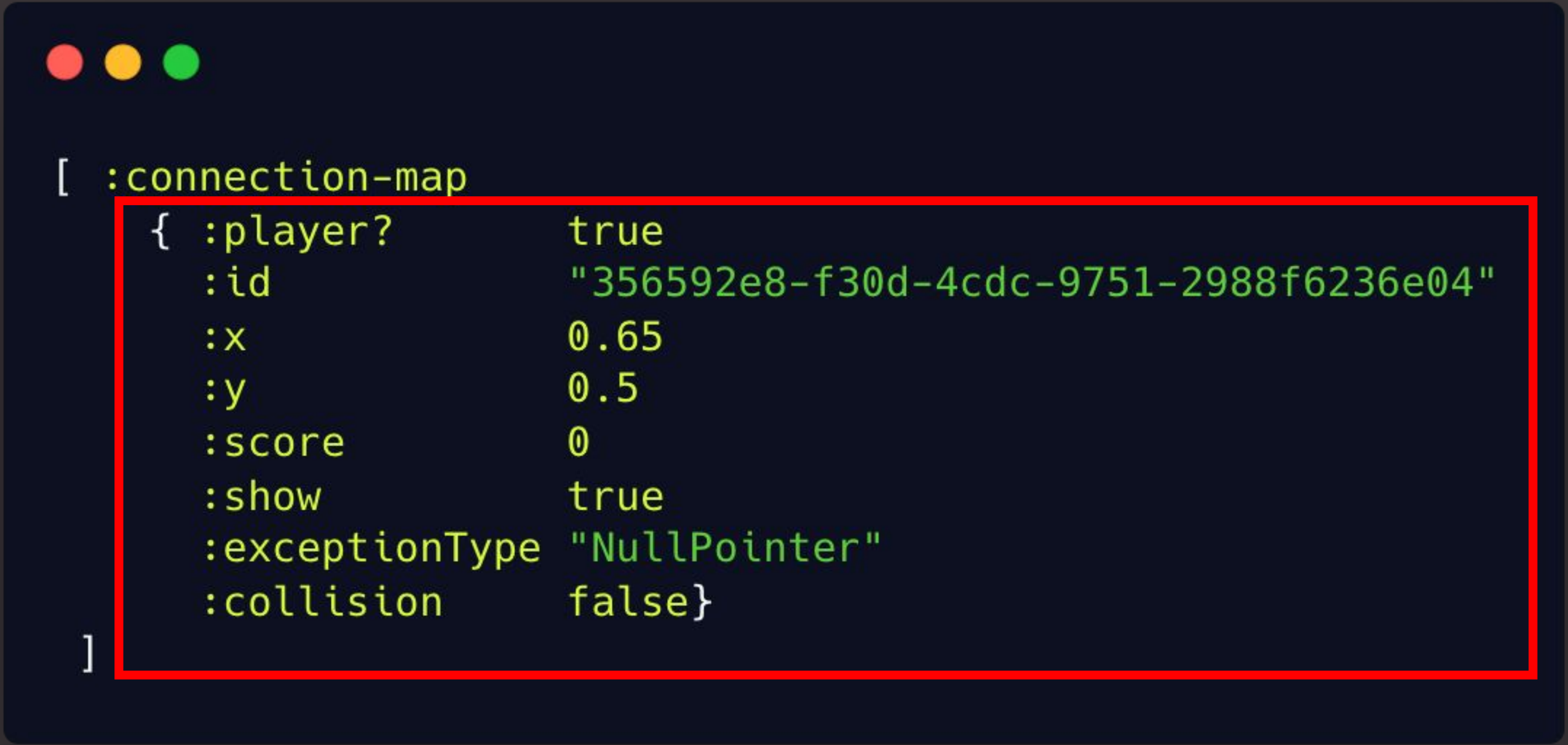
Game Entities Data Structures



```
[ :connection-map
    { :player?       true
      :id            "356592e8-f30d-4cdc-9751-2988f6236e04"
      :x             0.65
      :y             0.5
      :score         0
      :show          true
      :exceptionType "NullPointer"
      :collision     false}
 ]
```

@Ladymey

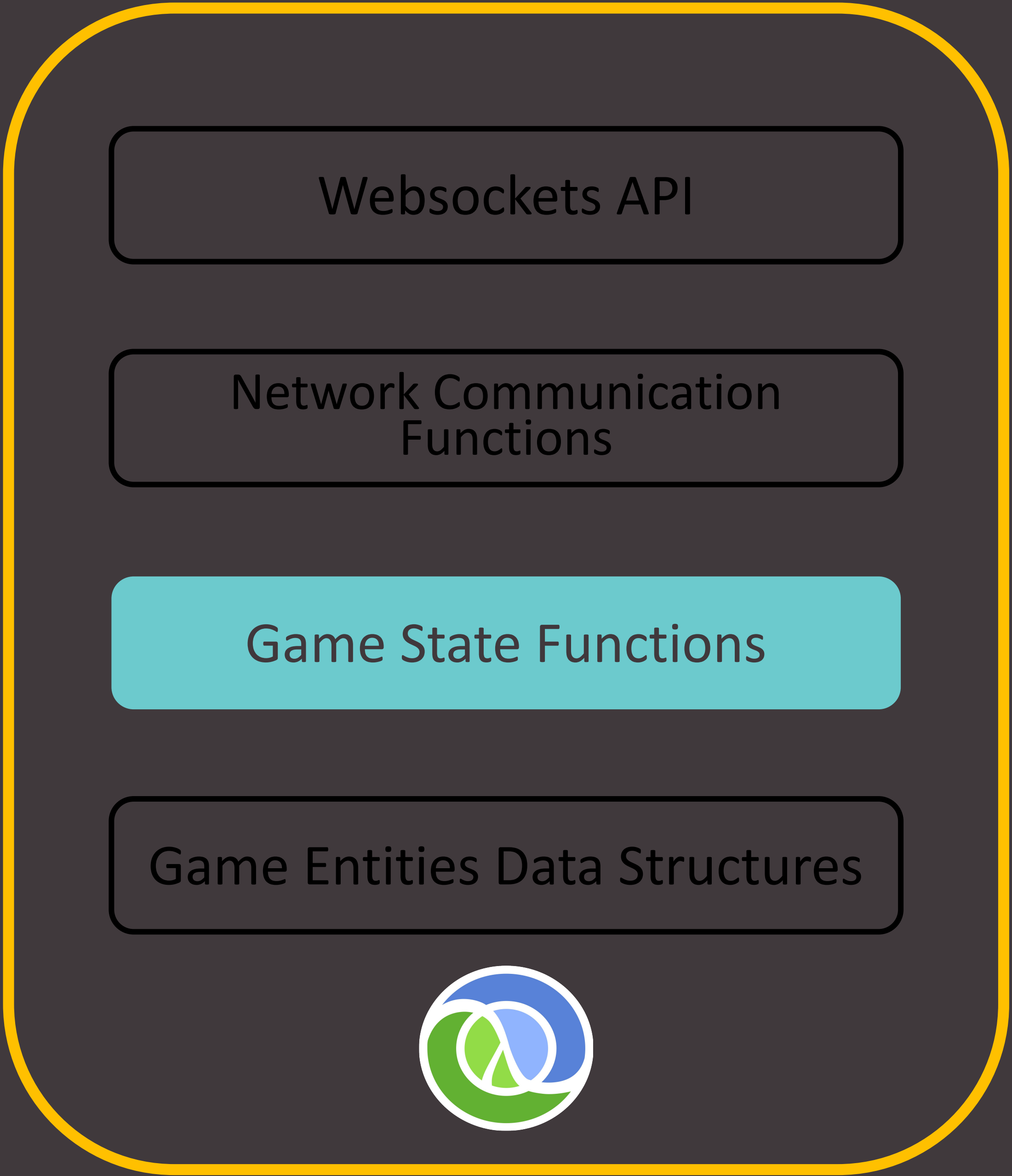
Game Entities Data Structures



```
[ :connection-map
  { :player?       true
    :id            "356592e8-f30d-4cdc-9751-2988f6236e04"
    :x             0.65
    :y             0.5
    :score         0
    :show          true
    :exceptionType "NullPointer"
    :collision     false}
 ]
```

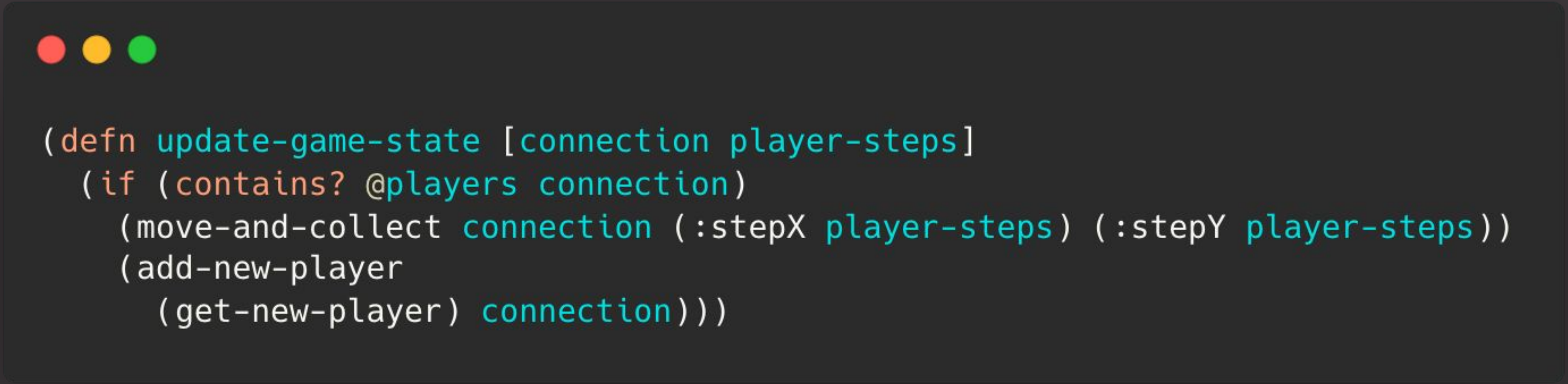
@Ladymey

Server Architecture


Websockets API
Network Communication Functions
Game State Functions
Game Entities Data Structures

@Ladymey

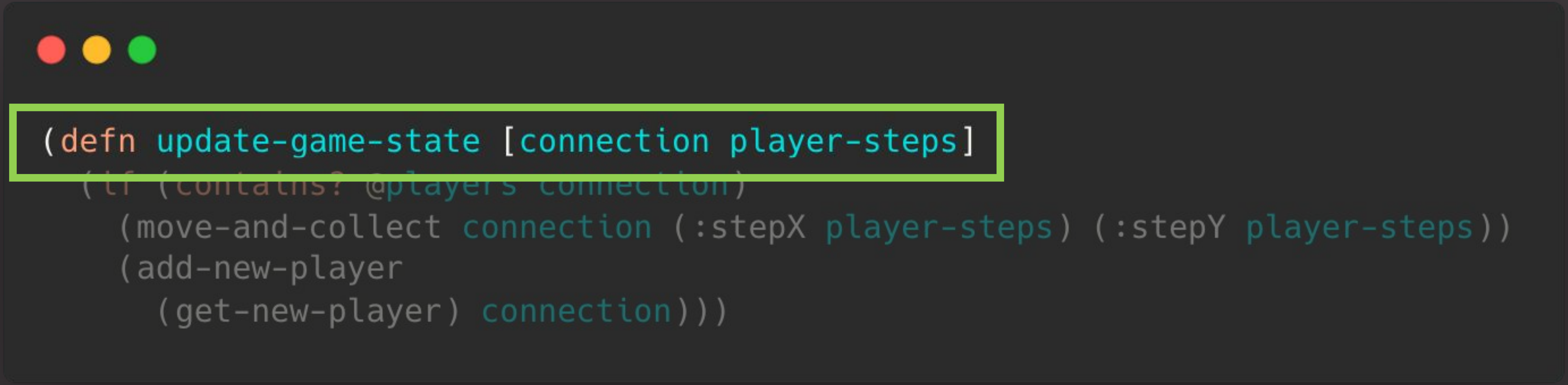
Manage Game State



```
(defn update-game-state [connection player-steps]
  (if (contains? @players connection)
    (move-and-collect connection (:stepX player-steps) (:stepY player-steps))
    (add-new-player
      (get-new-player) connection)))
```

@Ladymey

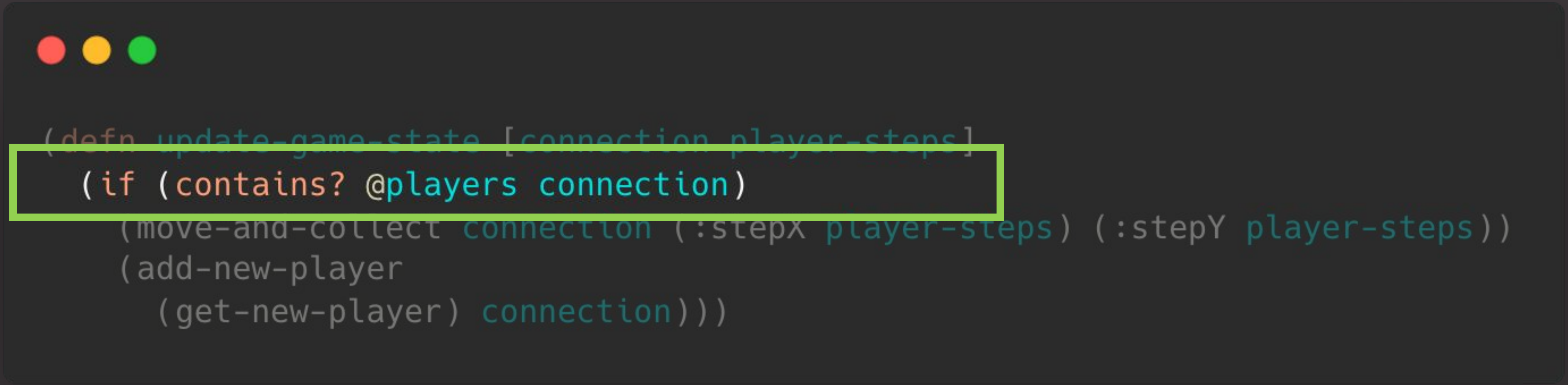
Manage Game State



```
(defn update-game-state [connection player-steps]
  (if (contains? @players connection)
    (move-and-collect connection (:stepX player-steps) (:stepY player-steps))
    (add-new-player
      (get-new-player) connection)))
```

@Ladymey

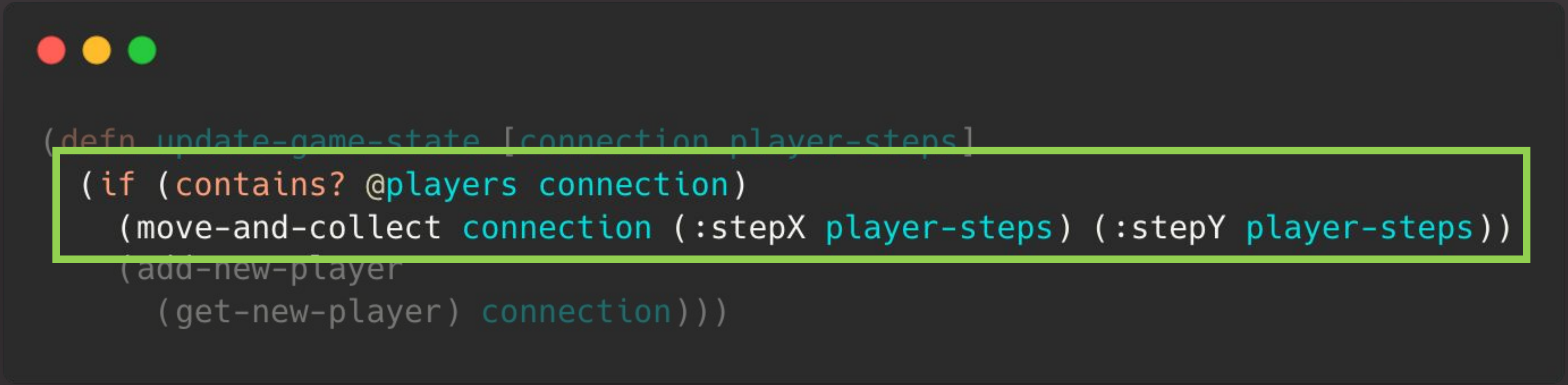
Manage Game State



```
(defn update-game-state [connection player-steps]
  (if (contains? @players connection)
    (move-and-collect connection (:stepX player-steps) (:stepY player-steps))
    (add-new-player
      (get-new-player) connection)))
```

@Ladymey

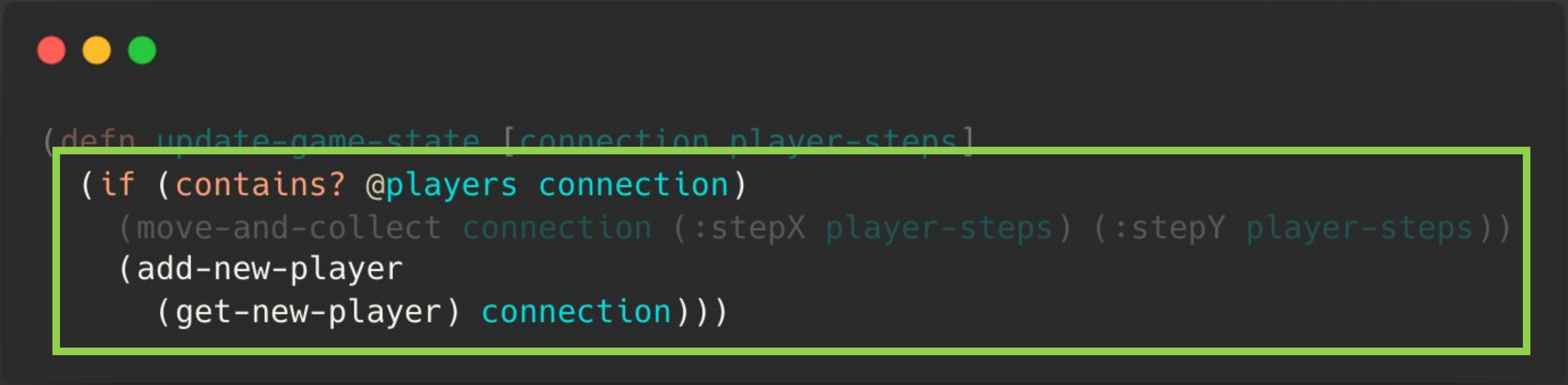
Manage Game State



```
(defn update-game-state [connection player-steps]
  (if (contains? @players connection)
    (move-and-collect connection (:stepX player-steps) (:stepY player-steps))
    (add-new-player
      (get-new-player) connection)))
```

@Ladymey

Manage Game State



```
(defn update-game-state [connection player-steps]
  (if (contains? @players connection)
    (move-and-collect connection (:stepX player-steps) (:stepY player-steps))
    (add-new-player
      (get-new-player) connection)))
```

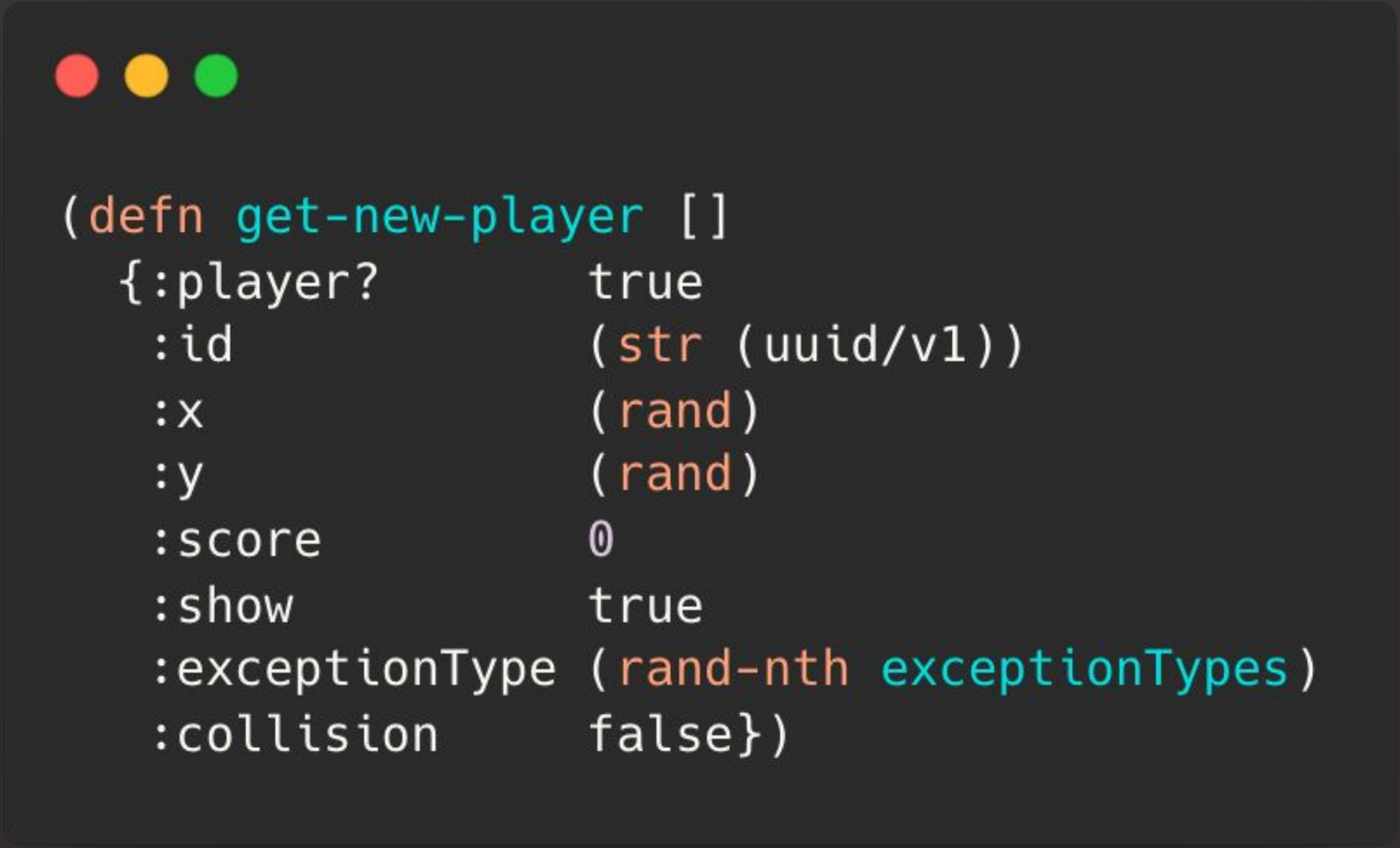
@Ladymey

Manage Game State

1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items

Manage Game State

1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items

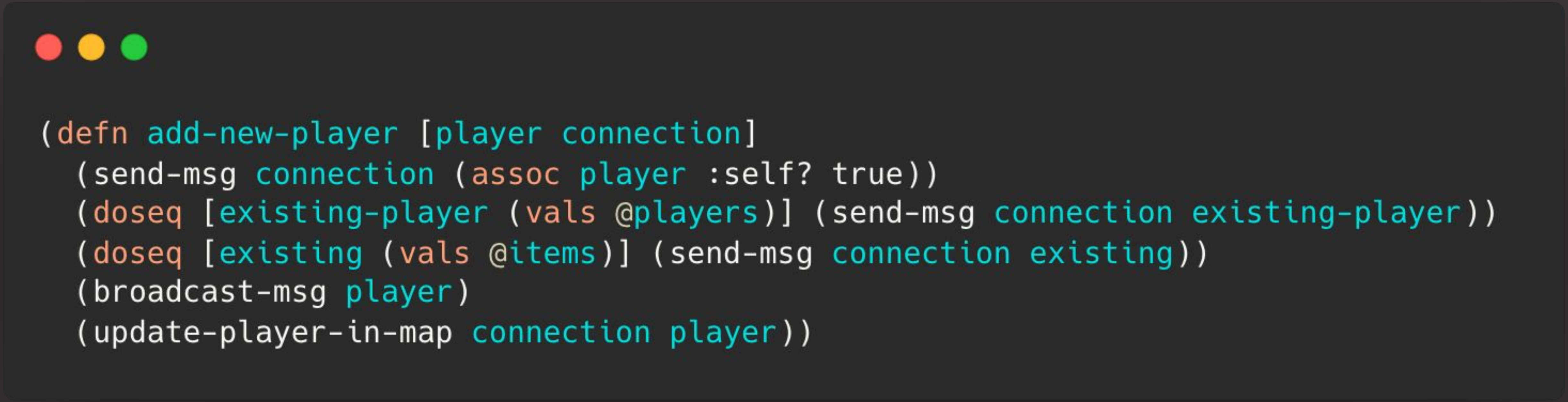


```
(defn get-new-player []
  {:player?        true
   :id             (str (uuid/v1))
   :x              (rand)
   :y              (rand)
   :score          0
   :show           true
   :exceptionType (rand-nth exceptionTypes)
   :collision      false})
```

@Ladymey

Manage Game State

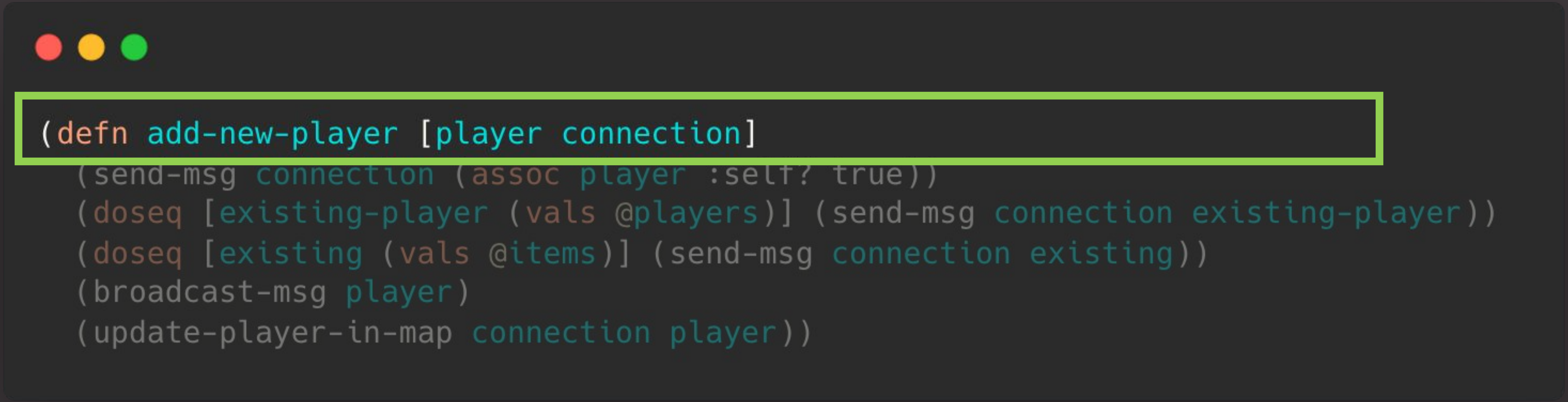
1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items



```
(defn add-new-player [player connection]
  (send-msg connection (assoc player :self? true))
  (doseq [existing-player (vals @players)] (send-msg connection existing-player))
  (doseq [existing (vals @items)] (send-msg connection existing))
  (broadcast-msg player)
  (update-player-in-map connection player))
```

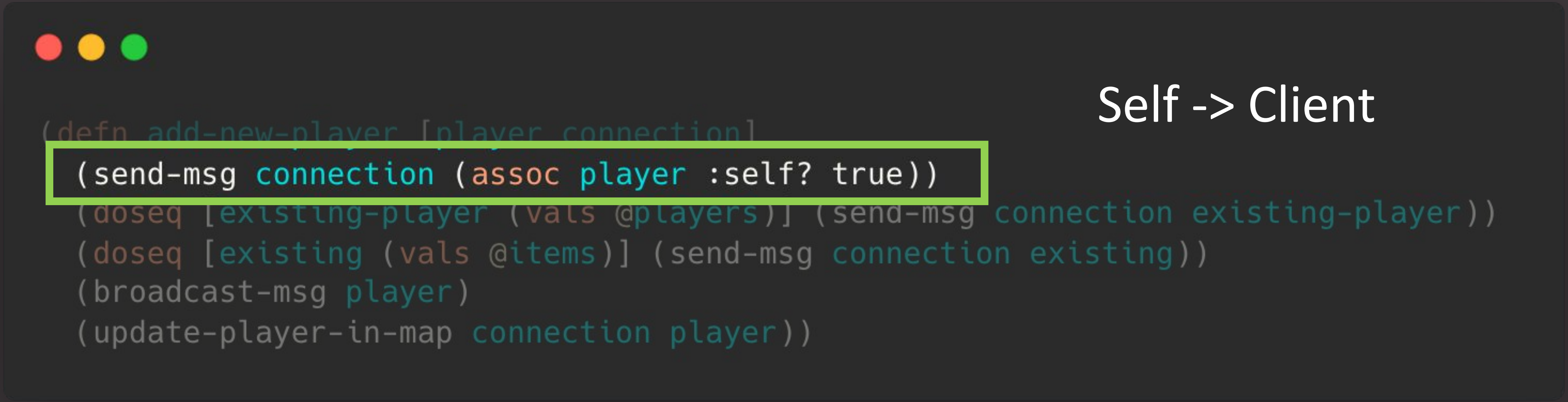
@Ladymey

Manage Game State


```
(defn add-new-player [player connection]
  (send-msg connection (assoc player :self? true))
  (doseq [existing-player (vals @players)] (send-msg connection existing-player))
  (doseq [existing (vals @items)] (send-msg connection existing))
  (broadcast-msg player)
  (update-player-in-map connection player))
```

@Ladymey

Manage Game State

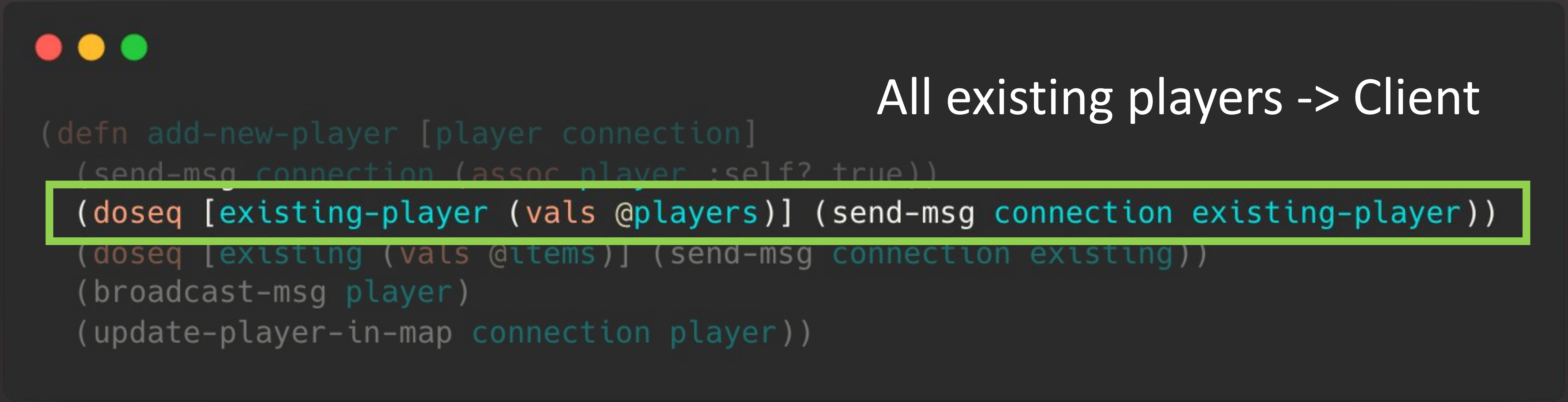


```
(defn add-new-player [player connection]
  (send-msg connection (assoc player :self? true))
  (doseq [existing-player (vals @players)] (send-msg connection existing-player))
  (doseq [existing (vals @items)] (send-msg connection existing))
  (broadcast-msg player)
  (update-player-in-map connection player))
```

Self -> Client

@Ladymey

Manage Game State

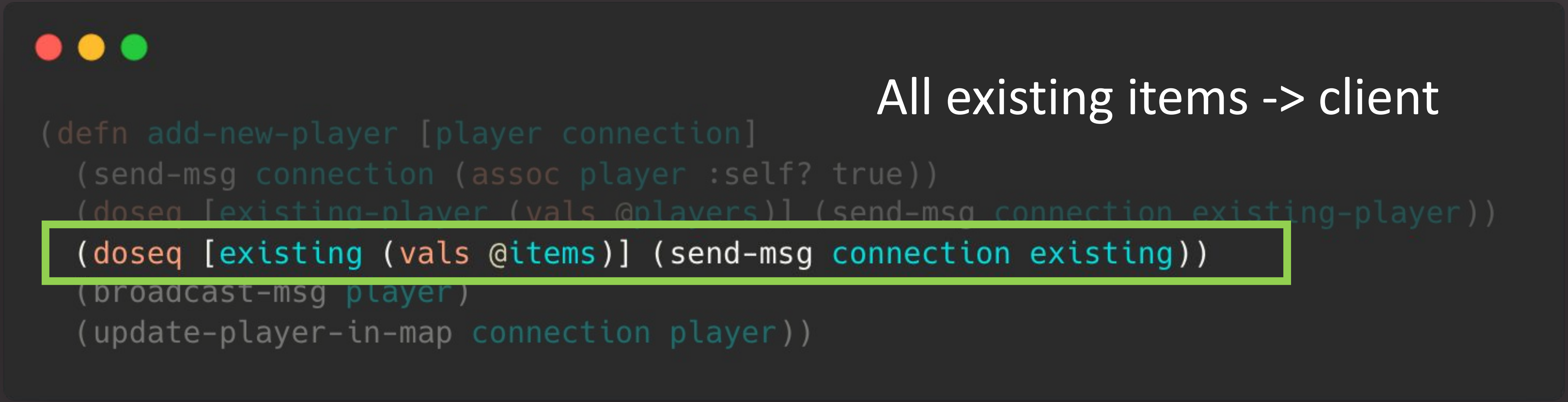


All existing players -> Client

```clojure
(defn add-new-player [player connection]
  (send-msg connection (assoc player :self? true))
  (doseq [existing-player (vals @players)] (send-msg connection existing-player))
  (doseq [existing (vals @items)] (send-msg connection existing))
  (broadcast-msg player)
  (update-player-in-map connection player))
```

@Ladymey

Manage Game State

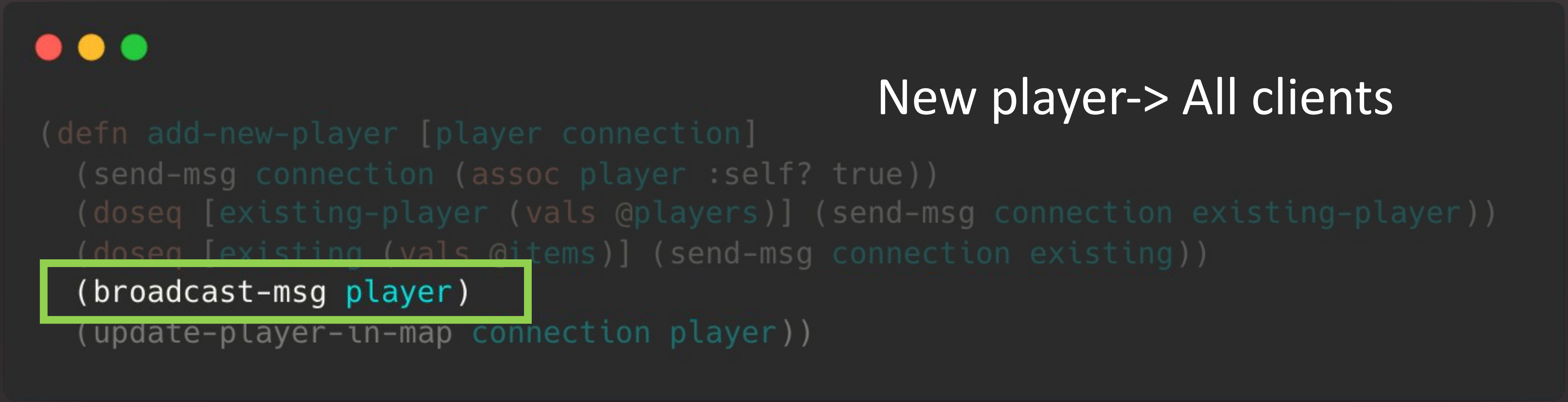


All existing items -> client

```
(defn add-new-player [player connection]
  (send-msg connection (assoc player :self? true))
  (doseq [existing-player (vals @players)] (send-msg connection existing-player))
  (doseq [existing (vals @items)] (send-msg connection existing))
  (broadcast-msg player)
  (update-player-in-map connection player))
```

@Ladymey

Manage Game State

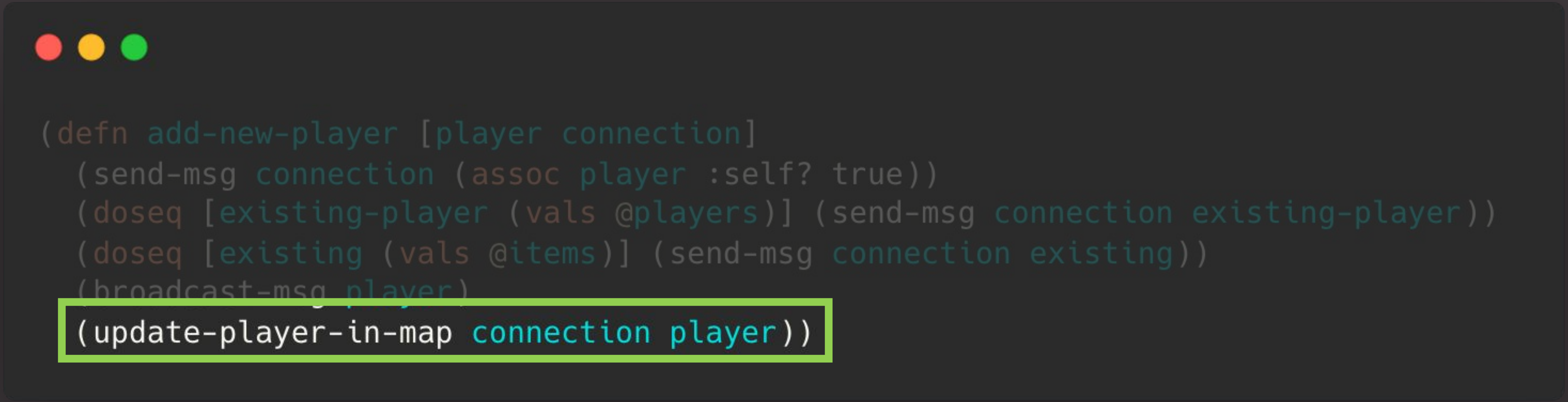


New player-> All clients

```
(defn add-new-player [player connection]
  (send-msg connection (assoc player :self? true))
  (doseq [existing-player (vals @players)] (send-msg connection existing-player))
  (doseq [existing (vals @items)] (send-msg connection existing))
  (broadcast-msg player)
  (update-player-in-map connection player))
```

@Ladymey

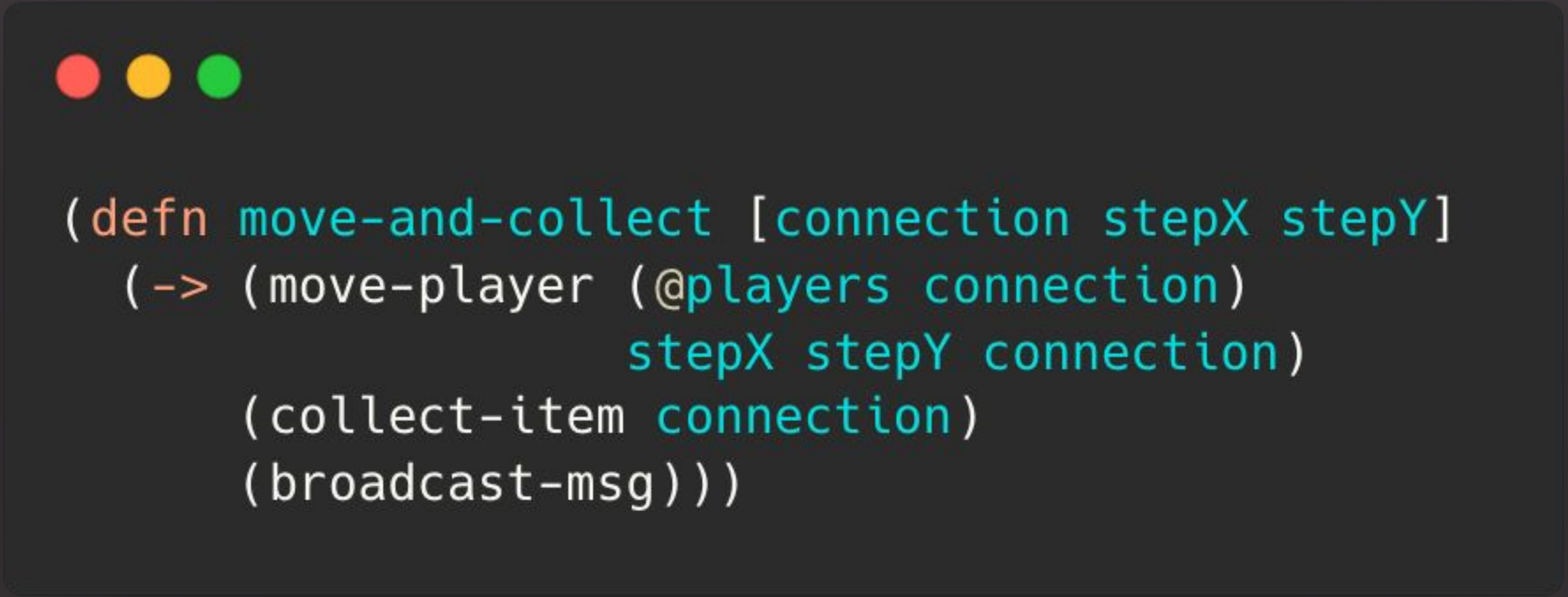
Manage Game State



```
(defn add-new-player [player connection]
  (send-msg connection (assoc player :self? true))
  (doseq [existing-player (vals @players)] (send-msg connection existing-player))
  (doseq [existing (vals @items)] (send-msg connection existing))
  (broadcast-msg player)
  (update-player-in-map connection player))
```

Manage Game State

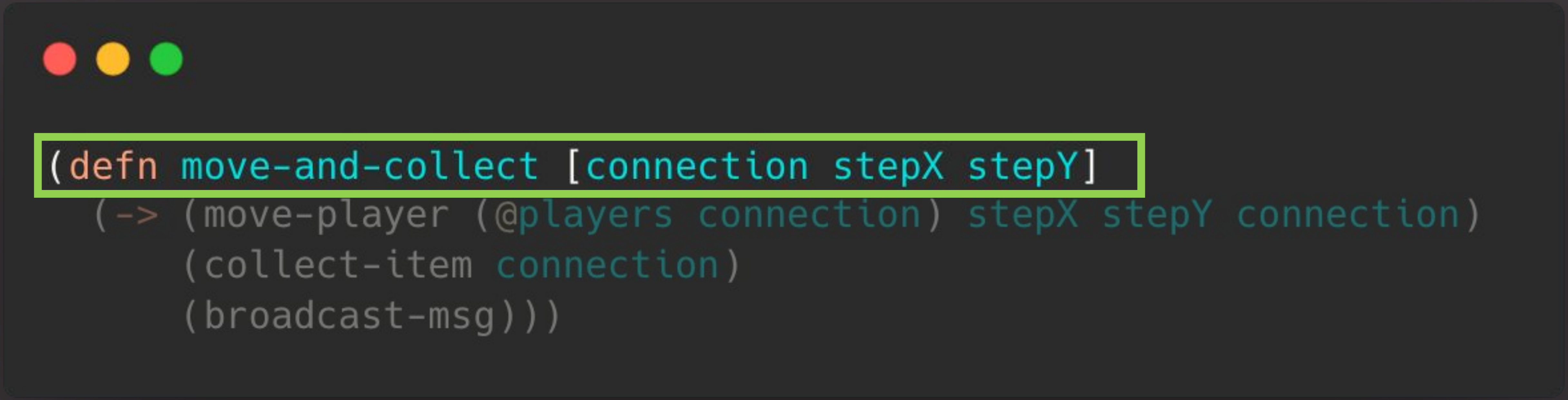
1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items



```clojure
(defn move-and-collect [connection stepX stepY]
  (-> (move-player (@players connection)
                   stepX stepY connection)
      (collect-item connection)
      (broadcast-msg)))
```

@Ladymey

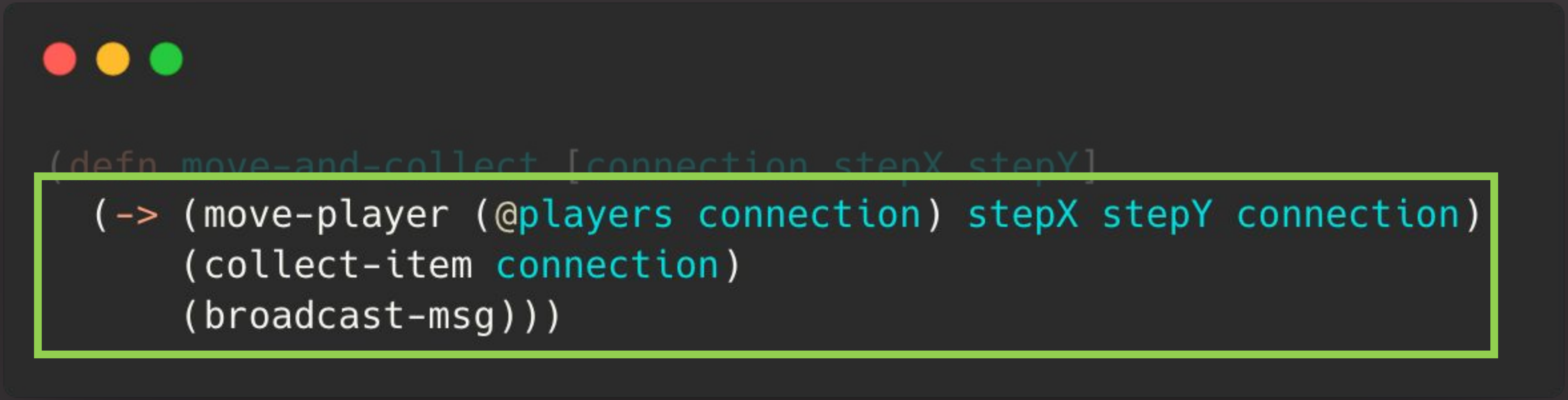
Manage Game State



```clojure
(defn move-and-collect [connection stepX stepY]
  (-> (move-player (@players connection) stepX stepY connection)
      (collect-item connection)
      (broadcast-msg)))
```

@Ladymey

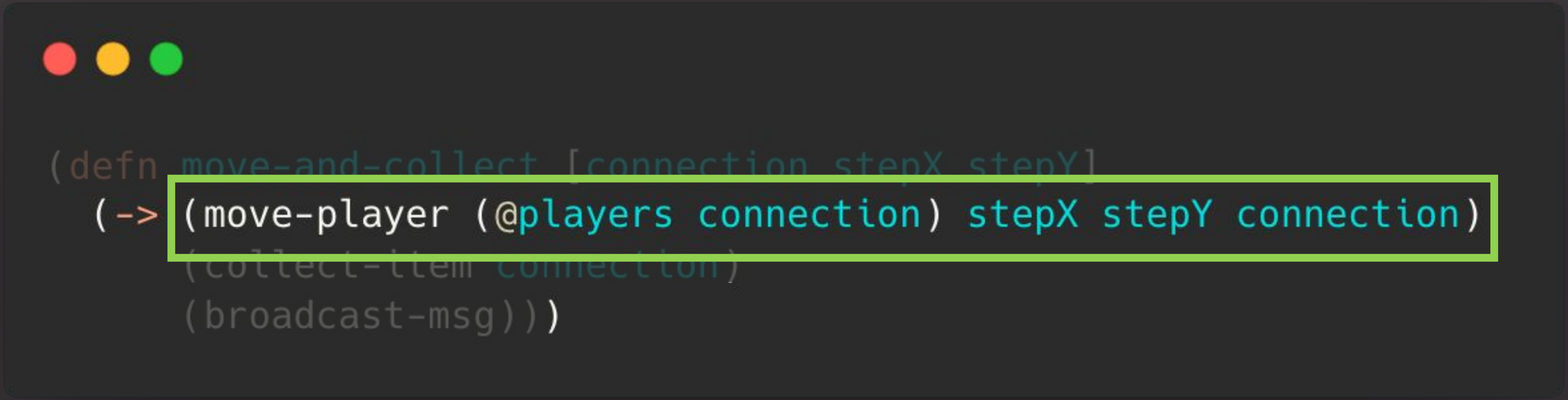
Manage Game State



```
(defn move-and-collect [connection stepX stepY]
  (-> (move-player (@players connection) stepX stepY connection)
      (collect-item connection)
      (broadcast-msg)))
```

@Ladymey

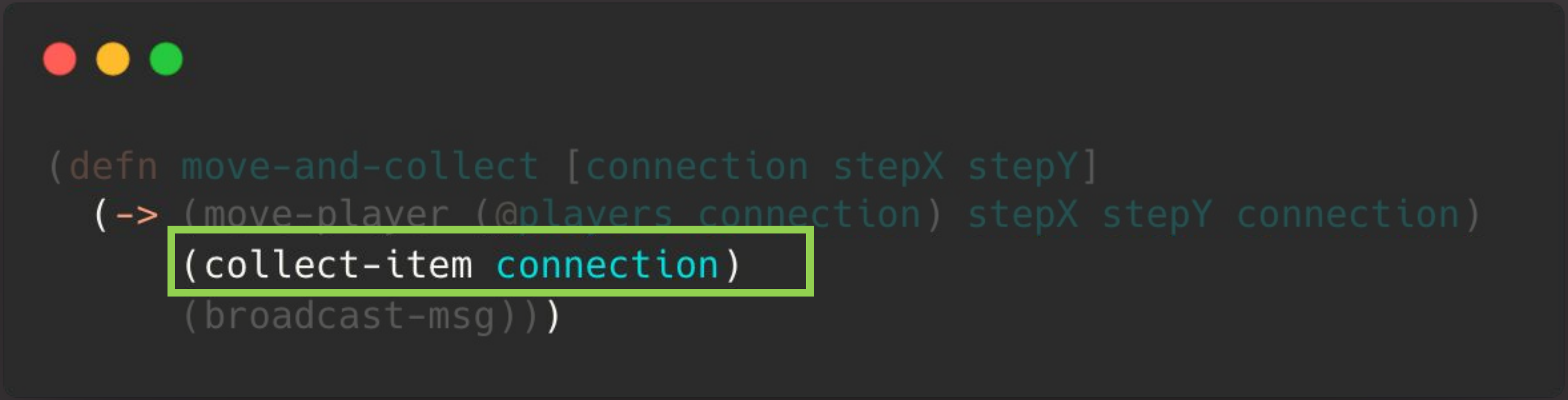
Manage Game State



```
(defn move-and-collect [connection stepX stepY]
  (-> (move-player (@players connection) stepX stepY connection)
      (collect-item connection)
      (broadcast-msg)))
```

@Ladymey

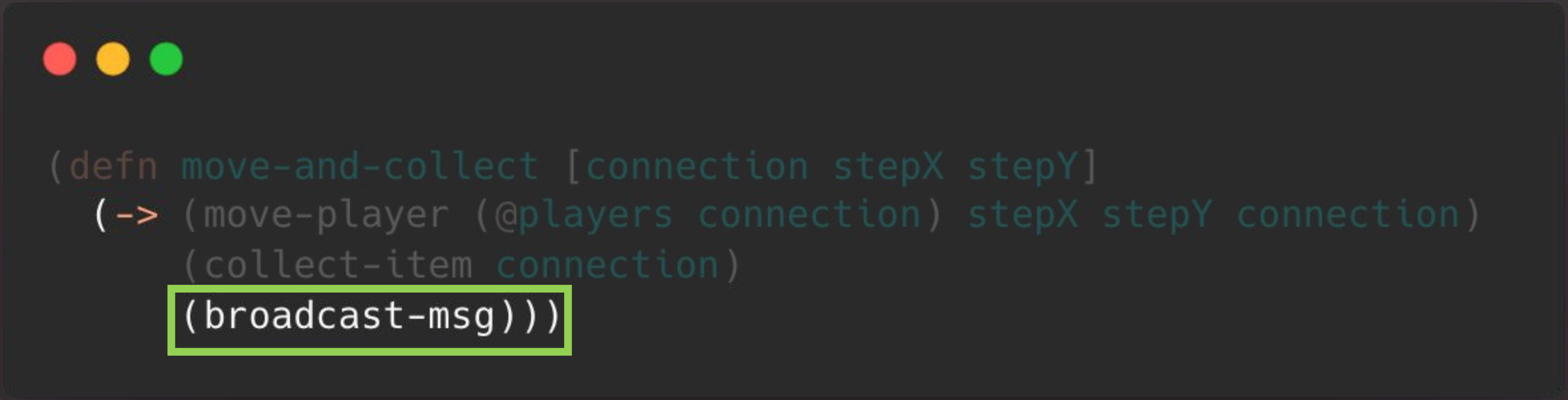
Manage Game State



```
(defn move-and-collect [connection stepX stepY]
  (-> (move-player (@players connection) stepX stepY connection)
      (collect-item connection)
      (broadcast-msg)))
```

@Ladymey

Manage Game State



```
(defn move-and-collect [connection stepX stepY]
  (-> (move-player (@players connection) stepX stepY connection)
      (collect-item connection)
      (broadcast-msg)))
```

@Ladymey

Manage Game State

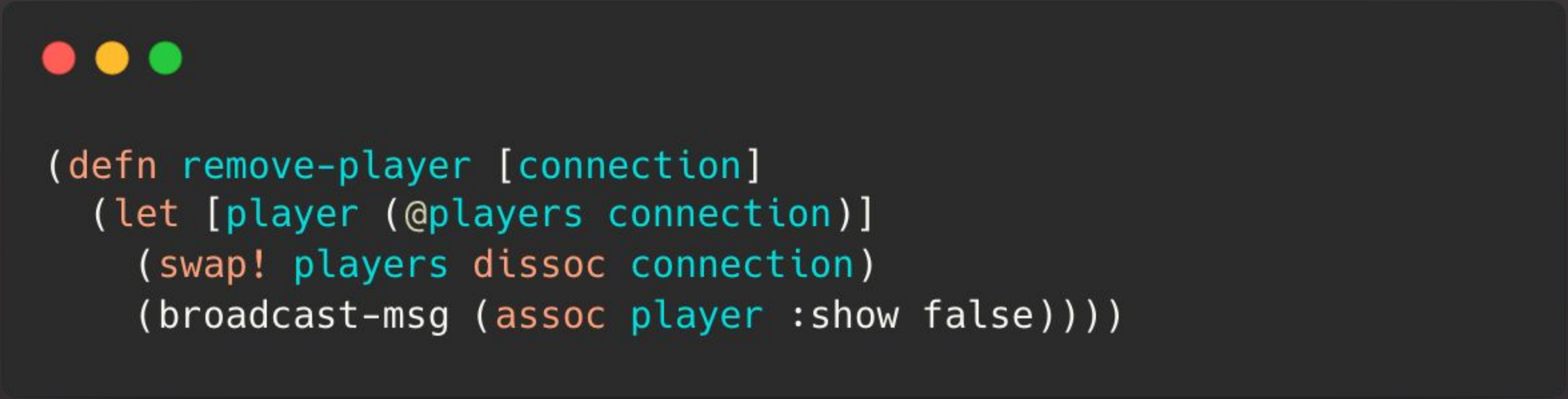
1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items

@Ladymey

Manage Game State

1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items

- Delete the player entity
- Update all players with the new game state



```clojure
(defn remove-player [connection]
  (let [player (@players connection)]
    (swap! players dissoc connection)
    (broadcast-msg (assoc player :show false))))
```

Manage Game State

1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items

- Generate collectables

@Ladymey

Manage Game State

1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items
   - Generate collectables
   - Detect collisions

@Ladymey

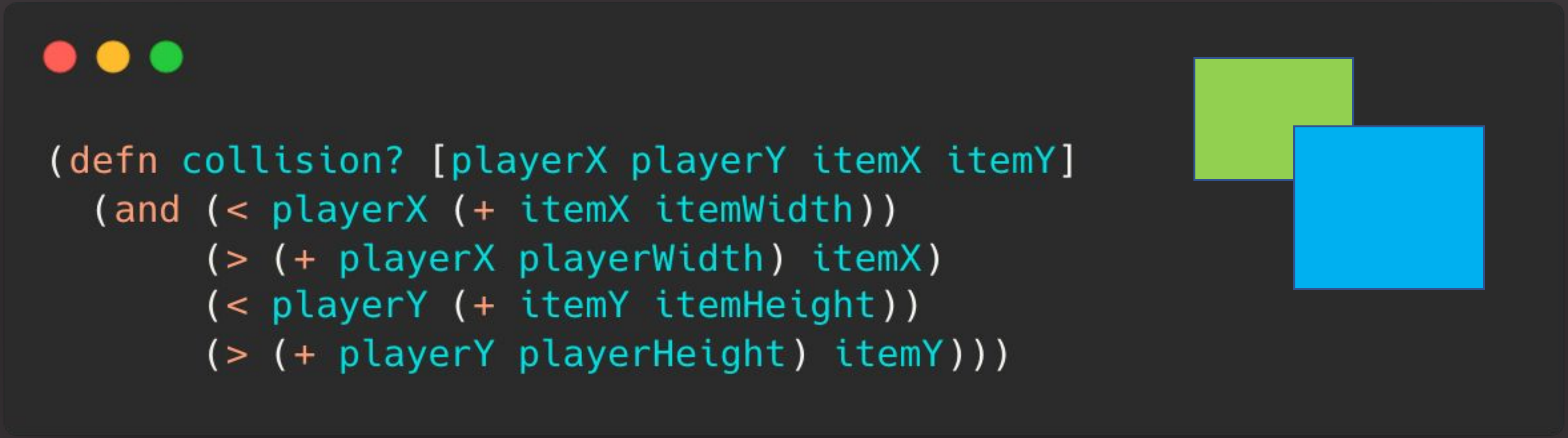
Manage Game State

1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items
   - Generate collectables
   - Detect collisions
   - Handle when collected:
     -- Update game state (players map, items map)
     -- Update all players with the new game state

@Ladymey

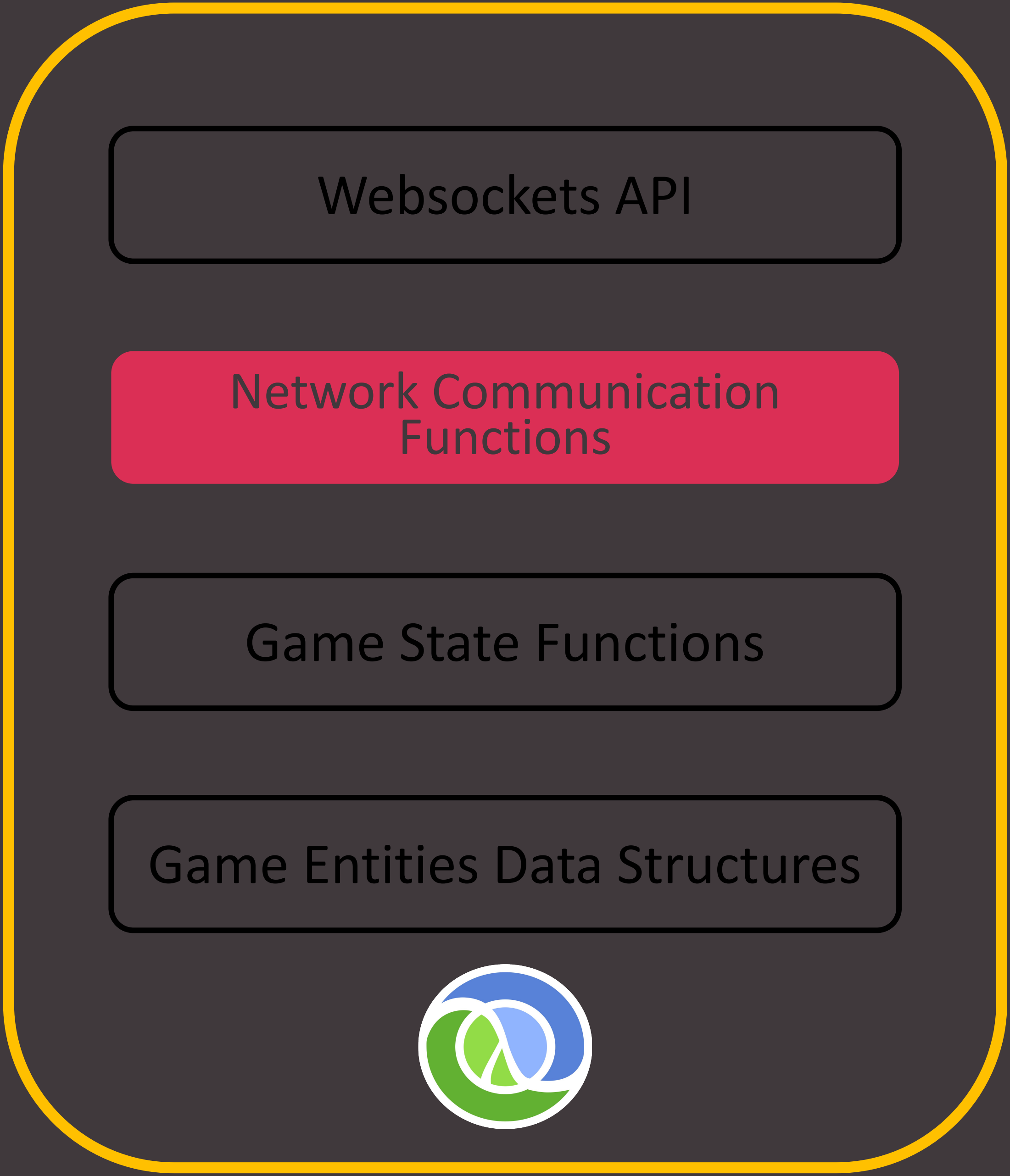
Manage Game State

1. Add new player
2. Player movement
3. Player left the game
4. Handle collectable items - Detect collisions



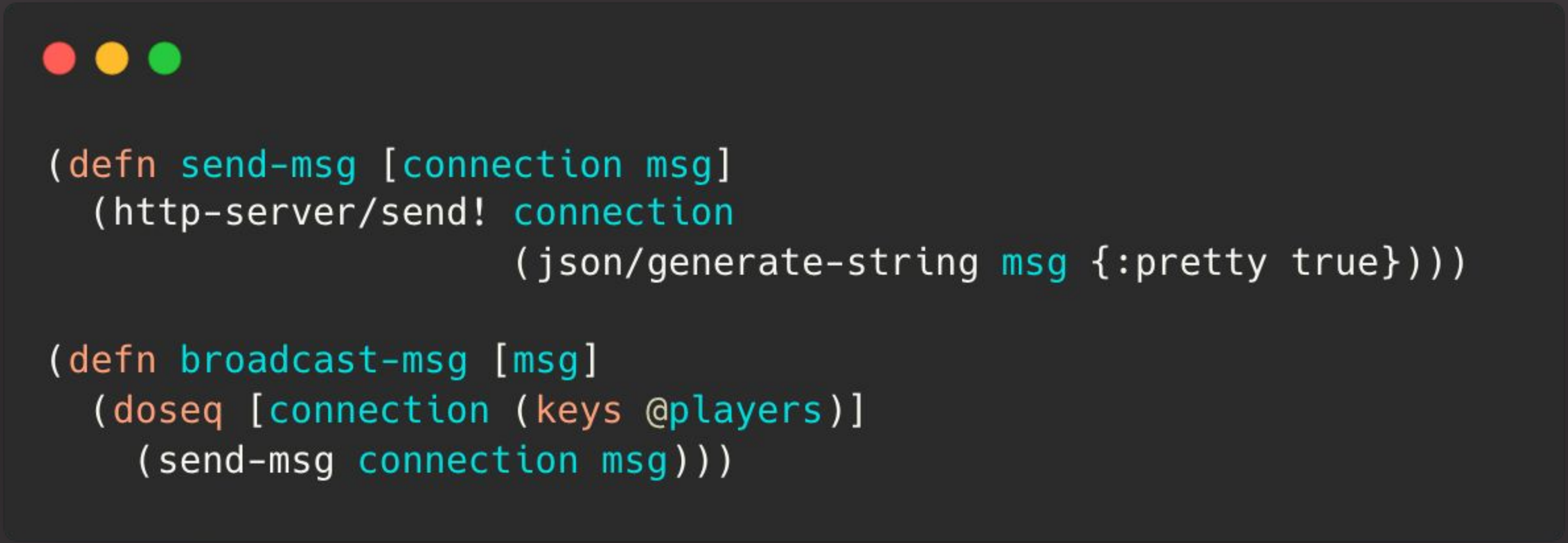
```
(defn collision? [playerX playerY itemX itemY]
  (and (< playerX (+ itemX itemWidth))
       (> (+ playerX playerWidth) itemX)
       (< playerY (+ itemY itemHeight))
       (> (+ playerY playerHeight) itemY)))
```

Server Architecture


Websockets API
Network Communication Functions
Game State Functions
Game Entities Data Structures

@Ladymey

Network Communication Functions



```clojure
(defn send-msg [connection msg]
  (http-server/send! connection
                     (json/generate-string msg {:pretty true})))

(defn broadcast-msg [msg]
  (doseq [connection (keys @players)]
    (send-msg connection msg)))
```

@Ladymey

Server Architecture

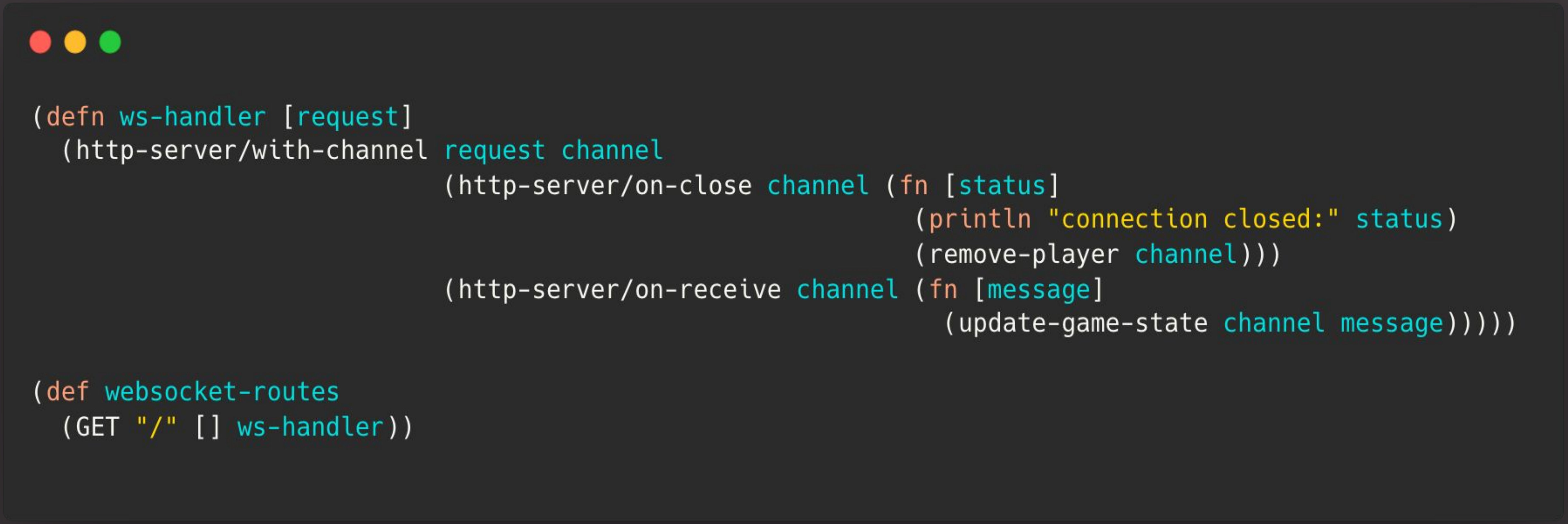
Websockets API

Network Communication Functions

Game State Functions

Game Entities Data Structures

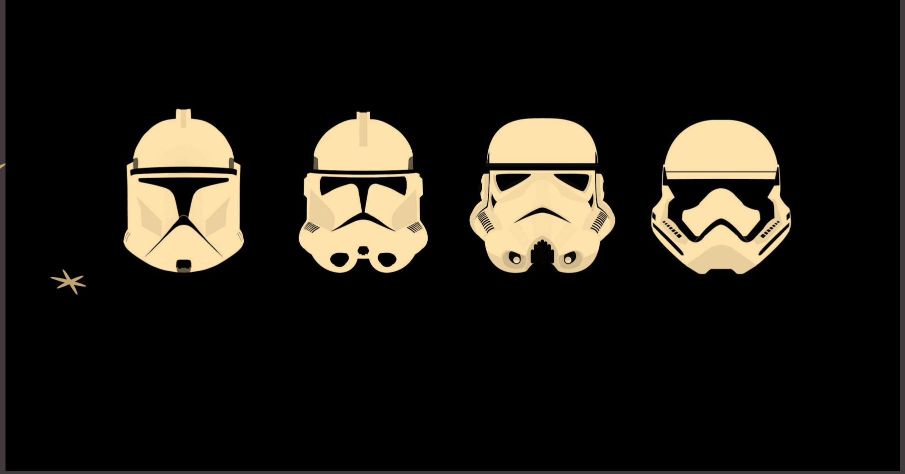
Websockets API



```clojure
(defn ws-handler [request]
  (http-server/with-channel request channel
                            (http-server/on-close channel (fn [status]
                                                            (println "connection closed:" status)
                                                            (remove-player channel)))
                            (http-server/on-receive channel (fn [message]
                                                              (update-game-state channel message)))))

(def websocket-routes
  (GET "/" [] ws-handler))
```

@Ladymey



Summary

@Ladymey

Server Architecture

Websockets API

Network Communication Functions

Game State Functions

Game Entities Data Structures

@Ladymey

Challenges

@Ladymey

Latency

@Ladymey

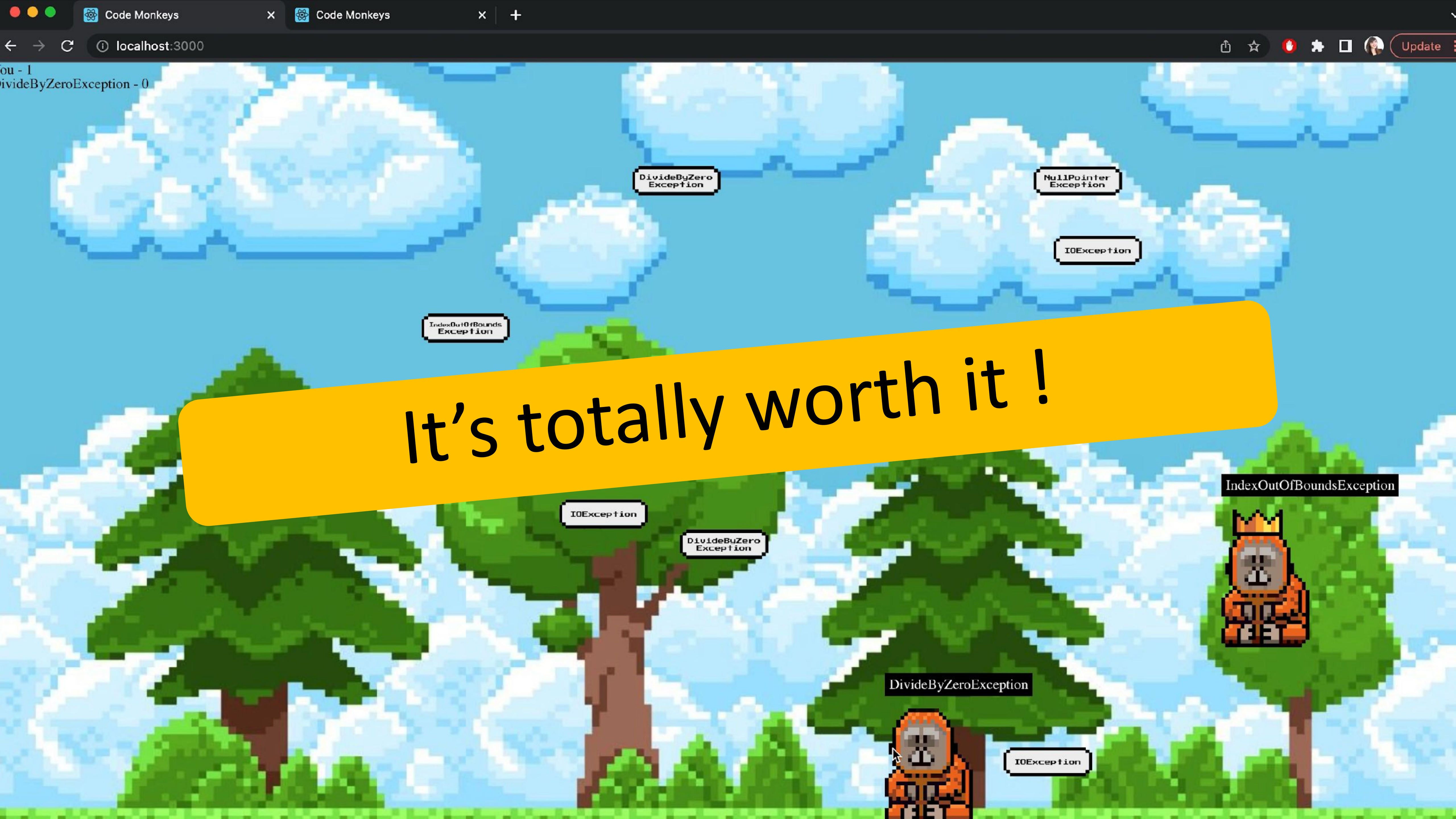
Scale

@Ladymey

Go develop a MOG!

@Ladymey

Although it won’t be Star Wars Battlefront II



It’s totally worth it !

Thank you!

@ladymeyy

@ladymey