Link “ http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms ”

**» Phòng MMO**

Kể từ **SmartFoxServer 2X phiên bản 2.8** , chúng tôi đã giới thiệu một loại đối tượng Phòng mới, **MMORoom** , hỗ trợ tương tác cục bộ giữa người dùng và đối tượng trò chơi dựa trên khoảng cách của họ.

**" Tổng quan**

MMORoom **mở** rộng chức năng của Phòng thông thường bằng cách thêm Khu vực quan tâm ( **viết tắt là AoI** ) để xác định phạm vi không gian của các sự kiện mà người dùng sẽ nhận được. Tham số **AoI** đại diện cho khu vực mà người dùng sẽ ảnh hưởng lẫn nhau, ví dụ như khi gửi tin nhắn công khai, cập nhật Biến người dùng, v.v.

Theo mặc định, **MMORoom** không kích hoạt các sự kiện phía máy khách USER\_ENTER hoặc USER\_EXIT thông thường **khi** người **dùng** vào hoặc rời khỏi phòng. Thay vào đó, danh sách người dùng của Phòng được cập nhật thông qua sự kiện **PROXIMITY\_LIST\_UPDATE** cung cấp vùng đồng bằng của danh sách người dùng hiện tại, trong AoI.

Nói cách khác, danh sách lân cận thay thế danh sách người dùng thông thường ở phía máy khách, tối ưu hóa số lượng cập nhật mà người dùng nhận được. Về phía máy chủ, toàn bộ danh sách người dùng của Phòng vẫn có thể truy cập được.

**" Mục đích sử dụng**

Theo đề xuất của tên lớp, các đối tượng **MMORoom** rất hữu ích để tạo các khu vực rất lớn, hầu như không giới hạn, có thể chứa hàng nghìn người chơi mà không làm quá tải máy khách với các bản cập nhật. MMORoom **có** thể được định cấu hình để điều tiết các sự kiện PROXIMITY\_LIST\_UPDATE nhằm tối ưu hóa lưu lượng mạng.

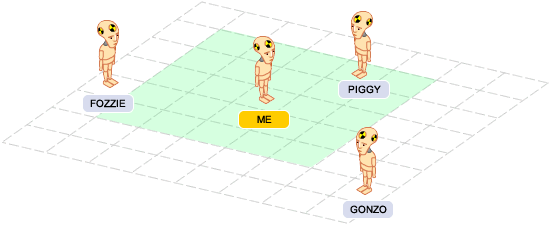
**» Tham gia và thiết lập vị trí Người dùng**

Trái ngược với các Phòng thông thường, **MMORoom** cần biết vị trí của từng người dùng trong không gian 2D hoặc 3D. Hệ tọa độ là trừu tượng và chung chung, cho phép các nhà phát triển sử dụng bất kỳ đơn vị đo lường nào (pixel, inch, mét, dặm, v.v.) được xác định bởi số nguyên 32 bit hoặc giá trị dấu phẩy động.

Khi người dùng tham gia **MMORoom** , vị trí của anh ta trên thế giới vẫn chưa được xác định và do đó anh ta sẽ ở trạng thái *lấp lửng* cho đến khi yêu cầu **SetUserPosition đầu tiên** được gửi. Để tránh người dùng dành quá nhiều thời gian ở trạng thái vô hình này, mỗi MMORoom có thể được định cấu hình để cho phép giá trị hết thời gian mà sau đó người dùng sẽ bị xóa khỏi Phòng (LƯU Ý: thời gian chờ chỉ áp dụng cho những người dùng chưa đặt vị trí ban đầu trong phòng).

Như đã đề cập trong phần tổng quan, không có sự kiện **USER\_ENTER** / **EXIT\_ROOM nào được** kích hoạt cho những người dùng khác như trong Phòng thông thường. Cách mà người chơi được cập nhật về những thay đổi của người dùng khác ở gần họ (nói cách khác là người dùng vào hoặc rời AoI) là thông qua sự kiện **PROXIMITY\_LIST\_UPDATE của** ứng dụng khách . Tất cả các sự kiện khác liên quan đến Phòng sẽ hoạt động giống như trong các Phòng thông thường, bao gồm cả **USER\_COUNT\_CHANGE** giúp người dùng trong cùng một Nhóm phòng được cập nhật về tổng số khách hàng trong mỗi Phòng.

Hãy xem một ví dụ trực quan:



Vùng màu xanh lục xung quanh người chơi của chúng tôi (được đánh dấu là "TÔI") đại diện cho AoI của anh ấy: tất cả người dùng trong khu vực đó sẽ có thể thấy người chơi của chúng tôi trao đổi tin nhắn/sự kiện với anh ấy. Ngoài ra, tất cả người chơi bên trong AoI đại diện cho danh sách người dùng của Phòng hiện tại theo quan điểm của khách hàng của chúng tôi. Trong trường hợp cụ thể của hình ảnh này, chúng ta sẽ chỉ thấy một người chơi khác, Piggy, trong đó "thấy" có nghĩa là nhận các sự kiện liên quan đến người chơi đó, hãy lưu ý về sự hiện diện của anh ta trong Phòng.

Tất nhiên, trong trò chơi, vị trí của người chơi sẽ thay đổi: mỗi khách hàng phải tiếp tục gửi yêu cầu **SetUserPosition** một cách thường xuyên để hệ thống biết được vị trí hiện tại của tất cả người dùng trong MMORoom và gửi sự kiện **PROXIMITY\_LIST\_UPDATE** tương ứng. Tốc độ đặt vị trí có thể khác nhau rất nhiều, tùy thuộc vào loại trò chơi. Kiểm tra các hướng dẫn được liên kết ở cuối trang để thảo luận sâu hơn về chủ đề này.

Giả sử rằng trong lần di chuyển tiếp theo, người dùng Fozzie sẽ vào AoI và người dùng Piggy sẽ rời khỏi đó: sự kiện **PROXIMITY\_LIST\_UPDATE tiếp theo** sẽ phản ánh tình huống mới này. Cụ thể, sự kiện cung cấp hai danh sách chứa tất cả người dùng đã tham gia AoI và tất cả những người đã rời khỏi AoI kể từ lần cập nhật trước. Từ góc độ API phía máy khách, đây là cách xử lý sự kiện **PROXIMITY\_LIST\_UPDATE** :

C# JS AS3

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **private** **void** OnProximityListUpdate(BaseEvent evt)  {      List<User> added = (List<User>) evt.**params**["addedUsers"];      List<User> removed = (List<User>) evt.**params**["removedUsers"];        // Add Users that have entered the proximity list  **foreach** (User user **in** added)      {          // Obtain the coordinates at which the User "appeared" in our range          Vec3D entryPoint = user.AoiEntryPoint;            // Add new avatar in the scene (PSEUDO CODE)          AvatarSprite avatarSprite = **new** AvatarSprite();          avatarSprite.x = entryPoint.px;          avatarSprite.y = entryPoint.py;          ...      }        // Remove Users that have left the proximity list  **foreach** (User user **in** removed)      {          // Remove the User avatar from the scene...      }  } |

Đoạn mã trên được trình bày dưới dạng một ví dụ chung sẽ hoạt động trong mọi loại máy khách (C#, AS3, Java, v.v.) cho dù chúng ta đang sử dụng thế giới 2D hay 3D. Hệ tọa độ **MMORoom** luôn hoạt động với hệ thống 3D (X, Y, Z) có thể được giảm xuống X và Y cho các ứng dụng hai chiều.

Mã này cho biết cách duyệt qua danh sách Người dùng **đã thêm** và **Người dùng đã xóa** để xử lý khía cạnh kết xuất của mọi thứ.

**» Biến người dùng và biến phòng**

**Biến người dùng** sẽ hoạt động bình thường trong MMORoom, chỉ ảnh hưởng đến những Người dùng nằm trong AoI của người gửi yêu cầu. Thay vào đó, **RoomVariables cần được sử dụng một cách tiết kiệm để tránh tạo ra lưu lượng truy cập lớn cho các máy khách.** Do **RoomVariables** chứa dữ liệu mà tất cả người dùng trong MMORoom quan tâm nên việc cập nhật chúng thường xuyên sẽ tạo ra các thông báo quảng bá rất lớn có nguy cơ làm bão hòa băng thông máy chủ. Điều quan trọng ở đây là tốc độ chúng được cập nhật nhiều hơn số lượng biến được sử dụng.

**» Giới hạn bản đồ**

Đối tượng **MMORoom** chấp nhận một cặp tham số **Vec3D** đại diện cho các giới hạn của bản đồ ảo trên ba trục (X, Y, Z). Trong ứng dụng 2D và 2.5D, nhà phát triển chỉ có thể sử dụng tọa độ X và Y để lại giá trị Z luôn được đặt thành 0. Bạn nên đặt các giới hạn này để hạn chế chuyển động của người dùng trong phạm vi thế giới ảo của bạn. Bằng cách này, các chuyển động bất hợp pháp có thể được phát hiện ở phía máy chủ và bị hệ thống từ chối.

**LƯU Ý**   
Nếu không đặt giới hạn bản đồ, có nguy cơ tiềm ẩn là người dùng độc hại có thể làm cạn kiệt bộ nhớ hệ thống bằng cách cố gắng lấp đầy một số lượng rất lớn khoảng trống cho các giá trị tọa độ cực trị.

**» Lấy điểm vào của Người dùng khác**

Việc biết người dùng đã nhập AoI của người chơi ở tọa độ nào thường rất hữu ích. Thông thường, mã máy khách sẽ cần biết vị trí này để hiển thị hình nhân vật/hình đại diện ở đúng vị trí. Theo mặc định, MMORoom luôn gửi vị trí mục nhập của mỗi Người dùng mới. Nếu một chút thông tin này không cần thiết trong ứng dụng của bạn, nó có thể được tắt đi để tiết kiệm thêm lưu lượng.

Cách sử dụng tính năng này được thể hiện trong ví dụ mã trước, bằng cách đọc giá trị **User.aoiEntryPosition** .

Điều quan trọng là phải hiểu rằng, như tên thuộc tính đã nói, đây là tọa độ của người dùng khi lần đầu tiên anh ta nhập AoI của người chơi, không phải tọa độ hiện tại. Nói cách khác, API SFS2X không tự động giữ các tọa độ đó đồng bộ với máy chủ. Nhà phát triển có trách nhiệm tìm ra chiến lược tốt nhất để đồng bộ hóa vị trí của người dùng trên máy khách tùy thuộc vào loại trò chơi, độ trễ giữa máy khách và máy chủ, v.v. Kiểm tra các hướng dẫn được liên kết ở cuối trang để thảo luận sâu hơn về chủ đề này.

**» Tạo một MMORoom**

MMORoom được tạo bằng cách gửi một **CreateRoomRequest thông thường** từ phía máy khách hoặc tương đương từ phía máy chủ (xem phương thức *SFSApi.createRoom* ). Sự khác biệt duy nhất là đối tượng cài đặt phải thuộc lớp **MMORoomSettings** . Hãy xem xét ví dụ mã bên dưới ( sfs là thể hiện của lớp *SmartFox* ):

C# JS AS3

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ...    // Add room-related event listeners during the SmartFox instance setup  sfs.AddEventListener(SFSEvent.ROOM\_JOIN, OnRoomJoin);    ... |

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | // Create the Room and autojoin it  MMORoomSettings cfg = **new** MMORoomSettings("New MMORoom");  cfg.defaultAOI = **new** Vec3D(30,25,12);  cfg.maxUsers = 5000;  cfg.maxSpectators = 0;  cfg.mapLimits = **new** MapLimits(**new** Vec3D(-3000, -500, -3000), **new** Vec3D(3000, 500, 3000));  cfg.userMaxLimboSeconds = 20;    sfs.Send(**new** Sfs2X.Requests.CreateRoomRequest(cfg, **true**)); |

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **private** **void** OnRoomJoin(BaseEvent evt)  {      Room room = (Room)evt.Params["room"];    **if** (room **is** MMORoom)      {          // This is an MMORoom so we need to set the Player position to become visible          // In a real case scenario we should find a free position on the virtual map            sfs.Send(**new** Sfs2X.Requests.MMO.SetUserPositionRequest(**new** Vec3D(100, 100, 100)));            // This in turn will trigger a PROXIMITY\_LIST\_UPDATE with the users present in our Player's AoI      }  } |

Trước tiên, chúng tôi tạo **MMORoomSettings** và định cấu hình một số hành vi của MMORoom. Cụ thể, chúng tôi đặt giới hạn vật lý của bản đồ thế giới ảo và giảm thuộc tính **userMaxLimboSeconds** xuống 20 giây (mặc định là 50).

Chúng tôi cũng đã đặt giới hạn giả định là 5000 người dùng cho bản đồ của mình. Giá trị này phần lớn có thể thay đổi tùy theo kích thước vật lý của bản đồ, kích thước của các nhân vật và thực thể do người chơi thể hiện và kích thước của các khu vực có thể khám phá trên bản đồ đó.

Như đã đề cập trước đó khi người chơi tham gia vào MMORoom, anh ta vẫn ẩn đối với những người dùng khác cho đến khi một vị trí thực tế cuối cùng được chỉ định. Để làm điều này, chúng tôi xử lý sự kiện ROOM\_JOIN và đảm bảo gửi ngay **SetUserPostionRequest** .

**» Xác định đúng điểm vào trên bản đồ**

Trong ví dụ trước, chúng tôi đã sử dụng một cách tiếp cận đơn giản để định vị người chơi trong bản đồ ảo, bằng cách sử dụng các giá trị được mã hóa cứng. Trong một tình huống thực tế, thường cần thêm một chút logic để tìm đúng vị trí.

Không thể sử dụng các khu vực không thể đi lại và các điểm đã có người chơi khác để sinh ra người chơi, do đó cần phải áp dụng một số logic trò chơi trước khi sinh ra người chơi.

Tùy thuộc vào cách trò chơi của chúng tôi hoạt động và nơi dữ liệu bản đồ được xử lý, chúng tôi sẽ cần áp dụng logic sinh sản như vậy ở phía máy khách hoặc máy chủ. Thông thường, đó sẽ là phía máy chủ, vì từ quan điểm đó, chúng tôi có thể kiểm soát toàn bộ bản đồ và vị trí của mọi người. Chẳng hạn, có thể có các cách tiếp cận khác nhau về cách chọn vị trí xuất hiện cho người chơi:

* sử dụng lại vị trí của người chơi trước đó (ví dụ: vị trí cuối cùng kể từ khi người đó rời đi lần trước);
* một vị trí ngẫu nhiên trong cùng một khu vực mà người chơi đã ở lần trước;
* một điểm sinh sản ngẫu nhiên trong danh sách các khu vực dịch chuyển có sẵn trên bản đồ;
* chọn ngẫu nhiên bất kỳ địa điểm miễn phí nào;
* vân vân.

Trong mỗi trường hợp, rất có thể dữ liệu cần thiết để tính toán vị trí xuất hiện nằm ở phía máy chủ và do đó sẽ thuận tiện hơn khi xử lý giai đoạn này trong Mã mở rộng. Chúng tôi đề xuất hai cách khả thi để thực hiện việc này:

* **Tham gia từ phía máy chủ**Máy khách gửi yêu cầu đến Tiện ích mở rộng để tham gia MMORoom, máy chủ thực hiện yêu cầu tham gia, tính toán điểm vào và gọi phương **[thức SFSAPI.setUserPosition(...)](http://docs2x.smartfoxserver.com/api-docs/javadoc/server/com/smartfoxserver/v2/api/SFSApi.html" \t "_blank)** ngay lập tức.
* **Tham gia từ máy khách, đặt vị trí từ máy chủ**   
  Máy khách gửi một JoinRoomRequest thông thường và xử lý phản hồi tương đối. Tiện ích mở rộng máy chủ cũng đang lắng nghe sự kiện ROOM\_JOIN và khi nó kích hoạt logic điểm đầu vào được thực thi và lệnh **[gọi SFSAPI.setUserPosition(...)](http://docs2x.smartfoxserver.com/api-docs/javadoc/server/com/smartfoxserver/v2/api/SFSApi.html" \t "_blank)** được thực hiện.

Trong cả hai trường hợp, khách hàng sẽ nhận được sự kiện **PROXIMITY\_LIST\_UPDATE** với danh sách người dùng cục bộ.

**» Tiện ích mở rộng và AoI của người dùng**

Từ mã phía máy chủ, cũng có thể tận dụng tính năng của MMORoom và gửi các sự kiện tùy chỉnh chỉ ảnh hưởng đến một số Người dùng trong một AoI nhất định.

Để cho thấy cách thức hoạt động của tính năng này, giả sử chúng ta muốn gửi một sự kiện tùy chỉnh tới tất cả người chơi nằm trong AoI của người dùng Piggy.

Java

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | MMORoom mmoRoom = (MMORoom) getParentZone().getRoomByName("VirtualMMO");  User piggy = mmoRoom.getUserByName("Piggy");  List<User> recipients = mmoRoom.getProximityList(piggy);    // Make sure we have 1 or more recipients  **if** (recipients.size > 0)  {      ISFSObject message = **new** SFSObject();        // ...populate the SFSObject with custom data...        // Send the object to the selected users      send("customMessage", message, recipients);  } |

Cũng có thể cung cấp **AoI tùy chỉnh** để nhắm mục tiêu nâng cao hơn. Ví dụ: người dùng Piggy có thể thực hiện một hành động ảnh hưởng đến một nhóm nhỏ người dùng ở gần cô ấy hơn, liên quan đến tổng số người dùng có trong AoI mặc định xung quanh cô ấy. Đảm bảo tham khảo javadoc phía máy chủ để biết thêm chi tiết. Tất cả các liên kết được tìm thấy ở cuối bài viết này.

**» MMOItem và Biến MMOItem**

MMORoom **cũng** cung cấp hỗ trợ cho một thực thể mới gọi là **MMOItem** đại diện cho một thực thể không phải người chơi bên trong bản đồ. MMOItems có thể được sử dụng làm phần thưởng, trình kích hoạt, đạn, v.v. hoặc bất kỳ đối tượng không phải người chơi nào khác sẽ được xử lý bằng cách sử dụng quy tắc hiển thị (AoI) của MMORoom.

Điều này có nghĩa là bất cứ khi nào một hoặc nhiều **MMOItem** nằm trong AoI của Người chơi, nó sẽ được thông báo cho Người dùng bằng sự kiện PROXIMITY\_LIST\_UPDATE. Đây là danh sách đầy đủ các tham số được cung cấp trong sự kiện:

|  |  |  |
| --- | --- | --- |
| **Tên** | **loại** | **sự miêu tả** |
| phòng | MMORoom | mục tiêu MMORoom |
| người dùng đã thêm | Lập danh sách | danh sách những người chơi mới có thể nhìn thấy |
| đã xóaNgười dùng | Lập danh sách | danh sách người chơi bị xóa |
| thêm mục | Lập danh sách | danh sách các MMOitem mới có thể nhìn thấy |
| đã loại bỏ | Lập danh sách | danh sách các MMOItem đã bị xóa |

Mỗi **MMOItem** được xác định bằng ID duy nhất và, tùy chọn, bằng một số biến tùy chỉnh được gọi là **Biến MMOItem** , hoạt động chính xác như Biến người dùng (lớp MMOItemVariable mở rộng Biến người dùng).

**LƯU Ý**   
Có một điểm khác biệt quan trọng giữa Biến người dùng và Biến MMOItem: biến thứ hai chỉ có thể được xác định và cập nhật **từ phía máy chủ** .

**» Tạo MMOItem từ phía máy chủ**

Chúng ta hãy xem ví dụ phía máy chủ đơn giản này:

Java

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **private** **void** createMMOItem()  {      // Reference to the MMOApi object      SFSMMOApi mmoApi = SmartFoxServer.getInstance().getAPIManager().getMMOApi();        // Reference to the game's MMORoom      Room targetRoom = getParentZone().getRoomByName("My MMO Room");        // Prepare a list of variables for the MMOItem      List<IMMOItemVariables> variables = **new** LinkedList<IMMOItemVariable>();      variables.add( **new** MMOItemVariable("type", "bonus") );      variables.add( **new** MMOItemVariable("points", 250) );      variables.add( **new** MMOItemVariable("active", **true**) );        // Create the MMOItem      MMOItem mmoItem = **new** MMOItem(variables);        // Deploy the MMOItem in the MMORoom's map      mmoApi.setMMOItemPosition(mmoItem, **new** Vec3D(50, 40, 0), targetRoom);  } |

Để triển khai một MMOItem mới bên trong Phòng của trò chơi, chúng ta cần ba thứ:

* một tham chiếu đến trò chơi MMORoom mà chúng ta có được thông qua Vùng của Tiện ích mở rộng thông qua phương thức getRoomByName();
* một danh sách các biến sẽ thêm các thuộc tính có ý nghĩa vào chính MMOItem; trong trường hợp này, chúng tôi đang xác định một đối tượng thưởng có giá trị điểm nhất định và một cờ để biết liệu nó có hoạt động trên thế giới hay không;
* chính đối tượng MMOItem được tạo cùng với danh sách các biến.

Cuối cùng, chúng tôi gọi **setMMOItemPosition(...)** để triển khai mục sẽ xuất hiện trong PROXIMITY\_LIST\_UPDATE tiếp theo trên máy khách bên trong thuộc tính **addedItems** của sự kiện.

**» Quản lý MMOItem trong game**

Tùy thuộc vào loại trò chơi mà chúng tôi đang phát triển và kích thước của bản đồ ảo, chúng tôi có thể tạo ra hàng trăm MMOItem để mô tả phần thưởng, trình kích hoạt, đạn và các thực thể tương tự trong trò chơi.

MMORoom **quản** lý các mục này bằng cách theo dõi tất cả các MMOItem đã được thêm vào. Nếu chúng ta chỉ cần truy xuất một mục từ id duy nhất của nó, chúng ta có thể sử dụng phương **thức MMORoom.getMMOItemById(...)** nếu không, chúng ta sẽ tốt hơn bằng cách sử dụng các danh sách riêng biệt cho từng loại đối tượng mà chúng ta đang quản lý trong trò chơi. Ví dụ: một danh sách cho dấu đầu dòng, một danh sách cho trình kích hoạt, v.v.

Một giải pháp phổ biến để xử lý một số lớp MMOItem trong trò chơi là mở rộng lớp MMOItem bằng cách thêm các thuộc tính cục bộ (ví dụ: chỉ phía máy chủ) mà chúng tôi cần:

Java

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | **public** **class** LaserBeamItem **extends** MMOItem  {  **public** **static** **final** String TYPE\_RED = "red";  **public** **static** **final** String TYPE\_YELLOW = "yellow";  **public** **static** **final** String TYPE\_GREEN = "green";    **private** String type;  **private** **int** strength;    **public** LaserBeamItem(String type)      {  **super**();  **this**.type = type;    **if** (type.equals(TYPE\_RED))  **this**.strength = 100;  **else** **if** (type.equals(TYPE\_YELLOW))  **this**.strength = 50;  **else** **if** (type.equals(TYPE\_GREEN))  **this**.strength = 25;      }    **public** String getType()      {  **return** **this**.type;      }    **public** **int** getStrength()      {  **return** **this**.strength;      }        // ...  } |

Bằng cách này, chúng tôi có thể thuận tiện thêm các thuộc tính và trạng thái phía máy chủ của riêng mình cho từng MMOItem trong khi xử lý các thuộc tính công khai thông qua MMOItemVariables.

Nếu MMOItems được sử dụng cho đạn hoặc các đối tượng tương tự có **vòng đời giới hạn trong trò chơi** , chúng tôi cũng phải đảm bảo xóa chúng khỏi Phòng khi chúng không còn được sử dụng. Để làm điều này, chúng ta có thể gọi phương **thức MMOApi.removeMMOItem(...)** , phương thức này sẽ cập nhật tất cả các máy khách bị ảnh hưởng thông qua PROXIMITY\_LIST\_UPDATE thông thường.

**» Tinh chỉnh hiệu năng MMORoom**

Có thể đạt được một số tối ưu hóa bằng cách thử nghiệm cài đặt MMORoom.

**» Cập nhật tốc độ**

Cài đặt tiệm **cậnListUpdateMillis trong lớp MMORoomSettings** cho phép quyết định tốc độ mà các sự kiện PROXIMITY\_LIST\_UPDATE sẽ được kích hoạt từ phía máy chủ. Tối ưu hóa khía cạnh này là rất quan trọng để có được hiệu suất máy chủ và máy khách phù hợp.

Giá trị mặc định được đặt thành **250ms.** có vẻ hơi "chậm". Trên thực tế, cài đặt này đã rất hào phóng và chúng tôi có thể cần tăng giá trị này tùy thuộc vào lượng lưu lượng truy cập đang chạy trong máy chủ.

Lý do tại sao các sự kiện PROXIMITY\_LIST\_UPDATE không được gửi trong thời gian thực là vì trong 250 mili giây, có khả năng bất kỳ hình đại diện nào trên màn hình sẽ di chuyển rất ít so với vị trí trước đó, rất ít đến mức không thể nhận thấy. Với suy nghĩ này, chúng tôi có thể giảm số lượng sự kiện được gửi tới mọi khách hàng bằng cách điều chỉnh chúng bằng **tham** số distanceListUpdateMillis.

Trong trường hợp mặc định, máy chủ sẽ đợi 250 mili giây. trước khi gửi thông báo rằng người dùng đã vào hoặc rời khỏi AoI của người chơi. Nếu các sự kiện khác cùng loại xảy ra trong thời gian trễ ngắn, chúng sẽ được tổng hợp và gửi cùng một lúc, giúp tiết kiệm lưu lượng. Trong một MMORoom rất bận rộn với hàng trăm hoặc hàng nghìn người chơi, điều này có thể tạo ra sự khác biệt lớn về mặt sử dụng mạng.

**H:** Không phải sự chậm trễ sẽ làm chậm hành động của trò chơi hay thế giới MMO sao?

Không, vì PROXIMITY\_LIST\_UPDATE không liên quan gì đến cách chúng tôi quản lý chuyển động vật lý của người chơi trên màn hình. Phần này tùy thuộc vào logic trò chơi của chúng tôi và MMORoom sẽ không cản trở. MMORoom sẽ chỉ phát tín hiệu khi người dùng vào hoặc rời AoI của người chơi. Một độ trễ nhỏ vài trăm mili giây sẽ không tạo ra bất kỳ sự khác biệt trực quan nào nhưng nó sẽ tiết kiệm đáng kể băng thông.

**H: Cài đặt tiệm cậnListUpdateMillis** được đề xuất là gì?

Đối với **loại tương tác trỏ và nhấp** (thế giới ảo MMO, v.v.), khoảng thời gian cập nhật có thể được đặt trong khoảng 250-1000 mili giây. dựa trên kích thước khu vực kết xuất và tốc độ tương tác tổng thể của chúng tôi. Chúng tôi khuyên bạn nên bắt đầu với giá trị mặc định và sau đó thử nghiệm bằng cách tăng giá trị theo gia số 100 mili giây. Chúng ta càng có thể tăng khoảng thời gian mà không gây ra độ trễ rõ ràng trong máy khách thì điều đó càng tốt cho việc tối ưu hóa băng thông.

Đối với **các trò chơi hành động/thời gian thực** , cài đặt mặc định (250 mili giây) có thể ổn và chúng tôi khuyên bạn nên áp dụng phương pháp tinh chỉnh trước đó, chỉ bắt đầu từ giá trị thấp hơn, ví dụ: 50 mili giây và tăng dần để kiểm tra xem nó có tạo ra bất kỳ "tạo tác" nào trong trò chơi. Theo hiện vật, chúng tôi muốn nói đến độ trễ có thể nhìn thấy khi xuất hiện các ký tự/thực thể trên màn hình.

**» Tối ưu hóa kích thước cập nhật**

Trong MMORooms, chúng ta sẽ thấy mình sử dụng khá nhiều Biến người dùng và Biến MMOItem để xác định các thuộc tính tùy chỉnh cho từng người chơi và vật phẩm trên bản đồ ảo.

Để đạt được hiệu suất tốt nhất từ quan điểm băng thông/mạng, chúng tôi khuyên bạn nên tối ưu hóa Biến bằng cách sử dụng tên rất ngắn và giá trị ngắn gọn nhất có thể. Ví dụ thay vì ví dụ chúng tôi đã cung cấp trong phần trước:

Java

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4 | List<IMMOItemVariables> variables = **new** LinkedList<IMMOItemVariable>();  variables.add(**new** MMOItemVariable("type", "bonus"));  variables.add(**new** MMOItemVariable("points", 250));  variables.add(**new** MMOItemVariable("active", **true**)); |

we suggest to trim down the variable names as much as possible:

Java

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/mmo-rooms)

|  |  |
| --- | --- |
| 1  2  3  4 | List<IMMOItemVariables> variables = **new** LinkedList<IMMOItemVariable>();  variables.add(**new** MMOItemVariable("t", 0));  variables.add(**new** MMOItemVariable("p", 250));  variables.add(**new** MMOItemVariable("a", **true**)); |

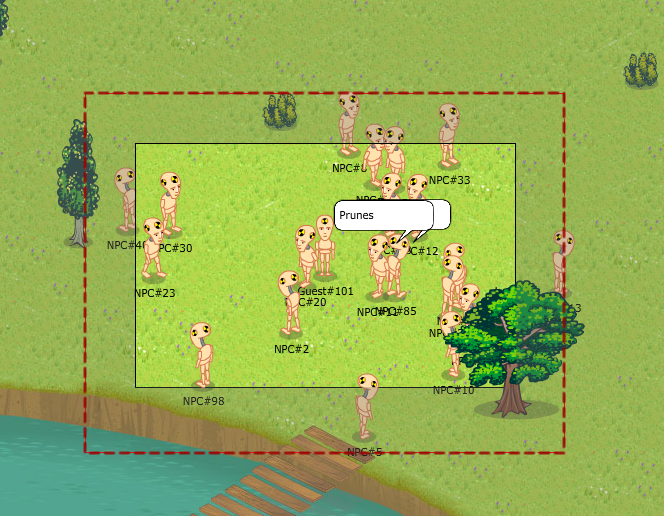
Lưu ý rằng chúng tôi cũng đã thay thế biến đầu tiên bằng một số thay vì một chuỗi, để giảm bớt mọi thứ càng nhiều càng tốt. Quy tắc chung là mỗi khi chúng ta sử dụng một chuỗi làm bộ mô tả cho một biến nào đó, chúng ta nên xem xét liệu có thể sử dụng một số hay không, số này sau đó sẽ được "giải mã" thành giá trị tương ứng của nó thông qua bảng khóa-giá trị.

Một ví dụ về vấn đề này là khi chúng ta cần chỉ định loại Hình đại diện cho từng Người dùng trong Biến của họ. Thay vì sử dụng một giá trị chuỗi chẳng hạn như "Warrior", "Wizard", "Thief", v.v., chúng ta có thể sử dụng một id số đơn giản như 0, 1, 2... Điều này sẽ giúp chúng ta tiết kiệm thêm băng thông, đặc biệt khi có hàng nghìn người chơi sẽ chơi cùng nhau.

**» Khu vực kết xuất so với AoI**

Đặc biệt trong thế giới ảo 2D và 2.5D, điều quan trọng là phải chọn kích thước phù hợp cho AoI dựa trên kích thước của khung nhìn trên màn hình. Sử dụng AoI lớn hơn một chút so với khung nhìn sẽ cho phép "ẩn" sự xuất hiện/biến mất đột ngột của các họa tiết trên màn hình một cách rất mượt mà.

Ảnh chụp màn hình này minh họa kỹ thuật:



Hình chữ nhật màu đỏ nét đứt bên ngoài hiển thị AoI thực tế trong khi hình chữ nhật bên trong là khu vực hiển thị. Bằng cách sử dụng chiến lược này, các hình đại diện sẽ vào và rời khỏi khung nhìn một cách liền mạch.

Để hiểu rõ hơn cách thức hoạt động của tính năng này, chúng tôi khuyên bạn nên xem **[hướng dẫn ví dụ MMO đơn giản](http://docs2x.smartfoxserver.com/ExamplesFlash/simple-mmo)** được cung cấp trong phần Ví dụ Flash.

**" Đọc thêm**

Chúng tôi đã triển khai nhiều khái niệm được nêu ở đây trong một ví dụ thực tế với các nguồn:

* học hỏi từ một ví dụ thực tế trong **[hướng dẫn ví dụ về MMO đơn giản của chúng tôi](http://docs2x.smartfoxserver.com/ExamplesFlash/simple-mmo)**
* nghiên cứu các **[kỹ thuật API MMO nâng cao](http://docs2x.smartfoxserver.com/AdvancedTopics/advanced-mmo-api)**
* đồng thời kiểm tra **[javadocs phía máy chủ](http://docs2x.smartfoxserver.com/AdvancedTopics/docs2x.smartfoxserver.com/api-docs/javadoc/server" \t "_blank)** để biết gói *com.smartfoxserver.v2.mmo*

The outer dashed red rectangle shows the actual AoI while the inner rectangle is the visible area. By using this strategy avatars will enter and leave the viewport in a seamless way.