**High-level multiplayer**

1. **API cấp cao so với cấp thấp**

Phần này giải thích sự khác biệt của mạng cấp cao và cấp thấp trong Godot cũng như một số nguyên tắc cơ bản.

Godot luôn hỗ trợ mạng cấp thấp tiêu chuẩn thông qua UDP, TCP và một số giao thức cấp cao hơn như SSL và HTTP. Các giao thức này rất linh hoạt và có thể được sử dụng cho hầu hết mọi thứ. Tuy nhiên, việc sử dụng chúng để đồng bộ hóa trạng thái trò chơi theo cách thủ công có thể là một khối lượng lớn công việc. Đôi khi công việc đó không thể tránh khỏi hoặc không đáng có, chẳng hạn như khi làm việc với triển khai máy chủ tùy chỉnh trên phụ trợ. Nhưng trong hầu hết các trường hợp, cần xem xét API mạng cấp cao của Godot, API này hy sinh một số quyền kiểm soát chi tiết của mạng cấp thấp để dễ sử dụng hơn.

Điều này là do những hạn chế cố hữu của các giao thức cấp thấp:

* TCP đảm bảo các gói sẽ luôn đến một cách đáng tin cậy và theo thứ tự, nhưng độ trễ thường cao hơn do sửa lỗi. Đây cũng là một giao thức khá phức tạp vì nó hiểu "kết nối" là gì và tối ưu hóa cho các mục tiêu thường không phù hợp với các ứng dụng như trò chơi nhiều người chơi. Các gói được lưu vào bộ đệm để được gửi theo lô lớn hơn, giao dịch chi phí cho mỗi gói ít hơn để có độ trễ cao hơn. Điều này có thể hữu ích cho những thứ như HTTP, nhưng thường không hữu ích cho trò chơi. Một số điều này có thể được cấu hình và vô hiệu hóa (ví dụ: bằng cách tắt "thuật toán của Nagle" cho kết nối TCP).
* UDP là một giao thức đơn giản hơn, chỉ gửi các gói (và không có khái niệm "kết nối"). Không có sửa lỗi làm cho nó khá nhanh (độ trễ thấp), nhưng các gói có thể bị mất trên đường đi hoặc nhận được không đúng thứ tự. Thêm vào đó, MTU (kích thước gói tối đa) cho UDP thường thấp (chỉ vài trăm byte), vì vậy việc truyền các gói lớn hơn có nghĩa là chia nhỏ chúng, tổ chức lại chúng và thử lại nếu một phần bị lỗi.

Nói chung, TCP có thể được coi là đáng tin cậy, có trật tự và chậm; UDP là không đáng tin cậy, không có thứ tự và nhanh chóng. Do có sự khác biệt lớn về hiệu suất, nên việc xây dựng lại các phần của TCP muốn có cho trò chơi (độ tin cậy tùy chọn và thứ tự gói) thường có ý nghĩa, đồng thời tránh các phần không mong muốn (tính năng điều khiển tắc nghẽn / lưu lượng, thuật toán của Nagle, v.v.). Do đó, hầu hết các game engine đều có cách triển khai như vậy và Godot cũng không ngoại lệ.

Tóm lại, ta có thể sử dụng API mạng cấp thấp để kiểm soát tối đa và triển khai mọi thứ trên các giao thức mạng trần hoặc sử dụng API cấp cao dựa trên SceneTree thực hiện hầu hết các công việc nặng nề đằng sau hậu trường theo cách nói chung được tối ưu hóa.

1. **Mức độ trừu tượng trung bình**

Godot sử dụng một đối tượng cấp trung NetworkedMultiplayerPeer . Đối tượng này không có nghĩa là được tạo trực tiếp, nhưng được thiết kế để một số triển khai C ++ có thể cung cấp nó.

Đối tượng này mở rộng từ PacketPeer , vì vậy nó kế thừa tất cả các phương thức hữu ích để tuần tự hóa, gửi và nhận dữ liệu. Ngoài ra, nó còn bổ sung thêm các phương pháp để thiết lập chế độ chuyển, ngang hàng, v.v. Nó cũng bao gồm các tín hiệu cho bạn biết khi nào đồng nghiệp kết nối hoặc ngắt kết nối.

Giao diện lớp này có thể tóm tắt hầu hết các loại lớp mạng, cấu trúc liên kết và thư viện. Theo mặc định, Godot cung cấp một triển khai dựa trên Enet ( NetworkedMultiplayerEnet), một dựa trên WebRTC ( WebRTCMultiplayer ) và một dựa trên WebSocket ( WebSocketMultiplayerPeer ), nhưng điều này có thể được sử dụng để triển khai các API di động (cho WiFi đặc biệt, Bluetooth) hoặc tùy chỉnh API mạng dành riêng cho thiết bị / bảng điều khiển.

Đối với hầu hết các trường hợp phổ biến, việc sử dụng đối tượng này trực tiếp không được khuyến khích, vì Godot cung cấp các phương tiện mạng cấp cao hơn. Tuy nhiên, nó được cung cấp trong trường hợp một trò chơi có nhu cầu cụ thể về API cấp thấp hơn.

1. **Khởi tạo mạng**

Đối tượng điều khiển mạng trong Godot cũng giống như đối tượng điều khiển mọi thứ liên quan đến cây: SceneTree.

Để khởi tạo mạng cấp cao, SceneTree phải được cung cấp một đối tượng NetworkedMultiplayerPeer. Để tạo đối tượng đó, trước tiên nó phải được khởi tạo dưới dạng máy chủ hoặc máy khách. Khởi tạo như một máy chủ với số lượng tối đa các đối tượng ngang hàng nhất định:

**var** peer = NetworkedMultiplayerENet.new()

peer.create\_server(SERVER\_PORT, MAX\_PLAYERS)

get\_tree().network\_peer = peer

Khởi tạo dưới dạng máy khách, kết nối với một IP và cổng nhất định:

**var** peer = NetworkedMultiplayerENet.new()

peer.create\_client(SERVER\_IP, SERVER\_PORT)

get\_tree().network\_peer = peer

Nhận mạng ngang hàng đã đặt trước đó:

get\_tree().get\_network\_peer()

Kiểm tra xem cây được khởi tạo dưới dạng máy chủ hay máy khách:

get\_tree().is\_network\_server()

Kết thúc tính năng kết nối mạng:

get\_tree().network\_peer = null

1. **Quản lý kết nối**

Một số trò chơi chấp nhận kết nối bất kỳ lúc nào, một số trò chơi khác trong giai đoạn vận động hành lang. Godot có thể được yêu cầu không còn chấp nhận các kết nối tại bất kỳ thời điểm nào. Để quản lý người kết nối, Godot cung cấp các tín hiệu sau trong SceneTree:

Máy chủ và Máy khách:

* network\_peer\_connected(int id)
* network\_peer\_disconnected(int id)

Các tín hiệu trên được gọi trên mọi đồng đẳng được kết nối với máy chủ (bao gồm cả trên máy chủ) khi một đồng đẳng mới kết nối hoặc ngắt kết nối. Máy khách sẽ kết nối với một ID duy nhất lớn hơn 1, trong khi ID mạng ngang hàng 1 luôn là máy chủ. Bất kỳ điều gì dưới 1 sẽ được xử lý là không hợp lệ. Ta có thể truy xuất ID cho hệ thống cục bộ thông qua SceneTree.get\_network\_unique\_id () . Các ID này chủ yếu sẽ hữu ích cho việc quản lý tiền sảnh và thường được lưu trữ, vì chúng xác định các máy khách được kết nối và do đó là người chơi. Người dùng cũng có thể sử dụng ID để chỉ gửi tin nhắn cho một số người chơi nhất định.

Máy khách:

* connected\_to\_server
* connection\_failed
* server\_disconnected

Một lần nữa, tất cả các chức năng này chủ yếu hữu ích cho việc quản lý tiền sảnh hoặc thêm / xóa người chơi khi đang di chuyển. Đối với những nhiệm vụ này, máy chủ rõ ràng phải hoạt động như một máy chủ và người dùng phải thực hiện các tác vụ theo cách thủ công như gửi thông tin người chơi mới được kết nối về những người chơi khác đã được kết nối (ví dụ: tên, số liệu thống kê của họ, v.v.).

1. **RPC**

Để giao tiếp giữa các đồng nghiệp, cách dễ nhất là sử dụng RPC (lệnh gọi thủ tục từ xa). Điều này được thực hiện như một tập hợp các chức năng trong Node :

* rpc("function\_name", <optional\_args>)
* rpc\_id(<peer\_id>,"function\_name", <optional\_args>)
* rpc\_unreliable("function\_name", <optional\_args>)
* rpc\_unreliable\_id(<peer\_id>, "function\_name", <optional\_args>)

Đồng bộ hóa các biến thành viên cũng có thể:

* rset("variable", value)
* rset\_id(<peer\_id>, "variable", value)
* rset\_unreliable("variable", value)
* rset\_unreliable\_id(<peer\_id>, "variable", value)

Các hàm có thể được gọi theo hai kiểu:

* Đáng tin cậy: lệnh gọi hàm sẽ đến bất kể điều gì, nhưng có thể mất nhiều thời gian hơn vì nó sẽ được truyền lại trong trường hợp không thành công.
* Không đáng tin cậy: nếu lệnh gọi hàm không đến, nó sẽ không được truyền lại; nhưng nếu nó đến, nó sẽ làm điều đó một cách nhanh chóng.

Trong hầu hết các trường hợp ta dùng mong muốn đáng tin cậy. Không đáng tin cậy chủ yếu hữu ích khi đồng bộ hóa các vị trí đối tượng (đồng bộ hóa phải diễn ra liên tục và nếu một gói bị mất, nó không tệ lắm vì một gói mới cuối cùng sẽ đến và nó có thể đã lỗi thời vì đối tượng di chuyển xa hơn trong thời gian chờ đợi, ngay cả khi nó đáng tin cậy).

Ngoài ra còn có get\_rpc\_sender\_idchức năng trong SceneTree, có thể được sử dụng để kiểm tra xem đồng đẳng nào (hoặc ID ngang hàng) đã gửi một RPC.

1. **Quay lại sảnh**

Hãy quay trở lại sảnh. Hãy tưởng tượng rằng mỗi người chơi kết nối với máy chủ sẽ cho mọi người biết về điều đó.

*# Typical lobby implementation; imagine this being in /root/lobby.*

**extends** Node

*# Connect all functions*

**func** \_ready():

get\_tree().connect("network\_peer\_connected", self, "\_player\_connected")

get\_tree().connect("network\_peer\_disconnected", self, "\_player\_disconnected")

get\_tree().connect("connected\_to\_server", self, "\_connected\_ok")

get\_tree().connect("connection\_failed", self, "\_connected\_fail")

get\_tree().connect("server\_disconnected", self, "\_server\_disconnected")

*# Player info, associate ID to data*

**var** player\_info = {}

*# Info we send to other players*

**var** my\_info = { name = "Johnson Magenta", favorite\_color = Color8(255, 0, 255) }

**func** \_player\_connected(id):

*# Called on both clients and server when a peer connects. Send my info to it.*

rpc\_id(id, "register\_player", my\_info)

**func** \_player\_disconnected(id):

player\_info.erase(id) *# Erase player from info.*

**func** \_connected\_ok():

**pass** *# Only called on clients, not server. Will go unused; not useful here.*

**func** \_server\_disconnected():

**pass** *# Server kicked us; show error and abort.*

**func** \_connected\_fail():

**pass** *# Could not even connect to server; abort.*

**remote** **func** register\_player(info):

*# Get the id of the RPC sender.*

**var** id = get\_tree().get\_rpc\_sender\_id()

*# Store the info*

player\_info[id] = info

*# Call function to update lobby UI here*

Sử dụng remote từ khóa trên hàm register\_player :

**remote** **func** register\_player(info):

Từ khóa này có hai cách sử dụng chính. Đầu tiên là cho Godot biết rằng chức năng này có thể được gọi từ RPC. Nếu không có từ khóa nào được thêm vào, Godot sẽ chặn mọi nỗ lực gọi hàm để bảo mật. Điều này làm cho công việc bảo mật dễ dàng hơn rất nhiều (vì vậy máy khách không thể gọi hàm xóa tệp trên hệ thống của máy khách khác).

Cách sử dụng thứ hai là chỉ định cách hàm sẽ được gọi thông qua RPC. Có bốn từ khóa khác nhau:

* remote
* remotesync
* master
* puppet

Từ remote có nghĩa là rpc()cuộc gọi sẽ đi qua mạng và thực hiện từ xa.

Từ remotesync có nghĩa là rpc()cuộc gọi sẽ đi qua mạng và thực hiện từ xa, nhưng cũng sẽ thực hiện cục bộ (thực hiện một cuộc gọi hàm bình thường).

Người dùng cũng có thể sử dụng get\_rpc\_sender\_id chức năng trên SceneTreeđể kiểm tra xem máy ngang hàng nào thực sự thực hiện cuộc gọi RPC register\_player.

Với điều này, quản lý tiền sảnh ít nhiều nên được giải thích. Sau khi trò chơi của bạn bắt đầu, rất có thể người dùng sẽ muốn thêm một số bảo mật bổ sung để đảm bảo người chơi không làm bất cứ điều gì sai trái (chỉ cần xác thực thông tin mà họ gửi theo thời gian hoặc trước khi trò chơi bắt đầu). Vì mục đích đơn giản và vì mỗi trò chơi sẽ chia sẻ thông tin khác nhau, điều này không được hiển thị ở đây.

1. **Bắt đầu game**

Khi đã có đủ người chơi tập trung tại sảnh, máy chủ có thể sẽ bắt đầu trò chơi.

Cảnh của người chơi

**7.1 Cảnh của người chơi**

Mỗi người chơi có thể sẽ có cảnh riêng . Đối với trò chơi 4 người chơi, mỗi người ngang hàng cần có 4 nút người chơi.

Vì vậy, làm thế nào để đặt tên cho các nút như vậy? Trong Godot, các nút cần phải có một tên duy nhất. Người chơi cũng phải tương đối dễ dàng để biết nút nào đại diện cho mỗi ID người chơi.

Giải pháp là chỉ cần đặt tên cho các nút gốc của các cảnh trình phát phiên bản cài đặt làm ID mạng của chúng . Bằng cách này, chúng sẽ giống nhau trong mọi máy ngang hàng và RPC sẽ hoạt động tốt! Đây là một ví dụ:

**remote** **func** pre\_configure\_game():

**var** selfPeerID = get\_tree().get\_network\_unique\_id()

*# Load world*

**var** world = load(which\_level).instance()

get\_node("/root").add\_child(world)

*# Load my player*

**var** my\_player = preload("res://player.tscn").instance()

my\_player.set\_name(str(selfPeerID))

my\_player.set\_network\_master(selfPeerID) *# Will be explained later*

get\_node("/root/world/players").add\_child(my\_player)

*# Load other players*

**for** p **in** player\_info:

**var** player = preload("res://player.tscn").instance()

player.set\_name(str(p))

player.set\_network\_master(p) *# Will be explained later*

get\_node("/root/world/players").add\_child(player)

**7.2 Đồng bộ hoá bắt đầu trò chơi**

Việc thiết lập trình phát có thể mất nhiều thời gian khác nhau cho mọi đối tượng ngang hàng do độ trễ, phần cứng khác nhau hoặc các lý do khác. Để đảm bảo trò chơi sẽ thực sự bắt đầu khi mọi người đã sẵn sàng, việc tạm dừng trò chơi cho đến khi tất cả người chơi sẵn sàng có thể hữu ích:

**remote** **func** pre\_configure\_game():

get\_tree().set\_pause(true) *# Pre-pause*

Khi máy chủ nhận được đồng ý từ tất cả các máy chủ ngang hàng, nó có thể yêu cầu họ bắt đầu, chẳng hạn như:

**var** players\_done = []

**remote** **func** done\_preconfiguring():

**var** who = get\_tree().get\_rpc\_sender\_id()

*# Here are some checks you can do, for example*

assert(get\_tree().is\_network\_server())

assert(who **in** player\_info) *# Exists*

assert(**not** who **in** players\_done) *# Was not added yet*

players\_done.append(who)

**if** players\_done.size() == player\_info.size():

rpc("post\_configure\_game")

**remote** **func** post\_configure\_game():

*# Only the server is allowed to tell a client to unpause*

**if** 1 == get\_tree().get\_rpc\_sender\_id():

get\_tree().set\_pause(false)

*# Game starts now!*

1. **Đồng bộ hoá trò chơi**

Trong hầu hết các trò chơi, mục tiêu của mạng nhiều người chơi là trò chơi chạy được đồng bộ hóa trên tất cả những người cùng chơi. Bên cạnh việc cung cấp RPC và triển khai bộ biến thành viên từ xa, Godot bổ sung thêm khái niệm về mạng chủ.

**8.1 Network master**

Network master của một nút là ứng dụng ngang hàng có quyền cuối cùng đối với nó.

Khi không được thiết lập rõ ràng, nút chính của mạng được kế thừa từ nút cha, nút này nếu không được thay đổi, sẽ luôn là máy chủ (ID 1). Do đó, máy chủ có quyền đối với tất cả các nút theo mặc định.

Mạng tổng thể có thể được đặt bằng hàm Node.set\_network\_master (id, đệ quy) (đệ quy là true theo mặc định và có nghĩa là tổng thể mạng cũng được đặt đệ quy trên tất cả các nút con của nút).

Việc kiểm tra xem một phiên bản nút cụ thể trên một máy ngang hàng có phải là mạng chính cho nút này cho tất cả các máy ngang hàng được kết nối hay không được thực hiện bằng cách gọi Node.is\_network\_master () . Điều này sẽ trở lại true khi được thực thi trên máy chủ và falsetrên tất cả các ứng dụng khách.

Mỗi máy ngang hàng được thiết lập để có quyền quản lý mạng cho trình phát riêng (Node) của họ thay vì máy chủ:

*# Load my player*

**var** my\_player = preload("res://player.tscn").instance()

my\_player.set\_name(str(selfPeerID))

my\_player.set\_network\_master(selfPeerID) *# The player belongs to this peer; it has the authority.*

get\_node("/root/world/players").add\_child(my\_player)

*# Load other players*

**for** p **in** player\_info:

**var** player = preload("res://player.tscn").instance()

player.set\_name(str(p))

player.set\_network\_master(p) *# Each other connected peer has authority over their own player.*

get\_node("/root/world/players").add\_child(player)

Mỗi khi đoạn mã này được thực thi trên mỗi máy ngang hàng, máy ngang hàng tự làm chủ trên nút mà nó kiểm soát và tất cả các nút khác vẫn là con rối với máy chủ là mạng chủ của chúng.

**8.2 Master and puppet keywords**

Lợi thế thực sự của mô hình này là khi được sử dụng với master/ puppet từ khóa trong GDScript (hoặc tương đương của chúng trong C # và Visual Script). Tương tự như remotetừ khóa, các chức năng cũng có thể được gắn thẻ với chúng

Ví dụ bom code:

**for** p **in** bodies\_in\_area:

**if** p.has\_method("exploded"):

p.rpc("exploded", bomb\_owner)

Ví dụ player code:

**puppet** **func** stun():

stunned = true

**master** **func** exploded(by\_who):

**if** stunned:

**return** *# Already stunned*

rpc("stun")

*# Stun this player instance for myself as well; could instead have used*

*# the remotesync keyword above (in place of puppet) to achieve this.*

stun()

Trong ví dụ trên, một quả bom phát nổ ở đâu đó (có thể được quản lý bởi bất kỳ ai là chủ nhân của nút bom này, ví dụ như máy chủ). Quả bom biết các cơ quan (các nút của người chơi) trong khu vực, vì vậy nó sẽ kiểm tra xem chúng có chứa một exploded phương thức trước khi gọi nó hay không.

Mỗi ngang hàng có một tập hợp hoàn chỉnh các phiên bản của các nút trình phát, một phiên bản cho mỗi đồng đẳng (bao gồm chính nó và máy chủ). Mỗi ngang hàng đã tự đặt mình làm cái chính của cá thể tương ứng với chính nó, và nó đã đặt một máy ngang hàng khác làm cái chính cho mỗi cá thể khác.

Quay trở lại lệnh gọi phương thức exploded, quả bom trên máy chủ đã gọi nó từ xa trên tất cả các cơ thể trong khu vực có phương thức. Tuy nhiên, phương pháp này nằm trong một nút trình phát và có một keyword master.

Từ master keyword về exploded phương thức trong nút trình phát có nghĩa là hai điều về cách thực hiện lệnh gọi này. Thứ nhất, từ quan điểm của máy ngang hàng đang gọi (máy chủ), máy ngang hàng đang gọi sẽ chỉ cố gắng gọi từ xa phương thức trên máy ngang hàng mà nó đã đặt làm mạng chủ của nút trình phát được đề cập. Thứ hai, từ quan điểm của máy chủ ngang hàng mà máy chủ đang gửi cuộc gọi đến, máy ngang hàng sẽ chỉ chấp nhận cuộc gọi nếu nó tự đặt mình là mạng chủ của nút trình phát với phương thức được gọi (có master keyword từ khóa).

Thiết lập trên có nghĩa là chỉ có người chơi bị ảnh hưởng mới có trách nhiệm thông báo cho tất cả những người chơi khác rằng cơ thể của họ đã bị choáng, sau khi va chạm quả bom của vật chủ. Do đó, ngang hàng sở hữu (vẫn còn trong phương thức exploded) cho tất cả các người chơi khác biết rằng nút trình phát của người chơi đó đã bị choáng. Máy ngang hàng thực hiện điều này bằng cách gọi stun phương thức từ xa trên tất cả các phiên bản của nút trình phát đó (trên các máy ngang hàng khác). Bởi vì phương thức stun có puppet keyword, chỉ những người chơi không đặt mình làm chủ mạng của nút mới gọi nó (nói cách khác, những người chơi đó được đặt làm con rối cho nút đó do không phải là chủ mạng của nó).

Chủ nhân của quả bom (vật chủ) lặp lại các bước trên cho từng khu vực, sao cho tất cả các trường hợp của bất kỳ người chơi nào trong khu vực đặt bom sẽ bị choáng trên màn hình của tất cả các người chơi.