Get started with Addressables

Course Overview

Đã bao giờ bạn muốn tạo ra một trò chơi hoạt động tốt trên hầu hết mọi thiết bị chưa? Thế còn DLC và các update theo chủ đề thì sao? Đó là những thứ mà Addressables có thể giúp bạn.

Hệ thống Addressables của Unity là một giải pháp quản lý tài nguyên, nó cho phép bạn **tài nguyên bạn cần** và **khi bạn cần.** Nó cho phép bạn cấu trúc tài nguyên của mình ngay Unity Editor khi bạn đang phát triển trò chơi, và API của nó cho phép bạn load và unload tài nguyên ngay khi trò chơi đang chạy.

Kết thúc khóa học này bạn sẽ:

- Nắm được lợi ích và chức năng của hệ thống Addressables để xác định xem nó có phù hợp với dự án của bạn không.
- Tạo và cấu hình tài nguyên theo địa chỉ để đóng gói AssetBundles.
- Quản lý quá trình build ứng dụng với Addressable
- Load và Instantiate các Addressable trong runtime
- Quản lý tài nguyên bằng các Group
- Quản lý tài nguyên bằng Label
- Thiết lập và cấu hình một local hosting server ngay trong Unity Editor để giả lập quá trình phân phối và nạp nội dung từ xa khi đang trong Play mode.

Tại sao phải sử dụng Addressables?

Tóm tắt

Hệ thống `Addressables` của Unity là một hệ thống quản lý asset động (*dynamic*). Nó được xây dựng dựa trên nền tảng công nghệ `AssetBundles` của Unity đồng thời cung cấp các công cụ ngay bên trong Unity Editor để giúp bạn chuẩn bị asset để load theo yêu cầu. Bất kể nội dung nào dù trên thiết bị hay trên cloud.

Trong hướng dẫn này, bạn sẽ có một cái nhìn tổng quan về cách tiếp cận của Unity đối với việc quản lý asset động, các lợi ích mà `Addressables` mang lại.

Bài hướng dẫn này bạn sẽ:

- Định nghĩa thế nào là quản lý asset động `(dynamic asset management)`
- Giải thích khái niệm AssetBundle và các lợi ích của nó.
- Giải thích cách hệ thống Addressables đơn giản hóa việc quản lý AssetBundle.

Tổng quan

Bạn đã bao giờ muốn tạo ra một tựa game mobile có hiệu suất chạy mượt mà trên gần như mọi thiết bị chưa? Hay việc thêm các DLC hoặc cập nhật nội dung theo chủ đề ngày lễ cho game của bạn thì sao? Đây chính là lúc hệ thống `Addressables` phát huy tác dụng.

Hệ thống `Addressables` của Unity là một giải pháp quản lý asset động, giúp cung cấp cho người dùng chỉ những asset mà họ thực sự cần, vào đúng thời điểm họ cần chúng. Nó cũng giúp bạn quy hoạch các asset của mình để đóng gói thành các `AssetBundles` ngay từ bên trong Unity Editor trong quá trình bạn phát triển game. Đồng thời, Runtime API của nó cho phép bạn load và unload asset một cách linh hoạt khi người dùng đang chơi game.

Bạn có thể chọn đóng gói tất cả asset đi kèm với bản build hoặc lưu trữ chúng trên một mạng lưới phân phối từ xa (CDN) ví dụ như Unity Cloud Content Delivery của Unity. Vị trí lưu trữ của asset được trừu tượng hóa thông qua các địa chỉ là một `string` do bạn tự định nghĩa.

Trong bài hướng dẫn này, bạn sẽ tìm hiểu về cách tiếp cận của Unity đối với việc quản lý asset động và lý do tại sao hệ thống `Addressables` có thể là một sự bổ sung đáng giá cho dự án của bạn. Bạn cũng sẽ nắm được thêm thông tin về khóa học này, để từ đó sẵn sàng bắt tay vào làm việc với

hệ thống `Addressables`.

Trước khi bắt đầu

Get started with Addressables là một khóa trung cấp.

Để hoàn thành khóa học này bạn cần có kinh nghiệm lập trình với Unity. Bên cạnh các kiến thức nền tảng đã được đề cập trong lộ trình Junior Programmer, bạn cũng nên nắm vững kiến thức về `event handlers` và `asynchronous methods`.

Chúng tôi cũng khuyến nghị bạn hoàn thành lộ trình Creative Core, hoặc có các kinh nghiệm tương đương để làm quen với tất cả các asset trong Unity mà bạn có thể áp dụng hệ thống `Addressables` với chúng.

Quản lý asset động là gì?

Hệ thống quản lý asset động (`Dynamic asset management`) giúp bạn cải thiện hiệu suất game bằng cách cho phép bạn quyết định chính xác việc tải asset **khi nào** và **từ đâu,** quyết định khi nào cần nạp vào và xóa khỏi bộ nhớ trong runtime.

Trong suốt lịch sử phát triển của Unity, đã có một vài phương pháp khác nhau để quản lý asset trong bộ nhớ. Hệ thống `Addressables` được thiết kế để thay thế hoặc cái tiến các hệ thống cũ - những hệ thống vốn đã lỗi thời hoặc không còn đủ khả năng đáp ứng cho một dự án vượt xa khỏi giai đoạn prototype. Trong khóa học này, bạn sẽ tiến hành nâng cấp một tựa game đang sử dụng một số phương pháp cũ để chuyển hoàn toàn sang hệ thống `Addressables`.

Dưới đây là một vài công cụ quản lý asset khác mà Unity đang hoặc đã từng cung cấp.

Tham chi?u tr?c ti?p (Direct references)

Khi bạn kéo thả một asset vào một scene hoặc vào một component thông qua Inspector của Editor, bạn đang tạo ra một tham chiếu trực tiếp tới asset đó. Khi bạn build ứng dụng của mình các asset này được lưu lại trong một tệp dữ liệu riêng biệt liên kết chặt chẽ với scene của bạn. Khi ứng dụng chạy trên thiết bị và scene được load lên, Unity Player sẽ nạp toàn bộ asset này vào bộ nhớ RAM trước khi scene thực sự xuất hiện, nhằm đảm bảo rằng scene có sẵn quyền truy cập vào tất cả mọi thứ mà nó có tham chiếu.

Nếu bạn chỉ sử dụng tham chiếu trực tiếp trong game thì cách quản lý này không linh hoạt (*not dynamic*). Việc load và unload toàn bộ scene là quyền kiểm soát duy nhất mà bạn có. Bạn có thể sẽ phải đối mặt với rủi ro tạo ra một game có build size khổng lồ và hiệu suất siêu tệ đặc biệt là trên các thiết bị có dung lượng RAM hạn chế.

Th? m?c Resources

Thư mục Resources và Resource API đã từng cung cấp một cách đơn giản để quản lý asset trong bộ nhớ. Trong cấu trúc thư mục của các dự án Unity cũ, bạn có thể bắt gặp một hoặc nhiều thư mục Resources chứa asset. Trong quá trình build ứng dụng, Editor sẽ tìm kiếm asset ở tất cả các thư mục Resources và đóng gói chúng thành một tệp tuần tự (`serialized file`) đi kèm với nó là `metadata` và `indexing infomation`, tệp này sau đó được đóng gói thẳng vào ứng dụng của bạn. Resource API cho phép bạn viết script để load và unload các asset nằm trong thư mục Resources này.

Hệ thongse Resources tồn tại một số nhược điểm so với các hệ thống mới hơn:

- Nó không cho phép quản lý bộ nhớ một cách chi tiết.
- Nó làm chậm thời gian khởi động ứng dụng.
- Nó có thể làm phình to dung lượng bản build
- Và nó hoàn toàn không hỗ trợ phân phối nội dung từ xa (CDN)

H? th?ng AssetBundle

Hệ thống AssetBundle quy hoạch asset vào các thùng containers được gọi là AssetBundles. Giống như thư mục Resources, hệ thống AssetBundle gom asset thành các tệp riêng biệt. Tuy nhiên, khác với Resources, AssetBundles có thể được lưu trữ cục bộ đi kèm với bản build hoặc lưu trữ trên cloud.

Thông qua bộ API của mình, hệ thống AssetBundles giúp giảm thiểu tối đa sự tiêu tốn băng thông và asset hệ thống. Nó làm được điều đó bằng cách cho phép bạn tải xuống các bundle chỉ khi nào cần thiết, nhờ đó mà bạn có thể bổ sung DLC và các bản cập nhật nội dung sau khi game đã ra mắt . Ví dụ bạn có thể phân phối nội dung mới để người chơi nạp tiền mua mà không bắt buộc họ phải tải lại toàn bộ file cài đặt mới từ store. Một khi các bundle được tải về máy, AssetBundles API cũng cấp các phương thức để load và unload asset từ các bundle đó.

Mặc dù là một hệ thống mang tính tùy biến rất cao, AssetBundles vẫn có những hạn chế nhất định khiến các lập trình viên trước đây phải tự xây dựng các giải pháp của riêng họ.

- AssetBundles chỉ có thể sử dụng thông qua scripts, giao diện trong Unity Editor rất hạn chế và quá trình build AssetBundles cũng yêu cầu bạn phải viết code.
- Bản thân AssetBundles API không tự động theo dõi sự phụ thuộc của asset giữa các AssetBundles khác nhau. Ví dụ: nếu bạn muốn load một prefab từ AssetBundle A, bạn sẽ phải tự tìm mọi thành phần phụ thuộc của nó như mesh, material, texture... nằm rải rác ở các AssetBundle khác. Sau đó bạn phải đảm bảo rằng các AssetBundles phụ thuộc đó đã được load xong trong lúc game chạy và trước khi bạn load prefab đó.
- Việc cấp phát và giải phóng bộ nhớ là tự tiếp và hoàn toàn thủ công. Do đó, rất dễ xảy ra tình trạng bạn vô tình unload một asset khỏi AssetBundle trong khi các đoạn code khác vẫn cần sử dụng asset đó. Điều này có thể dẫn đến lỗi thiếu nội dung hoặc rò rỉ bộ nhớ. Vấn đề này trở nên cực kỳ rắc rối với race conditions đòi hỏi bạn phải gia cố code của mình thật chặt chẽ để phòng ngừa.
- AssetBundles runtime API không hề biết bạn đang lưu trữ các bundle đã build ở trên thiết bị hay trên cloud. Điều này bắt buộc bạn phải tự lưu và theo dõi đường dẫn của từng AssetBundle, xem nó nằm trên cloud hay trên thiết bị để gọi hàm tải về cho đúng.

Addressables

Hệ thống Addressables được xây dựng dựa trên nền tảng của AssetBundle API nhằm cung cấp cho bạn một giao diện người dùng trực quan ngay trong Unity Editor, đồng thời tự động hóa các quy trình mà trước đây vốn chỉ có thể quản lý thủ công thông qua việc viết code. Với hệ thống Addressables, bạn có thể quy hoạch các asset của mình ngay trong Unity Editor và để mặc cho hệ thống tự xử lý các vấn đề dependencies, vị trí lưu trữ cũng như việc cấp phát và giải phóng bộ nhớ.

Tại sao bạn nên sử dụng Addressables?

Khi một asset được đánh dấu là Addressable nó sẽ sở hữu một địa chỉ duy nhất mà bạn có thể dùng để tham chiếu tới trong scripts của mình thay vì phải gọi bằng tên hay vị trí trong bundle. Mặc dù nghe có vẻ đơn giản nhưng địa chỉ này lại là chìa khóa cho phép hệ thống Addressables tự động hóa rất nhiều chi tiết phức tạp ở tầng dưới.

Dưới đây là giải thích chi tiết tại sao bạn nên sử dụng Addressables

S? linh ho?t

Hệ thống Addressables mang lại cho bạn sự linh hoạt trong việc chỉ định nơi lưu trữ asset. Bạn có thể để asset đi kèm trong bản build hoặc tải xuống khi cần từ một máy chủ. Sau đó bạn có thể thay đổi vị trí lưu trữ của một asset cụ thể, chẳng hạn như từ local sang remote hoặc từ một bundle nguyên khối (*monolithic bundle*) sang các bundle phân mảnh (*granular bundles*) khác. Tát cả đều có thể thược hiện mà không cần phải viết lại source code.

Qu?n lý s? ph? thu?c

Hệ thống Addressables tự động nạp tát cả các thành phần phụ thuộc của bất kỳ asset nào mà bạn load sao cho tất cả các mesh, shader, animation và các asset khác được nạp đầy đủ trước khi asset mà bạn yêu cầu được load lên.

Ví dụ: nếu bạn load một nhân vật thì hệ thống sẽ tự động load đầy đủ mesh, material và cả texture, shader mà material cần.

Qu?n lý b? nh?

Hệ thống Addressables tự động hóa phần lớn công việc tẻ nhạt của quá trình quản lý bộ nhớ. Khi bạn load và unload asset khỏi game thông qua code, hệ thống sẽ tự động theo dõi việc cấp phát bộ nhớ.

Bạn cũng có thể sử dụng các công cụ profiler của hệ thống để cải thiện hiệu năng sử dụng bộ nhớ có game của mình.

?óng gói n?i dung hi?u qu?

Bởi vì hệ thống Addressables thiết lập bản đồ cho các chuỗi phụ thuộc phức tạp nên nó giúp bạn đóng gói asset một cách hiệu quả ngay cả khi bạn chuyển hoặc đổi tên asset. Bạn có thể lựa chọn mức độ phân mảnh mà bạn muốn đối với các asset tại một vị trí lưu trữ, ưu tiên đóng gói riêng biệt hoặc gộp chúng lại với nhau. Bạn cũng có thể dễ dàng chuẩn bị asset cho cả việc triển khai local lẫn remote để hỗ trợ các nội dung được load theo yêu cầu, DLC. Điều này có thể giúp giảm build size.

Cloud Build và Phân ph?i n?i dung

Hệ thống Addressables đã được tích hợp vào nền tảng Unity Gaming Services (UGS), cụ thể là các dịch vụ Cloud Content Delivery và Cloud Build, mang đến cho bạn các dịch vụ toàn diện từ cập nhật game trực tiếp cho đến build ứng dụng.

Scriptable Build Pipeline

Hệ thống Addressables sử dụng Scriptable Build Pipeline (SBP), một quy trình mạnh mẽ và ổn định hơn so với build AssetBundle cũ. Bạn có thể sử dụng các luồng build được thiết lập sẵn hoặc sử dụng phương pháp nâng cao hơn nếu muốn. Bạn có thể tự tạo luồng build của riêng mình thông qua việc sử dụng API đã được chia nhỏ.

Localization

Hệ thống này cũng được tích hợp gói Localization của Unity để bạn có thể sử dụng nhiều ngôn ngữ khác nhau trong dự án của mình.

Chúng ta sẽ làm gì trong khóa học này?

Trong khóa học này chúng ta sẽ được học:

- Các kiến thức cơ bản về giao diện của hệ thống Addressables
- Những cách để biến asset thành addressable và load chúng bằng code.
- Cách Unity build ứng dụng với asset là addressable và cách quy hoạch dự án sao cho build hiệu quả nhất.
- Tùy chọn lưu trữ asset phù hợp với mục đích của bạn.

Trong các bài hướng dẫn tiếp theo, bạn sẽ học các kiến thức cơ bản về hệ thống Addressables thông qua việc thực hành với dự án của chúng tôi, một tựa game có tên là Loady Dungeons. Loady Dungeons được phát triển bằng cách sử dụng các tham chiếu trực tiếp và hệ thống Resources cũ. Trong suốt quá trình học, bạn sẽ tiến hành nâng cấp dự án này để chuyển sang hệ thống Addressables và học hỏi mọi thứ về addressable.

Bước tiếp theo

Cho dù bạn là một indie dev muốn thu hút sự chú ý trên các nền tảng di động hay là thành viên của một đội ngũ sản xuất lớn với những nhu cầu phức tạp về phân phối nội dung, hệ thống Addressables sẽ giúp tựa game của bạn trở nên tinh gọn và hoạt động nhanh hơn, đồng thời tự động hóa nhiều tác vụ quản lý tài nguyên trong runtime.

Tiếp theo, đã đến lúc bắt tay vào thực hành trong Unity. Bạn sẽ khám phá dự án Loady Dungeons và bắt đầu tiến hành thiết lập các asset thành addressable.

Biến asset thành addressable

Tóm tắt

Trong hướng dẫn này, bạn sẽ cài đặt package Addressables và học cách truy cập giao diện của nó.

Kết thúc hướng dẫn này bạn sẽ:

- Biết cách mở giao diện Addressables trong Unity Editor.
- Nhận diện asset có thể biến thành addressable
- Biết cách để đánh dấu asset là addressable
- Chỉnh sửa địa chỉ của asset

Tổng quan

Loady Dungeons là một game mobile mà người chơi điều khiển một con khủng long đi tìm chìa khóa trong rương và tìm cách để mở một cánh của để sang màn tiếp theo.

Loady Dungeons hiện đang sử dụng các tham chiếu trực tiếp và thư mục Resources xuyên suốt toàn bộ dự án. Nhiệm vụ của bạn là áp dụng hệ thống Addressables để có thể giảm thiểu dung lượng tải về, dễ dàng cập nhật nội dung suốt vòng đời của game và cho phép người dùng thêm hoặc cập nhật nội dung trong quá trình chơi.

Trong hướng dẫn này bạn sẽ bắt đầu áp dụng hệ thống Addressables vào dự án.

Trước khi bắt đầu

?i?u ki?n tiên quy?t

Để hoàn thành khóa học này bạn cần có kinh nghiệm lập trình với Unity. Bên cạnh tất cả các khái niệm đã được đề cập trong lộ trình Junior Programmer thì kiến thức về event handlers và lập trình bất đồng bộ cũng rất hữu ích. Chúng tôi sẽ bổ sung các tài liệu tham khảo để bạn học thêm nếu cần thiết.

Chúng tôi cũng khuyến khích bạn hoàn thành lộ trình Creative Core, hoặc có kinh nghiệm tương đương để làm quen với asset trong Unity - những asset mà bạn sẽ áp dụng hệ thống Addressables

C?p nh?t Unity Hub

Trước khi thiết lập dự án Unity, hãy cân nhắc cập nhật Unity Hub lên phiên bản mới nhất. Nếu bạn đang sử dụng một phiên bản cũ của Unity Hub bạn có thể thấy khác biệt giữa tài liệu hướng dẫn và giao diện thực tế

Thi?t l?p d? án Unity

Để thiết lập Unity và dự án cho quá trình học tập. Hãy làm theo các hướng dẫn sau:

1. Cài Unity 2022.2.9 hoặc mới hơn nếu bạn chưa có
2. Tải xuống file zip được đính kèm. (Tải xuống)
3. Giải nén file vừa tải về vào một thư mục mới. Hãy lưu nó tại một vị trí phù hợp vì chúng ta sẽ dùng nó xuyên suốt khóa học.
4. Thêm dự án trong Unity Hub. Bạn có thể thấy cảnh báo phiên bản Unity không phù hợp. Nhưng đừng lo bạn có thể upgrade project với bất kỳ phiển bản Unity nào từ 2022.2.9 trở lên, kể cả các bản 2022 LTS.
5. Mở dụ án
6. Cài đặt package Addressables phiên bản 1.21.8 trở lên.

Addressable là gì?

Khi bạn đánh dấu một asset là addressable, Unity sẽ tạo ra một địa chỉ. Địa chỉ này là một chuỗi định danh mà hệ thống Addressables liên kết với vị trí của asset đó, bất kể asset đó đang nằm trên thiết bị cùng với bản build hay trên một hệ thống phân phối (CDN).

Trong scripts, bạn có thể sử dụng địa chỉ của asset thay vì phải tự theo dõi vị trí của nó. Nhờ đó, bạn không cần phải viết code để chỉ định *where* mà chỉ cần chỉ định *what*. Bằng cách này, nếu vị trí của tài nguyên thay đổi, bạn sẽ không phải viết lại code.

Mặc dù tốt nhất địa chỉ nên là độc nhất, nhưng bạn không bị bắt buộc phải làm vậy. Bạn sẽ được học nhiều cách để tham chiếu đến asset addressable mà không cần địa chỉ độc nhất. Nhưng việc giữ chúng là độc nhất sẽ giúp dự án của bạn ngăn nắp và dễ quản lý hơn.

Tạo addressables trên Inspector

Nhấn nút **PLAY** và làm quen với Loady Dungeons.

Hiện tại, Loady Dungeons được thiết lập để cho phép người chơi lựa chọn chiếc mũ mà họ thích cho nhân vật khủng long, nhưng nó đang sử dụng hệ thống Resources cũ. Để giúp game trở nên tinh gọn và nhanh hơn, bạn sẽ tiến hành cập nhật dự án để chuyển sang sử dụng Addressables

Hãy cùng biến những chiếc mũ này thành addressable thông qua Inspector

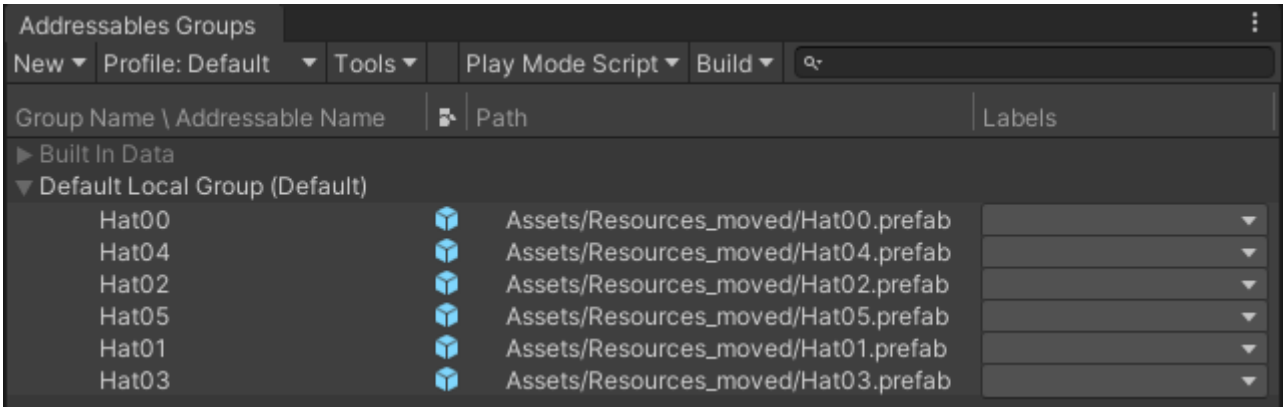
1. Trong cửa số Project, hãy chỏ tới Assets > Resources
   Hãy nhớ rằng Resources là một thư mục đặc biệt trong Unity. Nó là một phần của hệ thống Resources cũ dành cho việc quản lý tài nguyên động. Khi bạn build game ở trạng thái hiện tại, bất kỳ asset nào nằm trong thư mục Resources cũng sẽ được đưa vào một indexed file riêng biệt. Bạn đang tiến hành cấu hình lại để chuyển sang sử dụng Addressables.
2. Chọn prefab Hat00 và nhìn sang cửa sổ Inspector.
3. Trong cửa sổ Inspector, bật thuộc tính **Addressable.** *(Nếu không thấy, hãy kiểm tra lại Package Manager xem bạn đã cài package Addressable chưa).*

4. Khi này sẽ có một cửa sổ hiện lên hỏi bạn có muốn chuyển prefab này sang một thư mục khác không? Hãy chọn **Yes.** Vì nếu không thì việc tạo addressable sẽ trở nên vô ích.
5. Bạn cũng có thể chọn nhiều prefab cùng lúc và bật Addressable để tiết kiệm thời gian.

Mở cửa sổ Addressables Groups

Bạn cũng có thể biến asset thành addressable bằng cách kéo thả prefab vào cửa số Addressables Groups

Để mở cửa sổ này. Trong Unity Editor, Windows > Asset Management > Addressables > Groups. Vì bạn đã tạo một vài addressable nên bạn sẽ thấy nó như thế này.



Prefab sẽ tự động được đặt trong Default Local Group. Bạn sẽ còn tiếp tục tìm hiểu thêm về của sổ Addressables Groups ở các phần sau của khóa học này.

Tạo addressable trong cửa sổ Addressables Groups

Tiếp theo, chúng ta sẽ biến logo game thành addressable.

Để xem asset này. Bạn có thể chạy game từ scene MainMenu (trong folder Assets > Scenes). Bạn sẽ thấy ngay logo của game.

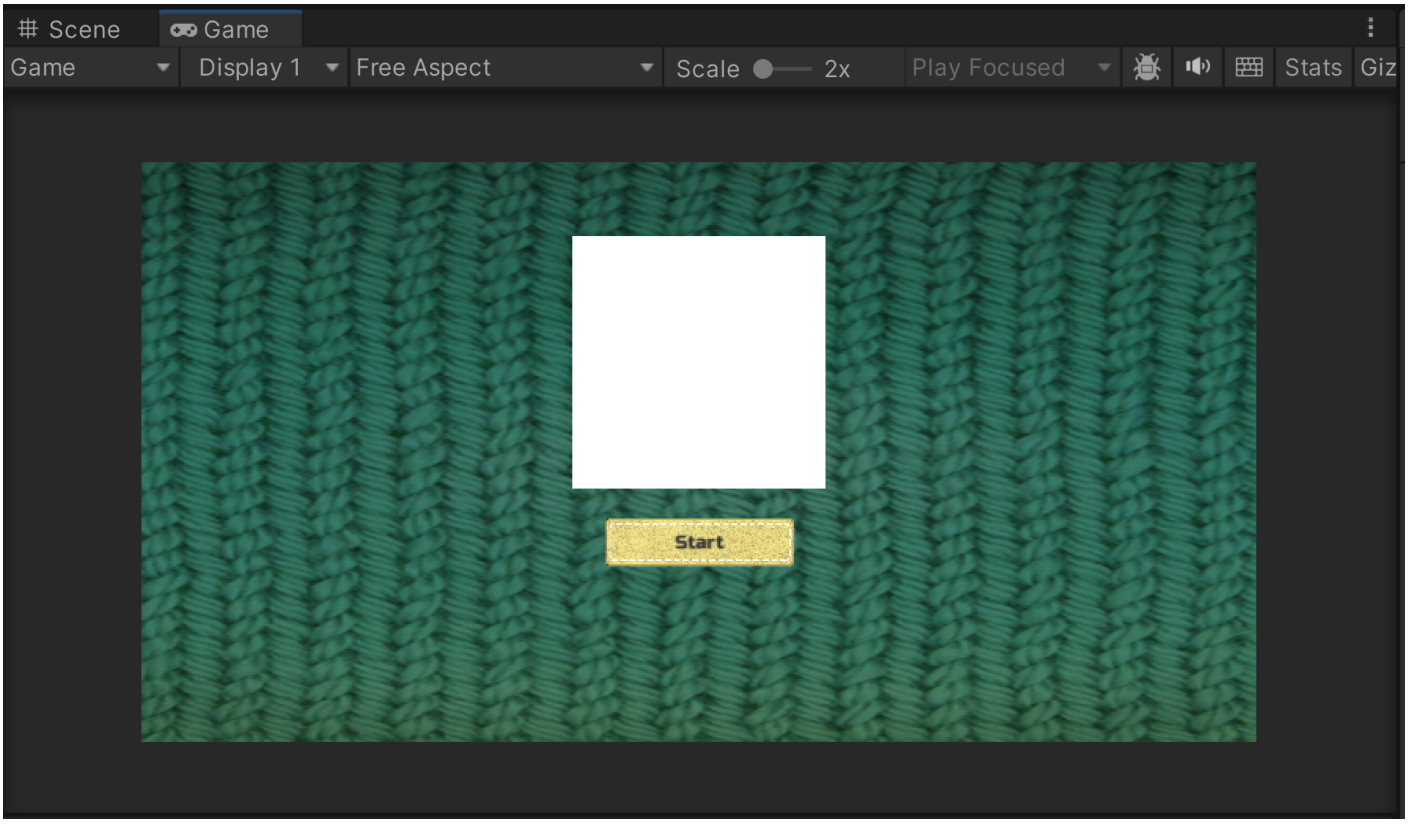
Vì logo còn nằm trong folder Resources nên nó được load lên bởi hệ thống Resources

Để biến logo này thành addressable với cửa sổ Addressables Groups thay vì Inspector. Bạn có thể làm như sau:

1. Đi tới thư mục Resources
2. Kéo file LoadyDungeonsLogo vào Default Local Group trong của sổ Addressables Groups
3. Bạn cũng sẽ thấy của sổ Move From Resources. Hãy chọn Yes

LoadyDungeonsLogo giờ đã là một phần của Default Local Group, và trong Inspector bạn cũng sẽ thấy thuộc tính Addressable đã được bật.

Để kiểm tra xem bạn đã làm đúng chưa. Hãy play game lại từ scene MainMenu. Logo không hiển thị được nữa vì hệ thống Resources không còn tìm thấy nó nữa.



Thay đổi địa chỉ của asset

Địa chỉ của asset là một string ID có nhiệm vụ định danh addressable asset. Bạn có thể sử dụng địa chỉ này như một key để load asset thông qua code.

Ch?nh s?a ??a ch? thông qua Inspector

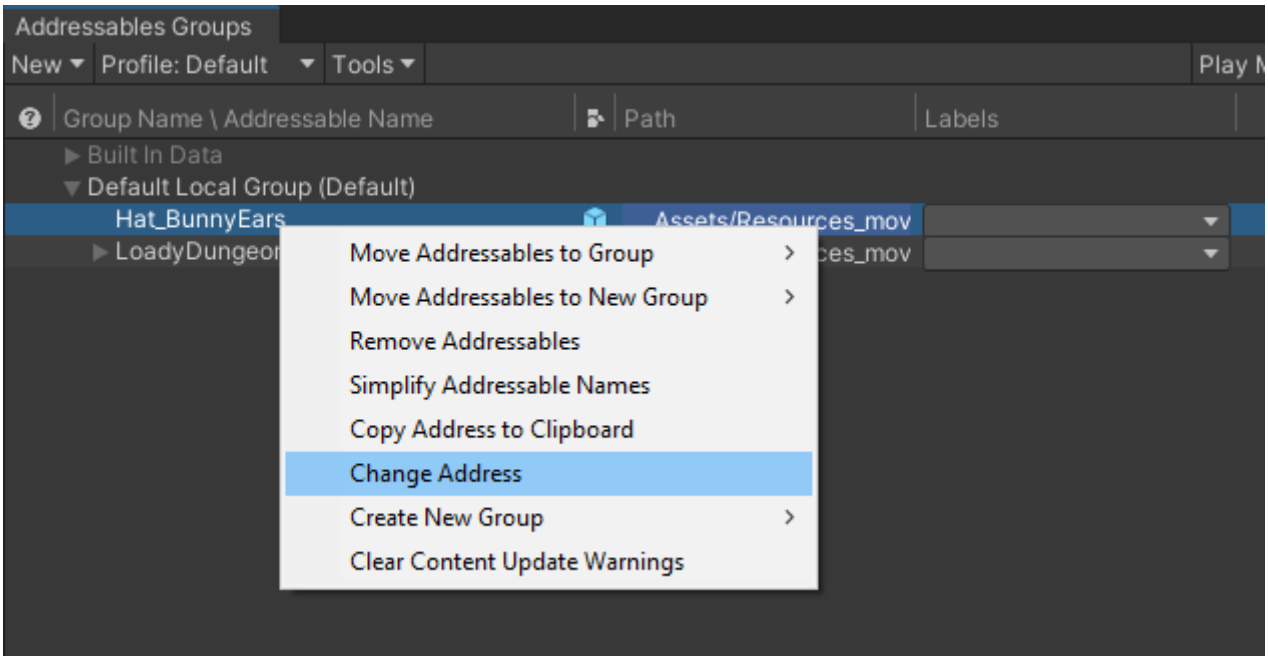
Bạn có thể chỉnh sửa địa chỉ thông qua của sổ Inspector sau khi bạn bật addressable.

1. Tìm đến prefab Hat00. Vừa nãy nó đã được chuyển tới thư mục Assets > Resources_moved.
2. Trong cửa sổ Inspector, thay đổi địa chỉ thành tên bất kỳ bạn muốn. Ở đây tôi để là Hat_BunnyEars
3. Giờ hãy tìm đến scene Assets > Scenes > LoadingScene nhưng đừng mở nó.
4. Trong Inspector, hãy bật addressable với scene này.
5. Thay đổi địa chỉ thành LoadingScene

Bạn sẽ thấy sự thay đổi trên cả 2 cửa sổ Addressables Groups bà Inspector.

Ch?nh s?a ??a ch? thông qua Addressables Groups

Một cách khác là bạn có thể thay đổi địa chỉ này thông qua của sổ Addressables Groups. Trong cửa sổ Addressables Groups chuột phải vào prefab mà bạn muốn thay đổi, chọn **Change Address.**



Một khi game đã được phát hành trên chợ, bạn nên tránh việc thay đổi địa chỉ của asset. Bản thân các asset có thể thay đổi thường xuyên trong suốt vòng đời dự án nhưng bất kỳ địa chỉ nào đại diện cho một asset đều nên được giữ nguyên. Điều này giúp bạn tránh việc phải sửa đổi source code nếu chỉ có nội dung asset thay đổi, bởi vì việc thay đổi source code bắt buộc cúng ta phải phát hành lại ứng dụng trên chợ.

Dọn dẹp

Mặc dù không bắt buộc nhưng bây giờ bạn có thể làm 2 việc

- Đổi tên thư mục Resources_moved. Vì Resources_moved chỉ là cái tên tạm mà Unity dùng. Bạn có thể đổi thành bất cứ thứ gì bạn muốn.
- Xóa thư mục Resources vì bây giờ nó đã trống.

Tiếp theo

Trong bài này bạn đã biết cách chuyển asset sang hệ thống Addressable. Trong bài sau bạn sẽ học cách load asset lên thông qua Addressable API.

Load addressable assets bằng scripts

Tóm tắt

Trong bài học này chúng ta sẽ tìm hiểu một vài cách để load addressable asset vào game và làm thế nào để giải phóng chúng khỏi bộ nhớ bằng scripts. Chúng tôi đã đính lèm một số tài liệu tham khảo và các phần ôn tập về những khái niệm lập trình mà bạn sẽ sử dụng với hệ thống Addressables.

Kết thúc bài này chúng ta sẽ:

- Định nghĩa một thao tác bất đồng bộ (asynchronous operation).
- Xác định các cách khác nhau để tham chiếu đến một addressable trong code
- Xác định các phương pháp khác nhau để load và release các addressable một cách bất đồng bộ.

Tổng quan

Trong hướng dẫn này, bạn sẽ học cách load và unload addressable vào game thông qua code. Trong suốt bài thực hành này, bạn sẽ tiến hành loại bỏ phần lớn source code sử dụng Resource API trong dự án Loady Dungeons và thay bằng Addressables API.

Trước khi bắt đầu

Trước khi bắt đầu chúng ta hãy cùng ôn tập lại những khái niệm quan trọng trong lập trình và đặc biệt cần thiết với Addressables.

L?p trình b?t ??ng b?

Trong Addressables API, nhiều tác vụ thực hiện việc load asset và dữ liệu sau đó trả về kết quả. Khi các tài nguyên hoặc dữ liệu đó nằm trên một máy chủ, các tác vụ này sẽ cần thêm thời gian để hoàn thành. Để tránh làm giảm hiệu suất và gây trì hoãn các tác vụ khác. Hệ thống Addressables sử dụng các thao tác bất đồng bộ, điều này có nghĩa là nhiều tác vụ có thể chạy đồng thời.

Để triển khai các thao tác bất đồng bộ với Addressables API bạn sẽ cần hiểu rõ những khái niệm sau (những khái niệm này nằm trong project Intermediate scripting):

- **Coroutines** cho phép bạn thực thi một hàm trong nhiều frame thay vì chờ nó hoàn thành ngay trong một frame.
- **Delegates** là các container đại diện cho các hàm có thể truyền đi dưới dạng tham số hoặc sử dụng như các biến thông thường.
- **Events** là một dạng delegate chuyên biệt, có khả năng phát tín hiệu thông báo cho class khác biết rằng có một sự kiện

Structs

Addressables API sử dụng struct có tên là AsyncOperationHandle để giữ tác vụ bất đồng bộ bên trong code của bạn.

Struct là một kiểu tham trị trong C#, có thể được sử dụng để lưu các dữ liệu liên quan với nhau thành một nhóm. Chúng giống với class ở chỗ có thể chứa các filed và các method nhưng khác với class chúng được truyền theo giá trị thay vì truyền theo tham chiếu. Có nghĩa là khi bạn tạo một struct và truyền vào một method hoặc gán nó cho một biến, một bản sao của struct sẽ được tạo ra thay vì tạo ra tham chiếu trỏ tới đối tượng gốc.

Struct AsyncOperationHandle có một giá trị bool để xác định xem method thành công hay thất bại và trả về giá trị sau quá trình load

Tu?n t? hóa (Serialization)

Tuần tự hóa là quá trình chuyển đổi và lưu trữ dữ liệu giữa các phiên làm việc (sesstion) của một ứng dụng. Giải tuần tự hóa (Deserialization) là quá trình trích xuất dữ liệu đã lưu trữ đó để nó có thể được tái cấu trúc lại khi ứng dụng chạy ở những lần sau.

Trong hướng dẫn này, bạn không cần trở thành một chuyên gia tuần tự hóa. Bạn chỉ cần biết rằng Unity thực hiện tuần tự hóa dữ liệu trong quá trình build để nó có thể giải tuần tự hóa dữ liệu được lưu trữ đó khi ứng dụng chạy. Thuộc tính [SerializeField] có tác dụng đánh dấu các trường dữ liệu private là được tuần tự hóa và các trường đã được tuần tự hóa sẽ có thể hiển thị trong Inspector để bạn có thể chỉnh sửa như các biến public.

Khi bạn sử dụng Addressables, bạn sẽ phải sử dụng quá trình tuần tự hóa để tối ưu hóa cách dự án của bạn tiêu thụ tài asset.

Bạn có thể tìm hiểu thêm về tuần tự hóa trong Unity Manual.

Load một addressable prefab

Dưới đây là một vài cách để load một addressable prefabs với Addressables API. Hãy bắt đầu với việc load asset bằng cách sử dụng địa chỉ của nó, điều mà bạn sẽ thực hiện trong quá trình chuyển đổi script `PlayerConfigurator` sang sử dụng Addressable API.

Để viết lại `PlayerConfigurator.cs` với Addressables API thay vì Resources API, bạn có thể làm theo hướng dẫn dưới đây:

1. Từ cửa sổ Project, mở file `Assets > Scripts > PlayerConfigurator.cs`
2. Khai báo các namespace của Addressable cần sử dụng

   ```
   using UnityEngine.AddressableAssets;
   using UnityEngine.ResourceManagement.AsyncOperations;
   ```

   Bạn sẽ cần sử dụng namespace này để load asset thông qua địa chỉ của nó và truy cập vào `AsyncOperationHandle` được trả về.
3. Trong class PlayerConfigurator tạp một biến string mới đại diện cho địa chỉ. Đặt attribute [SerializeField] ở trước đó để bạn có thể serialize nó và chỉnh sửa giá trị ngay trong Inspector.

   ```
   [SerializeField] private string m_Address;
   ```

   Sau khi khai báo bạn có thể thấy biến này xuất hiện trong của sổ Inspector với tên là Address
4. Trong script hãy tìm dòng code dưới đây. Nó đang sử dụng Resources API

   ```
   private ResourceRequest m_HatLoadingRequest;
   ```

   Và thay nó bằng dòng code sử dụng Addressables API dưới đây:

   ```
   private AsyncOperationHandle<GameObject> m_HatLoadOpHandle;
   ```

   Dòng mới này khai váo biến `m_HatLoadOpHandle` là một `AsyncOperationHandle` của GameObject.
5. Trong hàm `SetHat()` hãy xóa toàn bộ phần thân của nó đi và thay bằng dòng code dưới đây

   ```
   m_HatLoadOpHandle = Addressables.LoadAssetAsync<GameObject>(m_Address);
   ```

   Dòng code này sẽ load bất đồng bộ prefab thông qua địa chỉ `m_Address`. Biến `m_HatLoadOpHandle` sẽ chứa một giá trị bool cho biết asset có được tải thành công hay không. Nếu load thành công, handle này cũng chứa kết quả của request tức là chính prefab chúng ta đang load.
6. Bởi vì asset được load bất đồng bộ, nên các dòng code phía sau nó sẽ được chạy mà không cần chờ `LoadAssetAsync` chạy xong. Vậy nên để biết được việc load có thành công hay không chúng ta phải đăng ký một event để biết khi nào load xong.

```
m_HatLoadOpHandle.Completed += OnHatLoadComplete;
```

Dòng code này dùng để đăng ký một event handler (chúng ta sẽ khai báo nó ở bước sau). Vì chúng ta cần biết khi nào asset được load xong nên ở đây chúng ta sử dụng event `Completed` của `AsyncOperationHandle`

7. Ở đây chúng ta sẽ viết hàm `OnHatLoadComplete()` thay cho hàm `OnHatLoaded()` để in ra trạng thái của handle. Hàm này được gọi bởi Addressables khi nó thực hiện xong yêu cầu tải prefab ở trên.

```
private void OnHatLoadComplete(AsyncOperationHandle<GameObject> asyncOperationHandle)
{
    Debug.Log($"AsyncOperationHandle Status: {asyncOperationHandle.Status}");
}
```

8. Trong hàm OnDisable(), xóa bỏ toàn bộ phần body và thay thế bằng dòng code dưới đây để hủy đăng ký sự kiện khi gameObject bị disable. Nó sẽ chặn biệc Addressables gọi `OnHatLoadComplete()` khi `PlayerConfigurator` không hoạt động.

```
m_HatLoadOpHandle.Completed -= OnHatLoadComplete;
```

9. Save script và quay lại Unity.
10. Để test script này, mở scene Assets > Scenes > Level_00.
11. Trong cửa sổ Project, chọn prefab Assets > Prefabs > Player.
12. Nhìn sang cửa sổ Inspector, tìm component `PlayerConfigurator` và đặt giá trị cửa biến Address thành địa chỉ của chiếc mũ. Ở phần trên chúng ta đã đặt là Hat_BunnyEars.
13. Play game và nhìn vào cửa sổ Console, bạn sẽ thấy dòng thông báo "*AsyncOperationHandle Status: Succeeded*".
    Tại thời điểm này, asset được được load thành công. Tuy nhiên, bạn sẽ chưa nhìn thấy nó trên scene vì bạn chưa instantiate nó.
14. Giờ là lúc bạn instantiate prefab mà bạn vừa load lên. Viết lại hàm `OnHatLoadComplete()` như dưới đây:

```
private void OnHatLoadComplete(AsyncOperationHandle<GameObject> asyncOperationHandle)
{
    if (asyncOperationHandle.Status == AsyncOperationStatus.Succeeded)
    {
        Instantiate(asyncOperationHandle.Result, m_HatAnchor);
    }
}
```

15. Vào lại Play mode và bạn sẽ thấy player di chuyển cùng cái mũ.
16. Bạn có thể thử với những địa chỉ khác mà chúng ta đã tạo ở bài hướng dẫn trước.

> Lưu ý: Trong ví dụ này Address là một string và nó sẽ không xác thực xem bạn có nhập đúng địa chỉ hay không. Bạn chỉ biết khi bạn thực sự chạy load. Bạn có thể nhập bất cứ thứ gì và điều đó có thể gây ra lỗi khi chạy game. Hãy đảm bảo rằng bạn nhập chính xác địa chỉ của prefab muốn load.

Load một addressable prefab với AssetReference

Asset Reference là một kiểu dữ liệu dùng để tham chiếu đến addressable asset. Nó được thiết kế để sử dụng như một trường dữ liệu có thể serializable trong MonoBehaviour hay ScriptableObject. Khi bạn thêm một AssetReference vào trong các class này, bạn có thể gán một địa chỉ cho nó trong cửa sổ Inspector thông qua một công cụ chọn. Những lựa chọn này được giới hạn nghiêm ngặt, chỉ cho phép chọn các asset đã được đánh dấu là addressable.

Về mặt hình thức, bạn sẽ khai báo các AssetReference vào script giống hệt như khi bạn sử dụng tham chiếu trực tiếp, tức là có thể thông qua các trường public hoặc các trường private có sử dụng attribute [SerializableField]. Tuy nhiên, AssetReference không lưu trữ trực tiếp một tham chiếu đến asset. Thay vào đó, AssetReference lưu trữ mã định danh GUID của asset đó. GUID này được hệ thống Addressables sử dụng để lưu trữ và truy xuất đối tượng trong runtime.

Lợi ích cốt lõi của việc sử dụng AssetReference so với việc dùng địa chỉ dạng string là nó bắt buộc bạn chỉ được chọn asset thông qua Inspector. Điều này giúp tránh các rủi ro như không đánh có như sai chính tả...

Tiếp theo chúng ta sẽ viết lại `PlayerConfigurator` sử dụng AssetReference thay cho địa chỉ dạng string.

1. Vẫn là mở file `Assets > Scripts > PlayerConfigurator.cs`
2. Thay `m_Address` bằng dòng dưới đây:

   ```
   [SerializeField] private AssetReference m_HatAssetReference;
   ```
3. Trong hàm `SetHat()` chúng ta thay phần body dòng code sau để load asset:

   ```
   if (!m_HatAssetReference.RuntimeKeyIsValid()) return;
   m_HatLoadOpHandle = m_HatAssetReference.LoadAssetAsync<GameObject>();
   m_HatLoadOpHandle.Completed += OnHatLoadComplete;
   ```

   `RuntimeKeyIsValid` thực hiện kiểm tra xem các giá trị được chọn có phải là một địa chỉ hợp lệ không. Trong trường hợp này nó sẽ chặn `LoadAssetAsync` nếu `AssetReference` bị bỏ trống (None);
4. Save script và quay trở lại với Level_00;
5. Trong cửa sổ Project, chọn prefab Assets > Prefabs > Player.
6. Trong cửa sổ Inspector, tìm đến component PlayerConfigurator. Bây giờ trường Address đã được thay thế bằng trường Hat Asset Reference. Tại đây hãy chọn cái mũ mà bạn muốn.
7. Play game và xem nhân vật có di chuyển cùng cái mũ bạn chọn không.
8. Giờ bạn có thể thử chọn các AssetReference khác và quan sát kết quả.

Bạn cũng có thể đọc thêm về AssetReference tại đây.

Load một addressable prefab với AssetReferenceGameObject

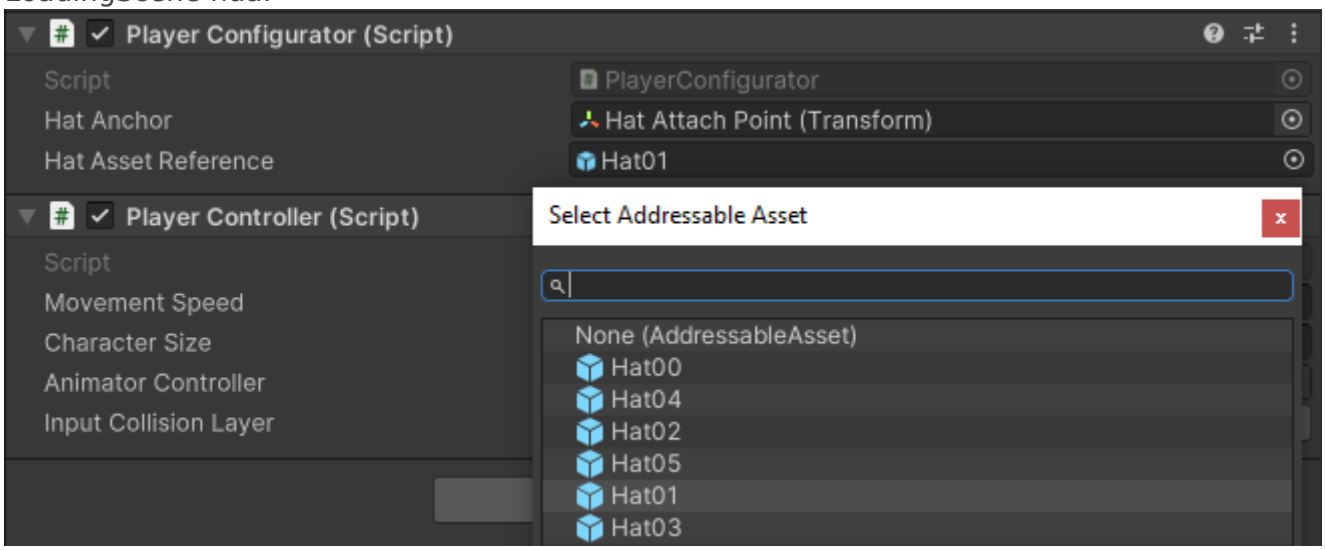
Công cụ chọn của AssetReference cho phép chọn tất cả các asset được đánh dấu là addressable nên bạn có thể thấy nó cho phép bạn chọn cả texture, scene... Mặc dù được phép chọn nhưng nếu bạn cố load chúng như những GameObject nó sẽ gây ra lỗi trong code của Addressables.

Bạn có thể muốn quản lý chặt chẽ hơn nữa và chỉ cho phép chọn các prefab từ công cụ chọn. Để làm được điều này bạn có thể sử dụng `AssetReferenceGameObject` thay vì `AssetReference`.

Tiếp tục viết lại `PlayerConfigurator` nhé.

1. Mở file `Assets > Scripts > PlayerConfigurator.cs`
2. Thay `m_HatAssetReference` bằng dòng dưới đây:

   ```
   [SerializeField] private AssetReferenceGameObject m_HatAssetReference;
   ```
3. Save script và quay trở lại Unity
4. Trong cửa sổ Project, chọn prefab Assets > Prefabs > Player.
5. Trong cửa sổ Inspector, tìm đến component PlayerConfigurator. Mở picker lên và chọn một cái mũ bạn muốn. Chú ý xem hiện giờ bạn không thể chọn LoadyDungeonsLogo hay LoadingScene nữa.



6. Play game và thử cái mũ mới.

Bạn cũng có thể đọc thêm về AssetReferenceGameObject tại đây.