1.Tham chiếu tới components là gì?
Mỗi gameObject đều có một loạt các components định nghĩa nó và component cơ bản nhất của gameObject là Transform nhằm định nghĩa vị trí của object đó trong thế giới game.

Đa số các components đều có các giá trị được public ra ở Inspector giúp bạn chỉnh sửa dễ dàng trên Unity Editor.

Vậy làm sao để bạn chỉnh sửa các thông số này khi runtime – game đang chạy?

Thay vì sửa tay các giá trị trên tab Inspector, chúng ta sẽ truy cập vào component đó bằng scripts và sửa giá trị thì sẽ dễ dàng và linh hoạt hơn.

Giả sử bạn có một MonoBehaviour tên là PlayerController và muốn tham chiếu tới component Rigidbody trong cùng một gameObject để thay đổi vận tốc. Unity hỗ trợ bạn hàm GetComponent<T>(); để PlayerController có thể tham chiếu tới Rigidbody:

```
public class PlayerController : MonoBehaviour
{
   private void Awake()
   {
      // tham chiếu tới Rigidbody trong cùng gameObject.
      Rigidbody rigid = GetComponent<Rigidbody>();
   }
}
```

Biến rigid của bạn sẽ là instance của Rigidbody gắn trên cùng gameObject, bạn có thể sử dụng hàm GetComponent<T>(); để lấy bất kỳ component nào liên kết với gameObject này.

“Ủa nếu gameObject mình không có Rigidbody thì sao?”

Giá trị biến rigid sẽ bằng null.

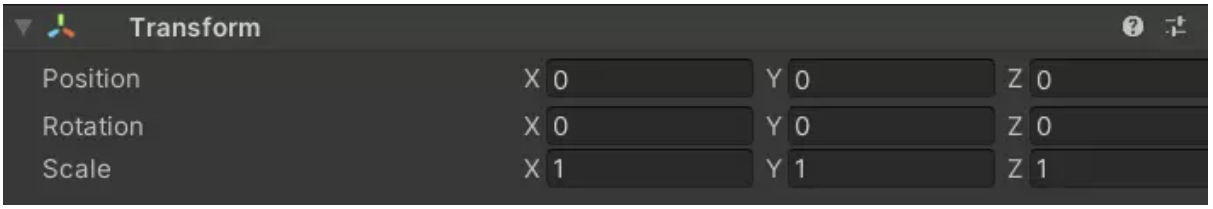
Đồng thời đối với component Transform, các bạn không cần phải GetComponent<Transform>();, trong mỗi MonoBehaviour đều tham chiếu sẵn tới nó. Như vậy bạn chỉ cần truy cập trực tiếp:

```
public class PlayerController : MonoBehaviour
{
   private void Awake()
   {
      transform.position = Vector3.zero; // không lỗi
   }
}
```

“Sao trong class Component cũng có rigidbody như transform mà mình không sử dụng?“

Không như transform, rigidbody bạn phải sử dụng GetComponent<Rigidbody>() bởi thuộc tính này đã lỗi thời: “Property rigidbody has been deprecated. Use GetComponent() instead. (UnityUpgradable)”. Các phiên bản hoặc tài liệu cũ vẫn có thể sử dụng như vậy.

Các biến public và Unity Editor
Một trong những tính năng “đỉnh cao” nhất của Unity Editor đó là khả năng show các biến config của Component cho chúng ta chỉnh sửa, giống như component Transform cho phép bạn chỉnh sửa thông số của: position, rotation và scale.



Transform component
Chúng ta cũng có thể kiểm soát được biến nào cần show ra và biến nào cần ẩn đi bằng access modifier: public, protected, private,…

Giả sử component PlayerController cần show 2 biến để bạn điều chỉnh trên Editor:

```
public class PlayerController : MonoBehaviour
{
    public float speed = 5.0f;

    public float sensitive = 0.8f;
}
```

“Sao phải chỉnh sửa thông số trên Editor mà không phải trong scripts?”

Có rất nhiều các thông số mà bạn phải config trước cho component, chúng ta đưa nó lên Editor để có thể cân bằng đến khi hợp ý chúng ta. Nếu bạn sửa trong code, bạn cần phải compile code mới có thể test được sự thay đổi.

Đồng thời việc này sẽ rất thuận tiện cho các bạn Game Designer chỉnh sửa ngay trên Editor, bởi có thể họ không biết code hoặc cấu trúc code của bạn hơi rối đối với họ.

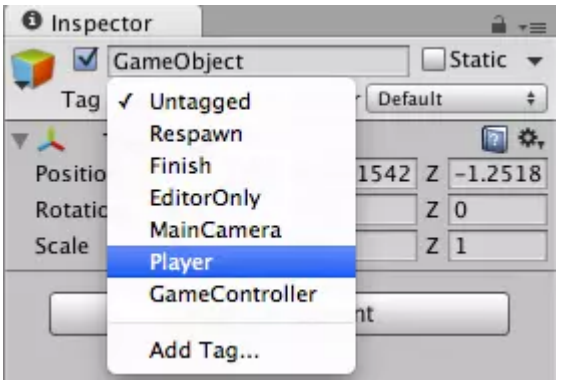
Nhưng đừng lạm dụng quá biến public chỉ vì chúng có thể show ra trên tab Inspector của Editor nhé. Bạn có thể sử dụng biến private và attribute [SerializeField] như SerializeField, Header và một số attributes trong Unity3D

Tham chiếu và truy cập Components/ GameObjects
Các bạn tự phân biệt tham chiếu tới Components và tham chiếu tới GameObjects nhé.

Ở trên mình đã nói về concept tham chiếu giữa các components, chúng ta sẽ đi qua các cách phức tạp hơn một chút để tham chiếu.

2.Tag là gì?
Trong Unity, chúng ta có một khái niệm gọi là Tags, bạn có thể đánh nhiều gameObjects cùng một tag, như kiểu bạn phân loại gameObject vậy.

Ví dụ, enemy có nhiều loại A, B, C, D nhưng mình có thể phân biệt chúng đều là Enemy bằng tag: Enemy.



Bạn có thể set tag cũng như thêm tag mới của một gameObject ở trên Unity Editor, ngay dưới tên của gameObject trên tab Inspector nhé.

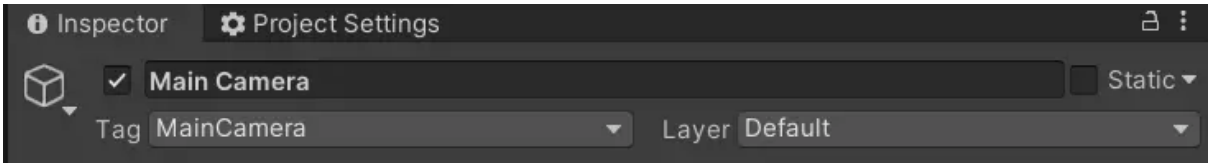
Tham chiếu bằng tên hoặc Tag
Như vậy một gameObject sẽ liên kết với một tên và một tag, đồng thời cả 2 yếu tố này có thể giúp bạn tham chiếu tới gameObject hoặc components.

Đầu tiên mình sẽ tham chiếu tới các GameObjects khác nhé:

```
private void Awake()
{
   // tham chiếu tới gameObject có tên là Main Camera
   GameObject camera = GameObject.Find("Main Camera");
}
```
Hoặc sử dụng tag:

```
GameObject camera = GameObject.FindGameObjectWithTag("MainCamera");
```

Và lưu ý rằng, để biến camera không bằng null thì Camera phải có tag là MainCamera nhé, thông thường mặc định khi tạo scene thì camera sẽ có tag “MainCamera” sẵn cho bạn.



“Vậy nếu có nhiều gameObject cùng tags thì sao?”

Thì hàm sẽ trả về gameObject đầu tiên mà nó query được, nếu bạn muốn lấy hết tất cả các objects có tags chỉ định thì:

GameObject [] enemies = GameObject.FindGameObjectsWithTag("Enemy");

Có rất nhiều hàm tìm kiếm các GameObject cũng như tìm kiếm Components bạn có thể search trên trang documentation của Unity.

Tuy nhiên bạn cần lưu ý rằng, các lệnh tìm kiếm tương đối chậm (vẫn nhanh nhưng chậm so với các API khác), vì vậy mình khuyến khích nên sử dụng chúng trong các hàm khởi tạo chỉ gọi một lần như Start(), Awake() và không nên đặt trong Update(), for, while,…

“Còn lưu ý nào không?”

Khác với tên của file trong folder, tên của gameObject không phải là duy nhất và chúng có thể trùng nhau, vì vậy không nên xem tên gameObject như là id để phân biệt các gameObject nhé.

Truy cập thuộc tính của components khác
Thông thường, đây là một yêu cầu cơ bản nhất trong game, component A cần biết thông tin của component B.

Giả sử như gameObject Enemy đang đuổi đánh Player chẳng hạn, component EnemyAI cần phải biết vị trí của Player để di chuyển tới mà vị trí của Player được lưu trữ trong Transform. Như vậy EnemyAI cần tham chiếu tới Transform của Player để lấy được vị trí.

Chúng ta sẽ thực hiện 3 công việc cho EnemyAI: tham chiếu tới gameObject Player, tham chiếu tới component Transform của gameObject Player và truy cập thuộc tính position:

```
public class EnemyAI : MonoBehaviour
{
   public GameObject player; // the player object

   void Start()
   {
      GameObject player = GameObject.Find("Player"); // (1)
   }

   void Update()
   {
      MoveTo(player.transform.position); // (2), (3)
   }

   void MoveTo(Vector3 target)
   {
```

```
        //...
    }
}
```

3.Xử lý va chạm

Game của chúng ta hoạt động theo hướng sự kiện (event driven), khi có một sự kiện xảy ra, chúng ta xử lý nó.

Collider là gì?

Một vài events được tạo bởi người dùng: input keyboard, một vài lại xảy ra theo một khoảng thời gian nhất định: Update(), và một vài events sẽ xảy ra do chính game của chúng ta.

Điều mình muốn nói đến ở đây là sự kiện va chạm giữa các gameObjects, khi có va chạm xảy ra, chúng ta sẽ muốn xử lý sự kiện này. Giả sử viên đạn va chạm với enemy chẳng hạn, bạn sẽ phải viết code để làm nổ viên đạn, xóa viên đạn, trừ máu enemy, kiểm tra máu enemy > 0, xóa enemy,…

Và để 2 gameObject va chạm, điều kiện cần của chúng là phải có một component gọi là Collider (bạn có thể gọi là hit box cho quen thuộc), các Collider sẽ tạo ra các khối hình, nếu các khối hình này đè lên nhau thì chúng phát ra sự kiện va chạm.

2 loại collider: collision và trigger

Collider chia thành 2 loại: collisions và triggers.

Bạn hãy tưởng tượng collision là một khối vật thể cứng, va chạm vào thì gameObject sẽ bị bật ra. Trong khi Trigger là một vật thể không chạm được như tia laser, màn sương hoặc con ma, bạn vẫn có thể phát hiện tia laser đang “chĩa” vào bạn mà không hề có va chạm vật lý nào xảy ra.

Giả sử, khi Player chạy và va chạm với cánh cửa, có thể player sẽ bị bật ra hoặc/ và cánh cửa bị vỡ ra, như vậy chúng va chạm theo kiểu collision.

Giả thiết khi Player đi vào căn phòng không có cửa, bạn vẫn có thể phát hiện ra khi Player đi vào phòng bằng cách đặt một cách cửa vô hình, chúng va chạm theo kiểu trigger.

Các hàm giúp bạn xử lý bao gồm:

Collision: void OnCollisionEnter(), void OnCollisionStay(), void OnCollisionExit();
Trigger: void OnTriggerEnter(), void OnTriggerStay(), void OnTriggerExit();
Nếu là game 2D thì bạn thêm 2D vào cuối tên hàm nhé.

“Tại sao cả 2 gameObjects của mình có component Colilders mà chúng không phát hiện va chạm vậy?”

Đúng vậy, 2 gameObjects chỉ có Collider mà không có Rigidbody ở một trong 2 sẽ không có va chạm, bởi chúng không tính là xử lý vật lý nếu không có Rigidbody.

4.Rigidbody là gì?

Trước đây, mình sẽ đề cập tới Rigidbody nhưng không nói rõ ràng nó là gì và chúng ta làm được gì với component này.

Lại tưởng tượng các gameObject trên Scene của bạn là các objects “tĩnh”, bởi chúng không sinh động và tương tác với nhau bằng lực và Rigidbody sẽ giúp các objects này “thật” hơn.

Khi một gameObject được gắn thêm Rigidbody, chúng sẽ trở thành một thực thể sống, chịu tác động của trọng lực, bị lực tác động, tác động lực lên vật khác, có trọng lượng,…, giả sử mình đặt trọng lượng của nó bằng 10f chẳng hạn:

GetComponent<Rigidbody>().mass = 10.0f;

Và bạn không cần phải tự mình định nghĩa va chạm là như thế nào? Khi va chạm thì gameObject bật ra như thế nào? Tất cả đã có Unity Engine lo việc này, việc của chúng ta là xử lý sau khi va chạm.

5.Di chuyển vật lý
Để di chuyển một tác nhân vật lý (có rigidbody) chúng ta không sử dụng transform, mà thường sử dụng lực bằng cách AddForce(…):

rb.AddForce(Vector3.up * 10f) ; // tác động một lực hướng lên

“Tại sao không di chuyển bằng component Transform?”

Bạn có thể đọc thêm ở bài Cơ bản về Rigidbody trong Unity3D

Rất nhiều người sử dụng cách này để tạo lực một cách tự nhiên nhất, tuy nhiên để kiểm soát tốc độ của vật một cách ổn định, mình thường sử dụng .velocity:

GetComponent<Rigidbody>().velocity = new Vector3(0, 10, 0);

Cách này sẽ làm gameObject di chuyển không tự nhiên bằng AddForce, tuy nhiên mình có thể kiểm soát tốc độ tốt hơn.

6.Kinematic
Trong trường hợp các bạn không muốn vật bị tác động bởi lực, như mô phỏng thang máy chẳng hạn, bạn có thể đặt kiểu body của Rigidbody là Kinematic.

Như vậy vật sẽ không hẳn di chuyển theo engine vật lý của Unity, bạn có thể điều khiển nó trong scripts đồng thời giảm đi tài nguyên mà Unity phải xử lý.

Bạn vẫn có thể nhận được các events va chạm nhé.

7.Static
Một trong những cách cực kỳ tối ưu cho vật lý là kiểu body Static, đối với những object không di chuyển như tòa nhà (building), mặt đất (ground),… bạn có thể đặt nó là Static.

Lưu ý rằng 2 objects static thì không va chạm nhau nhé.

Đồng thời, trong lúc chạy game, bạn cũng có thể chuyển từ một object dynamic sang object static và ngược lại, tuy nhiên trong Unity documentation khuyến cáo rằng việc này khá nguy hiểm bởi cách mà hệ thống vật lý của Unity xử lý.