1. Graphic
2. Lighting

* Direct lighting và indirect lighting
* Direct lighting: Không bị phản xạ hoặc phản xạ một lần
* Indirect lighting: Có hai hay nhiều lần phản xạ trước khi đến vị trí quan sát. Indirect lighting với 4 hay nhiều hơn lần phản xạ được biết là Global Illumination. GI là hệ thống giúp cho việc tạo hình ánh sáng đến từ một phẳng ảnh hưởng đến mặt phẳng khác
* Light types
* Directional Light: thể hiện nguồn sáng lớn và xa đến từ vị ngoài phạm vi của thế giới trong game. Thường được sử dụng để mô phỏng mặt trời trong trò chơi. Vị trí của nó không ảnh hưởng đến tất cả các Object cho dù nó có ở xa đến đâu. Thay đổi các góc của nó có thể thay đổi hướng chiếu của nguồn sáng
* Point Light: được đặt ở một điểm trong không gian và toả sáng như nhau trong mọi hướng. Nó có một tầm chiếu sáng nhất định và chỉ ảnh hưởng đến các vật nằm trong phạm vi chiếu sáng. Vật nằm càng xa nguồn sáng thì càng ít bị ảnh hưởng bởi ánh sáng.
* Spot Light: được đặt ở một điểm rõ ràng và chiếu sáng trong một chiều với một góc nhất định kết quả ra một cột sáng hình nón
* Area Light: chỉ chiếu sáng một hướng nhất định với cột sáng hình chữ nhật. Cần được bake để nó có thể hoạt động
* Light modes
* Realtime: tất cả các tính toán liên quan đến ánh sáng được tính và cập nhật mỗi frame. Tốn tài nguyên CPU và GPU
* Baked: các tính toán liên quan đến ánh sáng được tính ở editor và lưu lại trong ổ lưu trữ, khi chạy nó sẽ được sử dụng. Giảm độ tốn tài nguyên khi chạy nhưng không thể thay đổi theo thời gian. Do vậy nếu sử dụng Baked Lighting cho các vật có thể di chuyển hoặc có thể phá huỷ thì bóng của chúng vẫn sẽ còn dù vật đã không còn ở đó tạo ra các khung cảnh không thực tế. Vì vậy nên sử dụng Baked Lighting cho các vật static.
* Mix: giúp có mọi lợi thế từ Baked trong khi giữ lại việc thay đổi vị trí bóng khi di chuyển mà Realtime lights thể hiện từ cùng một nguồn sáng
* Light settings
* Type: kiểu ánh sáng hiện tại bao gồm Directional, Point, Spot và Area
* Range: định nghĩa ảnh sáng có thể lan toả bao xa từ vị trí trung tâm của Object
* Spot Angle: định nghĩa góc trên cơ sở của một cụm đèn sáng Spot Light
* Color: đặt màu cho ánh sáng
* Mode: quy định chế độ chiếu sáng bao gồm Realtime, Mixed và Baked
* Intensity: độ sáng của ánh sáng
* Indirect Multiplier: dùng để thay đổi độ sáng của indirect light. Nếu Indirect Multiplier được đặt nhỏ hơn 1, độ sáng của ánh sáng sẽ giảm dần sau mỗi lần phản xạ còn đối với lớn hơn 1 thì ánh sáng sẽ sáng hơn sau mỗi lần phản xạ
* Shadows
  + Shadow Type: xác định ánh sáng tạo ra Hard Shadow, Soft Shadow hoặc không có bóng. Hard Shadow tạo ra bóng với các cạnh sắc. Hard Shadow không thực thế khi so sánh với Soft Shadow nhưng nó cần ít sự tính toán hơn
  + Baked Shadow Angle: nếu Mode được đặt là Baked hay Mixed, Type đặt là Directional, Shadow Type đặt là Soft Shadows, nó thêm một số sự làm mờ nhân tạo vào các cạnh khiến chúng trở nên tự nhiên hơn
  + Baked Shadow Radius: nếu Mode được đặt là Baked hay Mixed, Type đặt là Point hay Spot, Shadow Type được đặt là Soft Shadows, nó thêm một số sự làm mờ nhân tạo vào các cạnh khiến chúng trở nên tự nhiên hơn
  + Realtime Shadows: nếu Mode được đặt là Realtime hay Mixed, Shadow Type được đặt là Hard Shadows hay Soft Shadows, nó có thể kiểm soát tạo bóng trong thời gian thực
  + Strength: kiểm soát độ tối của bóng được tạo bằng ánh sáng được thể hiện bằng giá trị giữa 0 và 1
  + Resolution: kiểm soát đồ phân giải của phân giải của bóng. Độ phân giải cao hơn tăng tính chính xác của bóng như yêu cầu nhiều thời gian và dung lượng nhớ hơn
  + Bias: kiểm soát khoảng cách của bóng được đẩy ra từ ánh sáng, được định nghĩa trong khoảng từ 0 đến 2. Nó hữu dụng để tránh sự toạ bóng giả
  + Normal Bias: kiểm soát khoảng cách mà bề mặt bóng thay đổi theo chiều hướng của mặt phẳng. Nó hữu dụng để tránh sự toạ bóng giả
  + Near Plane: kiểm soát giá trị cho mặt gần khi hiển thị bóng
* Các cài đặt thêm:
  + Cookie: xác định một Texture mask thông qua bóng được tạo ra. Với VertexLit shaders không thể thể hiện Cookies và bóng
  + Draw Halo: chọn hộp này để tạo một vòng sáng hình cầu với đường kính bằng với giá trị của Range
  + Flare: nếu ta muốn đặt Flare được hiển thị tại ví trí của ánh sáng
  + Render Mode: xác định độ ưu tiên hiển thị của ánh sáng được chọn. Nó có thể ảnh hưởng đến chất lượng và hiệu suất
    - Auto: phương pháp render được xác định trong thời gian chạy, phụ thuộc vào độ sáng của ánh sáng gần và cài đặt chất lượng hiện tại
    - Important: ánh sáng luôn luôn được render ở chất lượng từng pixel.
    - Not Important: ánh sáng luôn luôn được render bằng một chế độ nhanh hơn
  + Culling Mask: dùng để loại bỏ một nhóm các Object không bị ảnh hưởng bởi ánh sáng

1. Cameras

* Clear Flags: thể hiện phần nào của màn sẽ bị xoá. Thường được sử dụng khi có nhiều Camera để chiếu các thành phần khác nhau của game.
  + Skybox: đây là cài đặt mặc định. Bất kỳ khoảng trống nào trên màn hình sẽ được hiện Skybox của Camera
  + Solid color: bất kỳ khoảng trống nào trong màn hình sẽ được hiển thị theo màu của Background Color
  + Depth only:
  + Don’t clear:
* Background: màu được sử dụng để áp dụng cho tất cả các thành phần còn lại khi tất cả các thành phần khác đã được thể hiện
* Culling Mask: gồm các layers được render bằng Camera
* Projection:
* Perspective: có độ sâu, có thể đánh giá được khoảng cách của các vật
* Orthographic: không có độ sâu, kích thước của vật giữ nguyên bất kể ở vị trí nào
* Size: độ lớn của Camera khi ở dạng Orthographic
* FOV Axis (Perspective Camera): trục của góc nhìn
* Horizontal: camera sử dụng góc nhìn ngang
* Veritcal: camera sử dụng góc nhìn dọc
* Field of view (Perspective Camera): góc nhìn của camera, đo bằng độ cùng với trục được xác định trong FOV Axis
* Physical Camera: khi Physical Camera được bật, Unity tính tầm nhìn của camera bằng cách sử dụng các thuộc tính mô phòng camera trong thế giới thực: Focal Length, Sensor Size và Lens Shift
  + Focal Length: đặt khoảng cách (mm) giữ Camera Sensor và Camera Lens. Giá trị càng nhỏ thì tầm nhìn càng rộng và ngược lại. Unity sẽ tự động cập nhật Field of View khi thay đổi giá trị này
  + Sensor Type: xác định camera thực tế ta muốn camera mô phỏng
  + Sensor Size: xác định kích thước của sensor (mm). Unity xác định giá trị của X (chiều rộng của sensor) và Y (chiều dài của sensor) tự động khi Sensor Type được chọn. Ta có thể nhập giá trị mong muốn nếu cần
  + Lens Shift: di chuyển len theo chiều dạo và chiều ngang từ vị trí trung tâm. Ta có thể sử dụng Lens Shift để chính xác sự thay đổi xảy ra khi camera nghiêng với vật thể. X độ lệch ngang của sensor, Y độ lệch dọc của sensor
  + Gate Fit: thay đổi độ lớn của cổng độ phân giải (kích thước/ tỷ lệ khung hình khi ở game view) so với công phim (kích thước/tỷ lệ khung hình của Physical Camera sensor)
    - Verical: thích hợp với chiều cao của độ phân giả của cổng phim. Nếu tỷ lệ khung hình của sensor lớn hơn tỷ lệ của game view, Unity sẽ cắt các bên của hình ảnh được render. Nếu tỷ lệ khung hình của sensor nhỏ hơn tỷ lệ của game view, Unity quét quá mức hình ảnh ở các bên
    - Horizontal: thích hợp với chiều dài của độ phân giả của cổng phim. Nếu tỷ lệ khung hình của sensor nhỏ hơn tỷ lệ của game view, Unity sẽ quét quá mức hình ảnh được render ở trên và dưới. Nếu tỷ lệ khung hình của sensor lớn hơn tỷ lệ của game view, Unity sẽ cắt hình ảnh được render ở trên và dưới
    - Fill: chỉnh độ phân giải của cổng trùng với chiều cao hay chiều rộng của cổng phim thứ mà có giá trị nhỏ hơn
    - Overscan: chỉnh độ phân giải của cổng trùng với chiều cao hay chiều rộng của cổng phim thứ mà có giá trị lớn hơn
    - None: bỏ qua cổng độ phân giải sử dụng mỗi cổng phim. Cách kéo này làm cho hình ảnh khớp với game view
* Clipping Planes: Khoảng cách từ camera từ chỗ bắt đầu và kết thúc render
  + Near: điểm gần nhất so với camera mà việc thể hiện sẽ xảy ra
  + Far: điểm xa nhất so với camera mà việc thể hiện sẽ xảy ra
* Viewport Rect
  + X: điểm bắt đầu theo chiều ngang mà camera view sẽ thể hiện
  + Y: điểm bắt đầu theo chiều dọc mà camera view sẽ thể hiện
  + W: chiều rộng của camera đầu ra trên màn hình
  + H: chiều cao của camera đầu ra trên màn hình
* Depth: thứ tự thể hiện của các camera. Camera với giá trị Depth lớn hơn sẽ thể hiện lên trên các camera có giá trị Depth nhỏ hơn
* Render Path: là một chuỗi các tính toán liên quan đến ánh sáng và đổ bóng. Các Render Path khác nhau có khả năng và hiệu xuất khác nhau. Ta có thể ghi đè cho mỗi Camera
  + Use Player Settings: sử dụng Rendering Path được xác định trong Player Settings
  + Vertex Lit: tất cả các Object được thể hiện bằng camera sẽ được render như Vertex-Lit Object. Render mỗi Object một lần với ánh sáng từ toàn bộ ánh sáng được tính toàn cho mỗi vertex. Nó là rendering path nhanh nhất và có nhiều phần cứng hỗ trợ nhất. Vì nó được tính toán dựa trên vertex, nó không hỗ trợ hiệu ứng như đổ bóng, normal mapping, light cookies hay các hiệu ứng có tính chất chi tiết cao
  + Forward: tất cả các Object sẽ được thể hiện với mỗi lần cho mỗi material. Các nguồn sáng được phân biệt khác nhau phụ thuộc vào cài đặt và độ sáng của chúng. Một số nguồn sáng sáng nhất được thể hiện bằng per-pixel lit sau đó có thể lên tới 4 điểm sáng khác được tính bằng per-vertex, các điểm sáng còn lại được tính bằng Spherical Harmonics có tốc độ nhanh hơn nhưng chỉ là sự xấp xỉ
* Target Texture: chỉ đến một Render Texture thứ được bao gồm ở nguồn ra của Camera view
* Occulusion Culling: xác định xem Camera có render các Object bị che bởi các Object khác VD như các Object ở đằng sau các bức tường
* Allow HDR: bật High Dynamic Range cho camera
* Allow MSAA: bật antialiasing cho camera
* Allow Dynamic Resolution: bật Dynamic Resolution cho Camera cho phép bạn đổi tỷ lệ của mục tiêu render để giảm thiệu công việc của GPU
* Target Display: xác định Camera sẽ render hình ảnh ra nguồn nào. Nằm trong 1 và 8

1. Materials

* Materials để thiể hiện bề ngoài của mặt phẳng
* Materials và shaders liên kết chặt chẽ với với nhau, ta luôn luôn sử dụng materials với shaders
* Materials chứa một tham chiếu dẫn đến một Shaders. Nếu Shader định nghĩa các thuộc tính của Materials thì Materials bao gồm các dữ liệu như màu sắc hay các tham chiếu đến các textures
* Material là một class đại diện cho material bằng C#, vì vậy tất cả các biến của material mà ta thấy trên cửa sổ Inspector có thể truy cập bằng script cho phép ta có thể thay đổi hay animate cách àm một material thể hiện ở thời gian chạy
* Một số các phương thức hay được dùng nhất:
  + SetColor: thay đổi màu của material
  + SetFloat: thay đổi giá trị biến
  + SetInteger: thay đỏi giá trị biến
  + SetTexture: thay đổi một texture mới
* Các phương thức trên chỉ hoạt động khi các thuộc tính có trong Shader của Object. Nghĩa là nếu như shader không sử dụng texture thì việc dùng SetTexture sẽ không thay đổi hay ảnh hưởng đến shader

1. Shaders and Textures

* Shaders
* Shader là một chương trình với các tính toán và chuỗi các câu lệnh để xử lý màu cho các pixel trong phạm vi bao phủ bởi một Object
* Chương trình có đuôi “.shadergraph” chỉ có thể chạy với Universal RP hoặc High Definition RP, với chương trình có đuôi “.shader” có thể xử lý ở nhiều loại render piplines
* Trong Unity, có ít nhất 4 loại cấu trúc để tạo ra một shader, trong đó có vertex shader, fragment shader, surface shader và compute shader
* Các loại Shader:
* Standard surface shader: được tối ưu cho việc tương tác với ánh sáng cơ bản và chỉ hoạt đọng với Build-in RP
* Unlit Shader: không bị ảnh hưởng bởi ánh sáng, là cấu trúc cơ bản để tạo ra các hiệu ứng khác
* Image effect shader: cấu trúc tương đối giống Unlit shader, chủ yếu được sử dịng trong hệ ứng hậu kì trong Build-in RP
* Compute shader: được tính toán trên GPU, bên ngoài các graphic pipeline thông thường
* Ray tracing shader: tính toán liên quan đến ray tracing
* Queue Tags:
* BackGround : từ 0 đến 1499, giá trị mặc định 1000, sử dụng chủ yếu bởi các thành phần rất xa camera
* Geometry: từ 1500 đến 2399, giá trị mặc định 2000, giá trị cơ bản trên Queue sử dụng cho các Opaque Object
* AlphaTest: từ 2400 đến 2699, giá trị mặc định 2450, sử dụng cho các semi-transparent object phải đứng trước Opaque Object và xếp sau Transparent Object
* Transparent: từ 2700 đến 3599, giá trị mặc định 3000, sử dụng cho Tranparent Objects phải để trước các Object khác
* Overlay: từ 3600 đến 5000, giá trị mặc định 4000, thể hiện cho các thành phần đứng hàng đầu trong Scene
* Textures:
* Texture là hỉnh ảnh bitmap tiêu chuẩn được áp dụng vào bề mặt của mesh. Vị trí của texture được xác định bằng phần mềm tạo ra 3D model
* Texture dùng cho:
* 3D models: texture được áo dụng cho Object bằng cách sử dụng Materials. Ta có thể sử dụng hay hay nhiều texture kết hợp chúng để tạo ra sự linh hoạt
* 2D graphics: Sprite được thể hiện bằng các sử dụng texture cho các mesh phẳng mô phỏng tương đối hình dạng của Object
* GUI
* Particles
* Terrain Heightmaps

1. Physics
2. RigidBody

* Rigidbody cung cấp cho chúng ta một phương pháp dựa vào vật lý để kiểm soát vị trí và sự di chuyển của GameObject. Thay vì sử dụng Transform, ta có thể mô phỏng các tương tác vật lý tác động lên GameObject và để hệ thống vật lý tính toàn kết quả
* Các loại:
* Dynamic: bị ảnh hưởng bởi lực và phản hồi với các va chạm. Object có dynamic rigidbody di chuyển dưới sự ảnh hưởng của trọng lực, lức và các tương tác vật lý khác
* Kinematic: không bị ảnh hưởng bởi ngoại lực giúp cho việc điều khiển di chuyển dễ dàng hơn
* Static: không di chuyển, nằm ở vị trí xác định
* Thuộc tính
* Mass: khối lượng của vật
* Drag: thể hiện lực cản không khí tác động lên vật theo đường thẳng
* Angular Drag: thể hiện lực cản không khí ảnh hưởng đến vận tốc góc của vật
* Automatic Center of Mass: hệ thống sẽ tự xác định trọng tâm của Rigidbody phụ thuộc vào hình dạng và độ lớn của Rigidbody
* Automatic Tensor: hệ thống sẽ tự xác định tensor và tensor rotation của Rigidbody phụ thuộc vào tất cả các collider của Object
  + Inertia Tensor: bao nhiêu lực cần để làm cho Rigidbody di chuyển
  + Inertia Tensor Rotation: bao nhiêu lực để làm cho Rigidbody có thể xoay
* Use Gravity: xác định Object có bị tác động bỏi trọng lực không
* Interpolate: đồ hoạ của Unity được tính toán trong Update còn vật lý được xử lý trong FixedUpdate vì vậy thỉnh thoảng chúng sẽ không đồng nhất. Interpolate được dùng trong tình huống chúng ta cần đòng nhất đồ hoạ với vật lý của Object
* None: không áp dung
* Interpolate: sử dụng tư thế và gia tốc của Rigidbody từ 2 lần cập nhật vật lý trước để tính toán và áp dụng cho Rigidbody ở frame hiện tại. Có tính chính xác cao hơn Extrapolate nhưng có độ trễ
* Extrapolate: sử dụng tư thế và gia tốc của Rigidbody từ lần cập nhật vật lý trước và tư thế của Rigidbody trong lần cập nhật vật lý tiếp theo để tính toán và dự đoán tư thế trong frame hiện tại. Có thể không quá chính xác
* Collision Detection:

|  |  |  |
| --- | --- | --- |
| Mode | Hữu dụng cho | Không hữu dụng cho |
| Dicrete | Tương tác của các vật di chuyển chậm | Tương tác của vật di chuyển nhanh |
| Continuous Speculative | Tương tác của vật di chuyển nhanh | Một số các tương tác của vật di chuyển nhanh yêu cầu độ chính xác cao |
| Continuous | Tương tác tuyến tính của vật di chuyển nhanh với độ chính xác cao  Vật chỉ tương tác với Static Collider | Tương tác xảy ra do kết quả của việc xoay của cơ thể vật lý  Cơ thể vật lý tác động với Collider di chuyển |
| Continuous Dynamic | Tương tác tuyến tính của vật di chuyển nhanh với độ chính xác cao  Cơ thể vật lý tác động với Collider di chuyển | Tương tác xảy ra do kết quả của việc xoay của cơ thể vật lý |

* Constraints: hạn chế sự di chuyển của Rigidbody
* Freeze Position: hạn chế sự di chuyển của Rigidbody theo các chiều X,Y và Z
* Freeze Rotation: hạn chế góc xoay của Rigidbody quanh các trục X, Y và Z

1. Collider

* Các loại của Collider:
* 3D:
  + Box Collider
  + Sphere Collider
  + Capsule Collider
  + Wheel Collider
  + Mesh Collider
  + Terrain Collider
* 2D:
  + Circle Collider 2D
  + Box Collider 2D
  + Polygon Collider 2D
  + Edge Collider 2D
  + Capsule Collider 2D
  + Composite Collider 2D
  + Custom Collider 2D
* Các đặc điểm
* Layer: Unity cho phép chúng ta định nghĩa các collision layers và có thể tuỳ chỉnh các layers nào có thể tương tác với nhau
* Trigger: dùng để xác định sự va chạm mà không ảnh hưởng đến tính chất vật lý của chúng
* Dynamic Collider Adjustment: các collider có thể thay đổi trong thời gian chạy
* Compound Colliders: có thể sử dụng nhiều các collier trong một Object giúp việc tạo ra collider phức tạp bằng những collider đơn giản
* Collision Event:
* OnCollisionEnter: được gọi khi collider/rigidbody bắt đầu va chạm với collider/rigidbody khác
* OnCollisionExit: được gọi khi collider/rigidbody kết thúc va chạm với collider/rigidbody khác
* OnCollisionStay: được gọi mỗi frame cho mỗi Collider/Rigidbody chạm Collider/Rigidbody khác
* OnTriggerEnter: khi một Object va chạm với một Trigger
* OnTriggerExit: khi một Object ngừng va chạm với một Trigger
* OnTriggerStay: được gọi gần như tất cả các frame cho mỗi Collider chạm Trigger
* Tương tác giữa các loại colldier phát ra detection messages
  + Có ít nhất một dynamic collider

|  |  |  |  |
| --- | --- | --- | --- |
|  | Static Collider | Dynamic Colldier | Kinematic Collider |
| Static Collider |  | X |  |
| Dynamic Collider | X | X | X |
| Kinematic Collider |  | X |  |

* Tương tác phát ra trigger messages

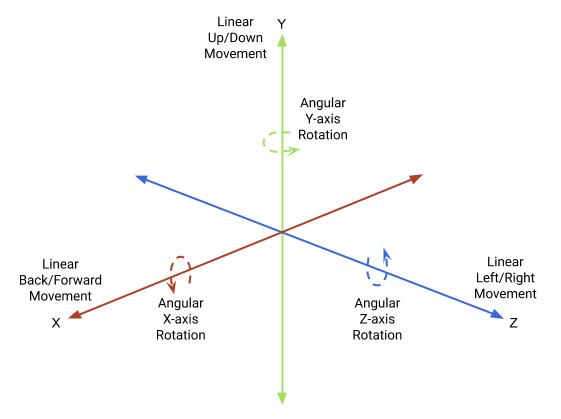
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Static Collider | Dynamic Collider | Kinematic Collider | Static Trigger Collider | Dynamic Trigger Collider | Kinematic Trigger Collier |
| Static Collider |  |  |  |  | X | X |
| Dynamic Collider |  |  |  | X | X | X |
| Kinematic Collider |  |  |  | X | X | X |
| Static Trigger Collider |  | X | X |  | X | X |
| Dynamic Trigger Collider | X | X | X | X | X | X |
| Kinematic Trigger Collider | X | X | X | X | X | X |

1. Physic materials

* Dynamic Friction: hệ số ma sát dùng khi di chuyển khi tương tác với collider khác
* Static Friction: hệ số ma sát dùng khi vật đứng yên trong khi tương tác với collider khác
* Bounciness: độ nảy của bề mặt. Với giá trị bằng 0 thì bề mặt sẽ không nảy, với giá trị bằng 1 thì nó sẽ nảy mà không mất đi năng lượng
* Friction Combine: xác định cách mà lực ma sát được tính toán
* Bounce Combine: xác định cách mà độ nảy được tính toán
* Cách tổng hợp giá trị của Collider surface:
  + Maximum: giá trị lớn giữa hai giá trị
  + Multiply: sử dụng tổng của một giá trị nhân với giá trị khác
  + Minimum: giá trị nhỏ hơn giữa hai giá trị
  + Average: giá trị trung bình của hai giá trị
  + Unity dùng sự ưu tiên khi có sự tương tác giữa hay collider có cài đặt kết hợp khác nhau. Maximum > Multiply > Minimim > Average

1. Physic joint

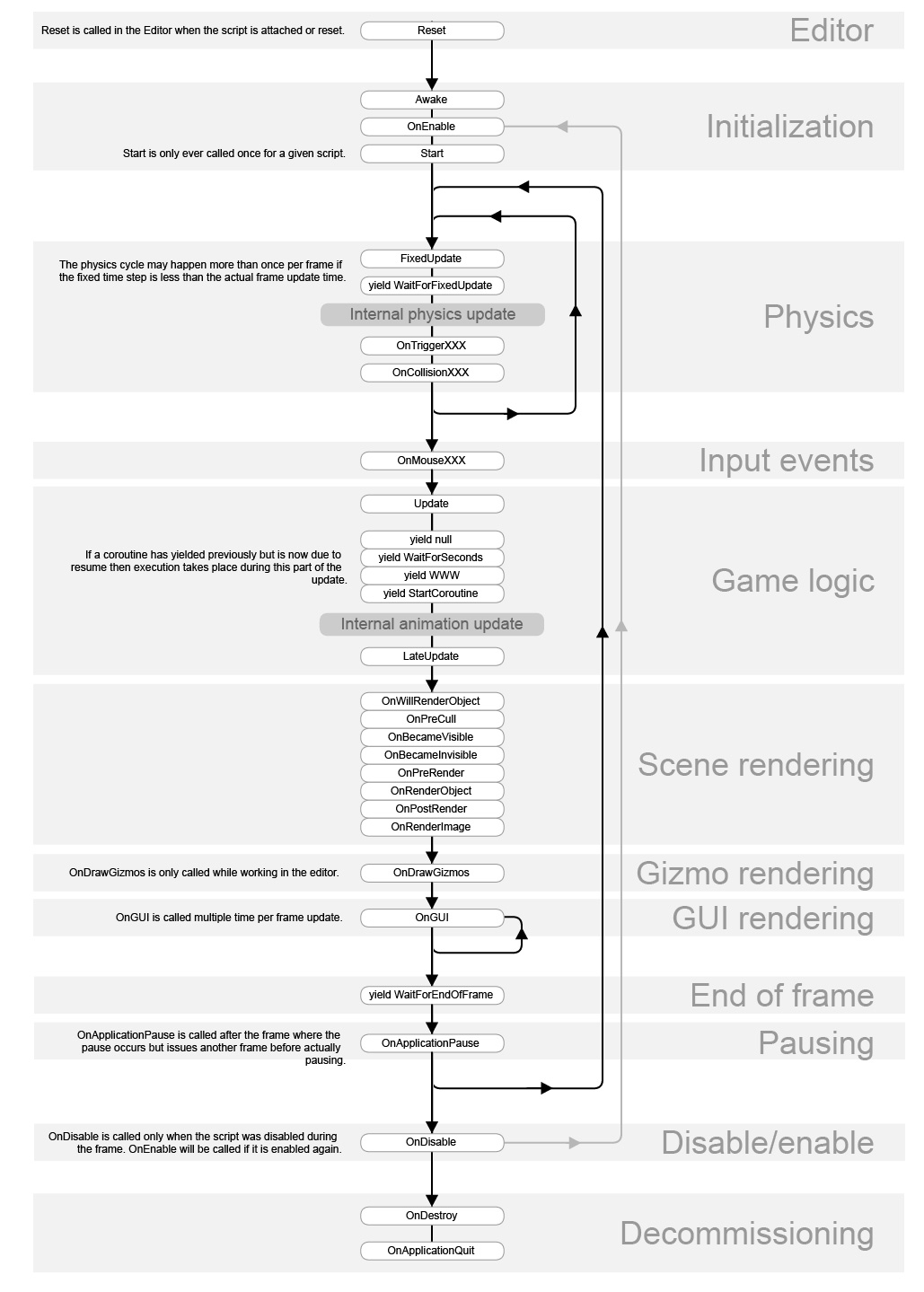
* Các loại Joint:
* 3D:
* CharacterJoint: mô phỏng một khớp bóng như hông hay vai. Hạn chế chuyển động của cơ thể theo các hướng và cho phép các góc quay. Rigidbody gắn với CharacterJoint quay quanh mỗi trục và trục quay từ một nguồn chung
* HingeJoint: gắn một Rigidbody với một Rigidbody khác hay một điểm ở trong không gian với cùng một nguồn góc cho phép Rigidbody quay xung quanh một trục nhất định từ nguồn đó
* SpringJoint: giữ Rigidbody ở một khoảng cách nhưng để cho khoảng cách giữa chúng có thể duỗi ra một ít. Lo xò hoạt động như một miếng đàn hồi cố gắng kéo hai điểm về lại vị trí chính xác ban đầu
* FixedJoint: hạn chế di chuyển của Rigidbody để đi theo sự di chuyển của Rigidbody gắn với nó
* ConfigurableJoint: mô phỏng khớp xương, như là ragdoll. Ta có thể chỉnh sửa để bắt buộc hay hạn chế sự di chuyển của Rigidbody trong bất cứ góc nào
* 2D:
  + Distance Joint 2D: kết nối hai GameObject kiểm soát bỏi Rigidbody với nhau trong khi giữ nguyên khoảng cách của chúng
  + Fixed Joint 2D: giữ hai Object ở một vị trí nhất định đối với nhau sao cho khoảng cách và độ lệch của chúng luôn được giữ nguyên
  + Friction Joint 2D: giảm vận tốc và vận tốc góc giữa hai GameObject kiểm soát bởi Rigidbody về 0
  + Hinge Joint 2D: cho phép một GameObject kiểm soát bởi Rigidbody gắn với một điểm trong không gian xung quanh mà nó có thể xoay
  + Relative Joint 2D: cho phép hai GameObject kiếm soát bởi Rigidbody duy trì một vị trí phụ thuộc vào vị trí của GameObject khác.
  + Slider Joint 2D: cho phép một GameObject kiểm soát bởi Rigidbody trượt dọc theo một đường trong không gian
  + Spring Joint 2D: cho phép hai GameObject kiểm soát bởi Rigidbody phản ứng lại như chúng được liên kết với nhau bởi một lò xo
  + Target Joint 2D: kết nối với một mục tiêu xác định không phải một Rigidbody Object khác như các joint khác. Nó là Joint dạng lò xo
  + Wheel Joint 2D: mô phỏng trạng thái của một bánh xe quay
* Thuộc tính cơ bản



* Edir Joint Angular Limits: thêm các Gizmo vào Scene để giúp ta thay đổi các giới hạn góc của các Joints
* Connected Body: Rigidbody Object mà Joint liên kết. Ta có thể để là None để thể hiện Joint liên kết với một vị trí nhất định trong không gian thay vì liên kết với một Rigidbody khác
* Anchor: vị trí điểm thể hiện trung tâm của Joint. Tất cả các tính toán vật lý sử dụng điểm này như trung tâm của các sự tính toán
* Axis: xác định trục xoay tự nhiên dựa trên mô phỏng vật lí
* Auto Configure Connected Anchor: bật chức năng này thì sẽ tự động tính toán vị trí Connected Anchor để phù hợp với vị trí toàn thể
* Connected Anchor: thay đổi vị trí của Connected Anchor bằng tay
* Secondary Axis: Axis và Secondary Axis xác định hệ toạ độ của bản thân. Trục thứ ba vuông góc với hai trục còn lại
* X, Y, Z Motion: đặt sự di chuyển dọc theo trục X, Y hoặc Z là Free, Completely Locked hay Limited tương ứng với các hạn chế được thể hiện ở dưới.
* Angular X, Y, Z Motion: đặt việc xoay quanh các trục X, Y hoặc Z là Free, Completely Locked hay Limited tương ứng với các hạn chế được thể hiện ở dưới
* Linear Limit Spring: áp dụng lực đàn hồi để đẩy vị trí của Object về lại khi nó đi quá vị trí giới hạn
  + Spring: lực đàn hổi. Đặt giá trị này bằng 0 sẽ khiến giới hạn không thể đi qua. Với giá trị khác 0 khiến giới hạn co giãn
  + Damper: giảm lực đàn hồi lỉ lệ với tốc độ di chuyển của Joint. Với giá trị lớn hơn 0 cho phép Joint làm suy yếu những dao động. Với giá trị nhỏ 0 những dao động sẽ tiếp tục vô hạn
* Linear Limit: đặt giới hạn di chuyển tuyến tính của Joint
  + Limit: khoảng cách từ vị trí bắt đầu đến vị trí giới hạn
  + Bounciness: đặt lực nảy áp dụng nên Object để kéo nó lại khi nó đi đến vị trí giới hạn
  + Contact Distance: vị trí nhỏ nhất cho phép giữa vị trí của Joint và giới hạn. Độ chấp nhận cao nghĩa là giới hạn khó bị vi phạm khi vật di chuyển nhanh nhưng nó sẽ khiến cho việc mô phỏng vật lí diễn ra nhiều hơn khiến cho việc giảm hiệu xuất
* Angular X Limit Spring: áp dụng mo men xoắn để quay Object trở lại khi nó đi quá giới hạn góc của Joint
  + Spring: mô men xoắn. Đặt nó bằng 0 sé khiến giới hạn không thể phá vỡ. Với giá trị khác sẽ khiến giới hạn này co giãn
  + Damper: sự giảm của mô men xoắn tỉ lệ với tốc độ góc của Joint. Với giá trị lớn hơn 0 sẽ là cho Joint suy yếu dần các dao động. Với giá trị nhỏ hơn 0 các dao động sẽ tiếp tục vô hạn
* Low Angular X Limit: giới hạn góc dưới của Joint quanh trục X, được xác định là một góc từ góc bạn đầu của Joint
  + Limit: góc giới hạn
  + Bounciness: đặt mô men phản áp dụng nên Object khi góc quay chạm đến góc giới hạn
  + Contact Distance: góc nhỏ nhất cho phép giữa vị trí của Joint và giới hạn. Độ chấp nhận cao nghĩa là giới hạn khó bị vi phạm khi vật di chuyển nhanh nhưng nó sẽ khiến cho việc mô phỏng vật lí diễn ra nhiều hơn khiến cho việc giảm hiệu xuất
* High Angular Xlimit: tương tự như Low Angular X Limit nhưng dùng để xác định vị trí giới hạn góc trên của Joint
* Angular YZ Limit: tương tự như Angular X Limit Spring nhưng thay vào đo xoay quanh trục Y và Z
* Angular Y Limit: tương tự như Angular Y Limt
* Angular Z Limit: tương tự như Angular Z Limit
* Target Position: vị trí mục tiêu mà lực của Joint di chuyển đến
* Target Velocity: vận tốc mong muốn mà Joint di chuyển đến Target Position dưới tác động của lực đẩy
* Xdrive: đặt lực mà Unity sử dụng để xoay Joint xung quanh trục bằng Position Spring và Position Damper. Biến Maximum Force hạn chế lực này
  + Position Spring: mô men xoắn mà Unity dùng để quay Joint từ vị trí hiện tại đến vị trí xác định
  + Position Damper: sự giảm của mô men xoắn tỉ lệ với tốc độ góc của Joint. Với giá trị lớn hơn 0 sẽ là cho Joint suy yếu dần các dao động. Với giá trị nhỏ hơn 0 các dao động sẽ tiếp tục vô hạn
  + Maximum Force: giới hạn lực mà chuyển động có thể áp dụng.
* Angular YZDrive: tương đương với Angular X Drive nhưng áp dụng cho trục Y và Z
* Slerp Drive: lực xoắn truyền vào các Joint quay xung quanh các trục cục bộ của nó. Chỉ có thể sử dụng nếu Rotation Drive Mode được đặt là Slerp
  + Position Spring: mô men xoắn mà Unity dùng để quay Joint từ vị trí hiện tại đến vị trí xác định
  + Position Damper: sự giảm của mô men xoắn tỉ lệ với tốc độ góc của Joint. Với giá trị lớn hơn 0 sẽ là cho Joint suy yếu dần các dao động. Với giá trị nhỏ hơn 0 các dao động sẽ tiếp tục vô hạn
  + Maximum Force: giới hạn lực mà chuyển động có thể áp dụng.
* Projection Mode: cách mà các Joint quay trở lại với các hạn chế của nó khi hạn chế bị vượt qua một cách bất ngờ, bởi vì physics engine không thể kết hợp các tổng hợp lực hiện tại trong quá trình mô phỏng
* Projection Distance: khoảng cách mà Joint phải vượt qua để physics engine kéo nó lại
* Projection Angle: góc mà Joint phải vượt qua để physics engine kéo nó trở lại
* Configured in World Space: bật tính năng này sẽ đặt các thược tính của mục tiêu và lực trong world space thay vì vị trí cục bộ không gian của Object
* Swap Bodies: cho phép thay đổi thức tự physics engine ảnh hưởng đến Rigidbody liên quan đến Joint.s
* Break Force: nếu lực lớn hơn giá trị này thì Joint sẽ bị phá vỡ và bị xoá
* Break Torque: nếu mo men xoắn lớn hơn giá trị này thì Joint sẽ bị phá vỡ
* Enable Collision: cho phép Object collide với Object mà nó kết nối với nhau. Nếu tắt thì Joint và Object sẽ xuyên qua nhau
* Enable Preprocessing: nếu preprocessing bị tắt, các cấu hình không thể của joint được giữ ổn định hơn thay vì di chuyển mất kiểm xoát
* Mass Scale: tỉ lệ áp dụng đến phản lực và tensor quán tính của Rigidbody từ 0,00001 tới vô hạn
* Connected Mass Scale: tỉ lệ áp dụng đến phản lực và tensor quán tính của Rigidbody kết nối từ 0,00001 đến vô hạn

1. Scripting

* Vòng đời của script:



* Tải Scene đầu tiền: các hàm được gọi khi scene bắt đầu
  + Awake: luôn luôn được gọi trước bất của một hàm Start. Nếu GameObject không được bật trong lúc khởi động thì Awake chỉ được gọi khi GameObject được bật trở lại
  + OnEnable (chỉ gọi khi Object được bật): hàm này được gọi ngay sau khi Object được kích hoạt
* Editor:
  + Reset: được gọi đẻ khởi tạo các thuộc tính của script khi nào được gắn vào một Object lần đầu tiên và cũng là lúc lệnh Reset được dùng
  + OnValidate: được gọi mỗi khi thuộc tính của script được đặt, bao gồm các đối tượng được mã hoá
* Trước lần cập nhật frame đầu tiền:
  + Start: được gọi trước khi cập nhập frame đầu tiên và chỉ khi script được bật
* Giữa các frame:
  + OnApplicationPause: được gọi ở lúc kết thúc của frame khi sự dừng lại được xác định, hữu dụng giữa các cập nhật frame bình thường
* Thứ tự cập nhật:
  + FixedUpdate: FixedUpdate thường được gọi nhiều hơn Update. Tất cả các tính toán và cập nhật vật lí được xảy ra ngay sau FixedUpdate. Không phụ thuộc vào tốc độ khung hình
  + Update: được gọi một lần cho mỗi frame
  + LateUpdate: được gọi một lần cho mỗi frame sau khi Update kết thúc. Tất cả các tính toàn ở Update sẽ kết thúc trước khi LateUpdate bắt đầu
* Vòng lặp cập nhật Animation:
  + OnStateMachineEnter: trong khi bước State Machine Update, được gọi trong lần cập nhật frame đầu tiên khi State Machine tạo một sự chuyển đổi thông qua trạng thái Entry. Sẽ chỉ xảy ra nếu có một Controller Component (VD: AnimationController, AnimatorOverrideController, AnimatorControllerPlayeable)
  + OnStateMachineExit: trong bước State Machine Update, nó được gọi ở lần cập nhật frame cuối cùng tiên khi State Machine tạo một sự chuyển đổi thông qua trạng thái Exit. Sẽ chỉ xảy ra nếu có một Controller Component (VD: AnimationController, AnimatorOverrideController, AnimatorControllerPlayeable)
  + Fire Animation Events: gọi tất cả các sự kiện animation từ tất cả các mẫu clip từ thời điểm cập nhật cuối cùng đến thời điểm cập nhật hiện tại
  + StateMachineBehaviour (OnStateEnter/ OnStateUpdate/ OnStateExit): layer có thể có 3 trạng thái hoạt động: current state, interrupted state và next state. Hàm được gọi cho mỗi trạng thái hoạt động với một StateMachineBehaviour có gọi đến OnStateEnter, OnStateUpdate hay OnStateExit
  + OnAnimatorMove: mỗi một lần cập nhật frame, nó sẽ được gọi một lần cho mội Animator component để chỉnh sửa Root Motion
  + StateMachineBehaviour(OnStateMove): được gọi cho mỗi trạng thái hoạt động với một StateMachineBehaviour định nghĩa nó
  + OnAnimationIK: thiết lập animation IK. Nó được gọi mỗi lần cho mỗi Animator Controller layer có IK pass được bật. Chỉ được thực hiện nếu ta sử dụng Humanoid rig
  + StateMachineBehaviour(OnStateIK): được gọi cho mỗi trạng thái hoạt động với một StateMachineBehaviour định nghĩa nó trên một layer có IK pass được bật
  + WriteProperties: viết tất cả các thuộc tính lên Scene từ luồng chính
* Rendering:
  + OnPreCull: được gọi trước khi Camera cull
  + OnBecomeVisiable/OnBecomeInvisible: được gọi khi một vật có thể nhìn thấy/ không nhìn thấy đối với một camera
  + OnWillRenderObject: được gọi một lần cho mỗi camera nếu Object được nhìn thấy
  + OnPreRender: được gọi trước khi camera bắt đầu render
  + OnRenderObject: được gọi sau khi các quá trình render thông thường đã xong
  + OnPostRender: được gọi khi camera kết thúc việc render
  + OnRenderImage: được gọi sau khi render khung hình được thực hiện xong cho phép xử lí hậu kì của hình ảnh
  + OnGUI: được gọi nhiều lần mỗi một frame để phản hồi lại các sự kiện GUI
  + OnDrawGizmos: sử dụng để vẽ Gizmos trong màn hình
  + Chỉ hoạt động ở Built-in Render Pipeline
* Coroutines:
  + yield: coroutine sẽ tiếp tục khi tất cả các hàm Update được gọi cho frame tiếp theo
  + yield WaitForSeconds: tiếp tục sau một khoảng thời gian xác định sau tất cả các hàm Update được gọi cho frame
  + yield WaitForFixedUpdate: tiếp tục sau tất cả các FixedUpdate được gọi trên tất cả các script. Nếu nó được gọi trước FixedUpdate thì nó sẽ tiếp tục sau FixedUpdate ở frame hiện tại
  + yield WWW: tiếp tục sau khi WWW tải hoàn thành
  + yield StartCoroutine: liên kết chuỗi Coroutine và sẽ đợi cho đến khi Coroutine được gọi hoàn thành trước
* Khi Object bị phá huỷ:
  + OnDestroy: được gọi tất cả các lần cập nhật frame khi nó ở trong frame cuối cùng mà nó tồn tại
* Khi kết thúc:
  + OnApplicationQuit: được gọi khi tất cả các GameObject trước khi ứng dụng kết thúc. Khi ở editor thì được gọi khi người dùng dừng playmode
  + OnDisable: được gọi khi behaviour bị tắt hoặc không được kích hoạt
* MonoBehaviour: khi được gắn vào một Object nó sẽ tham gia vào vòng cập nhật của game và cho phép code được chạy mỗi frame
* Callbacks: (VD: Start, Update, Awake, LateUpdate,…) Unity không đảm bảo thứ tự của các scripts khác nhau có phương thức hoạt động tuy vậy ta vẫn có thể sắp xếp thứ tự của chúng trong Settings với Script Executio Order
* Coroutines: dùng cho đoạn code hoạt động trong nhiều frame. Coroutine là một phương thức có thể tạm thời trả về hay yield và phần còn lại có thể được tiếp tục sau đó. Coroutine là một phương thức trả về IEnumerator và bắt đầu bằng StartCoroutine
* ScriptableObject: là nơi lưu trữ lượng lớn dữ liệu mà không phu thuộc vào các class khác. Dùng chủ yếu để làm giảm dung lượng bộ nhớ bằng cách tránh sao chép lại các giá trị cũ
* Singletons: một class duy nhất mà các scripts có thể luôn luôn truy cập như các biến toàn cục.

1. Audio
2. Audio Source

* Sử dụng để phát âm thanh trong trò chơi
* Âm thanh được chạy là 3D hay 2D phục thuộc vào AudioImporters
* Gắn vào một Object để phát âm thanh từ đó
* Có thể chạy một âm thành bằng Play, Pause và Stop. Ta cũng có thể thay đổi âm lượng của nó bằng cách sử dụng “volumn” hay di chuyển đến một thời điểm cụ thể trong audio bằng cách sử dụng “time”
* Có thể chạy nhiều âm thanh trên một AudioSource bẳng cách sử dụng PlayOneShot

1. Audio Listener

* Chịu trách nhiệm cho việc nghe âm thanh
* Tương tự như người nghe trong không gian 3D
* Trong scence chỉ có thể có một Audio Listener

1. Audio Clip

* Tệp chưa dữ liệu âm thanh được sử dụng bởi Audio Source để phát
* Unity hỗ trợ các tệp âm thanh mono, stereo và multichannel lên đến tám channel. Ta có thể import các định dạng âm thanh sau: .aif, .wav, .mp3 và .ogg. Unity cũng hỗ trợ việc import các module tracker trong các định dạng .xm, .mod, .it hay .s3m
* Các thuộc tính:
  + ambisonic: xác định đoạn audio có phải là ambisonic
  + channels: số lượng channel của audio
  + frequency: tần số mẫu của đoạn âm thanh
  + length: độ dài của audio
  + loadInBackGround: quá trình tải sẽ diễn ra chậm để tránh chặn nguồn chính
  + loadState: trạng thái tải hiện tại của dữ liệu âm thanh
  + loadType: tải loại của clip
  + preloadAudioData: tải trước dữ liệu của Audio trước khi Audio Clip được tải
  + samples: độ dài của audio theo đơn vị mẫu

1. Animation
2. Animator

* Controller: Animator Controller gắn với Object
* Avatar: Avatar của Character (Sử dụng để animate nhân vật dạng người)
* Apply Root Motion: kiểm soát vị trí và góc của Object từ animation hay từ script
* Update Mode: lựa chọn khi nào Animator cập nhật
* Normal: Animator cập nhật cùng với Update, vận tốc tương ứng với timescale hiện tại
* Animate Physics: Animator cập nhật cùng với FixedUpdate
* Unscaled Time: Animator cập nhật cùng với Update, vận tốc không bị ảnh hưởng bởi biến timescale
* Culling Mode:
  + Always Animate: luôn luôn animate, không culling kể cả khi tắt màn hình
  + Cull Update Transforms: Retarget, IK và sự thay đổi của Transforms bị tắt khi renderers không nhìn
  + Cull Completely: animation tắt hoàn toàn khi renderers không nhìn
* Animation curve information: mỗi animation clip chưa dữ liệu ở dạng “curves”, thể hiện dữ liệu thay đổi như thế nào theo thời gian. Nó có thể thể hiện vị trí góc quay của một Object hay thay đổi giá trị trong clip như là thay đổi màu sắc của material
  + Clip Count: số lượng các Animation Clips được sử dụng bở Animator Controller gắn với Animator
  + Curves (Pos, Rot & Scale): số lượng của các curve Unity dùng để animate vị trí, góc và độ lớn của GameObject
  + Muscles: số lượng curve của muscle dùng để animate GameObject dạng người của chính Animator. Những curve này được dùng để animate các Humanoid Avatar chuẩn
  + Generic: số lượng đường cong dạng dố được sử dụng bởi animator
  + PPtr: số lượng sprite animation curve
  + Curves Count: tất cả số lượng các animation curve
  + Constant: số lượng animation curve được tối ưu như giá trị không đổi. Unity tự động chọn nếu animation có chứa curve với giá trị không đổi
  + Dense: số lượng curve được tối ưu để sử dụng phương pháp “dense” để lưu trữ dữ liệu. Sử dụng ít dung lượng hơn đáng kể phương pháp “stream”
  + Stream: số lượng curve được tối ưu để sử dụng phương pháp “stream” để lưu dữ liệu.

1. Animation Clip

* Animation Clip hỗ trợ việc import animation từ các nguồn ngoài bao gồm
  + Humanoid animation được tạo bởi motion capture studio
  + Animation được từ các ứng dụng 3D
  + Từ các thư viện của bên thứ ba
* Tạo và chỉnh sửa Animation Clip trong Unity
  + Vị trí, góc và độ lớn của GameObject
  + Thuộc tính của các Component như màu của material, độ sáng ở nguồn sáng, đọ lớn của âm thanh
  + Thuộc tính trong script bao gồm float, integer, enum, vector hay boolean
  + Thời gian gọi các hàm trong script

1. Animation Controller

* Thông tin cơ bản
* Cho phép ta sắp đặt và duy trì tập animation cho một nhân vật hay các GameObject
* Animation Controller có các tham chiếu dẫn đến các animation clip được sử dụng bên trong nó và kiểm soát nhiều trạng thái khác nhau của animation, sự chuyển đổi giữa chúng được gọi là State Machine
* Animator Controller Asset: được tạo bởi Unity cho phép ta duy trì tập các animation của nhân vật hay Object.
* Animation State Machine:
* Thông thường một nhân vật hay một Game Object có nhiều animation khác nhau tương đương với hành động nó có thể thể hiện trong game. Mecanim sử dụng dạng của một biểu đồ luồng để biểu hiện State Machine cho phép ta kiếm soát và nối tiếp các animation clip mà ta muốn sử dụng
* State Machine bao gồm State, Transition, Event là các Sub State Machine nhỏ hơn có thể sử dụng như một phần của một State Machine lớn hơn.
* Animation Parameter là biến được khai bảo ở trong Animator Controller có thể truy cập và thay đổi giá trị từ script. Đây là cách script kiểm soát và ảnh hướng đến tiến trình của State Machine. Ta có 4 kiểu cơ bản:
  + Integer: sử dụng SetInteger để truy cập
  + Float: sử dụng SetFloat để truy cập
  + Bool: sử dụng SetBool để truy cập
  + Trigger: sử dụng SetTrigger và ResetTrigger để truy cập
* Entry node được sử dụng khi truyền vào một State Machine. Entry node sẽ đánh giá và rẽ nhánh đến trạng thái dích phụ thuộc vào tệp các điều kiện. Bằng cách đó entry node có thể kiếm soát trạng thái bắt đầu của State Machine bắt đầu sau khi kiếm tra trạng thái của các biến khi State Machine được bắt đầu
* Exit node được sử dụng khi muốn chỉ ra nơi mà State Machine nên kết thúc
* State Machine Behaviour: là một class đặc biệt của script. Ta có thể gắn nó với bất kì một trạng thái độc lập nào bên trong State Machine. Nó cho phép ta có thể viết code thực thi khi State Machine đi vào kết thúc hay ở trong các trạng thái đặc biệt
* Sub-State Machine: thông thường các nhân vật có các hành động phức tạp bao gồm nhiều giai đoạn. Thay vì xử lí toàn bộ hành động bằng một trạng thái duy nhất, ta có thể phân chia các giai đoạn khác nhau và sử dụng các trạng thái khác nhau cho mỗi giai đoạn. Mặc dù hữu dụng cho việc kiểm soát nhưng việc phân chia cũng làm cho State Machine trở nên lớn hơn và khó kiếm soát hơn nếu có nhiều các hành động phức tạp. Mecanim cho phép chúng ta gộp các state thành một nhóm với một tên duy nhất trong biểu đồ State Machine. Nhóm đó được gọi là Sub-State Machine
* Animation Layer: được sử dụng để quản lý các State Machine phức tạp cho các bộ phận khác nhau của cơ thể.
* TargetMatching: thông thường trong game, trường hợp khi nhân vật phải di chuyển sao cho bàn tay hoặc bàn chân đặt lên một nơi nhất định vào một khoảng thời gian nhất định, để xử lí vấn đề này ta có thể sử dụng hàm Animator.MatchTarget.
* Inverse Kinematic:
  + Phần lớn các animation được thể hiện bằng cách xoay góc của các Joint tới một giá trị được xác định trước. Vì vậy giá trị vị trí của Joint con thay đổi theo góc của cha của nó do đó điểm kết của chuỗi các Joint có thể xác định từ các góc và vị trí tương đối của các joint mà nó chứa. Phương pháp này được gọi là forward kinematics
  + Nếu bạn muốn tạo ra một nhân vật có khả năng chạm vào vật ở một điểm được chọn bởi người dùng và đặt bàn chân một cách hợp lý trong một mặt phẳng không đồng đều thì ta cần làm ngược lại forward kinematics cho các điểm được chọn trong không gian, truy ngược lại để tìm cách thích hợp để xoay các Joint sao cho điểm kết thúc đặt đúng điểm được chỉ định. Phương pháp này được gọi là Inverse Kinematics

1. AvatarMask

* Mask cho phép bỏ đi một phần dữ liệu của animation cho phép animate tuỷ chỉnh các phần của Object thay vì toàn bộ tất cả Object
* Humanoid: các bộ phận cơ thể bảo gồm Head, Left Arm, Right Arm, Left Hand, Right Hand, Left Leg, Right Leg và Root( được định dạng bằng bóng ở dưới chân). Ta có thể chọn vào các bộ phận để chọn xem nó có được bao gồm animation hay IK để xác định xem IK có được bao gồm trong animation
* Transform: Nếu animation sử dụng Humanoid avatar hoặc bạn muốn kiểm soát chi tiết hơn mỗi xương. Mỗi xương có checkbox thể hiện nó có thể được bao gồm trong animation hay không

1. Navigation

* Navigation Mesh là thành phần chủ yếu để định nghĩa khu vực trong game mà các thực thể trong game có thể di chuyển. Trước khi sử dụng các tính toán không gian, ta cần bake NavMesh trước
* NavMeshAgent: là thành phần gắn vào vật chuyển động cho phép nó có thể khảo sát Scence sử dụng NavMesh
  + Các thuộc tính:
    - acceleration: gia tốc tối đa của agent
    - agentTypeID: type ID của agent
    - angularSpeed: vận tốc tối đa khi quay
    - areaMask: chỉ định khu vực NavMesh nào có thể đi qua được
    - autoBraking: agent có nên tự động dừng để tránh làm quá tải điểm đến
    - autoRepath: agent có nên tự động tìm đường thay thế nếu đường hiện tại trở nên không thể đi
    - autoTraverseOffMeshLink: agent có nên tự động di chuyển dọc OffMeshLink
    - avoidancePriority: mức độ ưu tiên tránh né
    - baseOffset: sự xê dịch theo chiều dọc của GameObject sở hữu
    - currentOffMeshLinkData: OffMeshLinkData hiện tại
    - desiredVelocity: vận tốc mong muốn bao gồm các nguồn tiềm năng
    - destination: tìm hoặc cố để xác đinh vị trí của agent bằng đơn vị không gina
    - hasPath: agent hiện tại có đường đi hay không
    - height: chiều cao của agent dùng cho việc đi qua phía dưới của vật chặn
    - isOnNavMesh: agent liệu có ràng buộc với navmesh hay không
    - isOnOffMeshLink: agent có đang đứng ở trên OffMeshLink hay không
    - isPathStale: liệu đường hiện tại có còn hiệu quả hay không
    - radius: phạm vi né tránh của agent
    - remainingDistance: khoảng cách giữa vị trí của agent và địa điểm cần đến trên đường hiện tại
    - speed: vận tốc tối đa khi đi theo con đường
    - steeringTarget: mục tiêu lái hiện tại dọc theo con đường
    - stoppingDistance: dừng trong khoảng cách này từ vị trí của mục tiêu
    - updatePosition: lấy hoặc đặt liệu vị trí có đồng nhất với vị trí mô phỏng của agent
    - updateRotation: liệu agent có cập nhật hướng
    - updateAxis: cho phép ta có thể định rõ agent có được căn chỉnh theo trục lên của NavMesh hay kết nối với nơi nó được đặt lên
    - velocity: truy cập vào vận tốc hiện tại của NavMeshAgent hoặc đặt một vận tốc để kiểm soát agent
* Các phương thức:
  + ActivateCurrentOffMeshLink: bật hoặc tắt OffMeshLink hiện tại
  + CalculatePath: tính toán đường đi đến một điểm nhất định và lưu kết quả
  + CompleteOffMeshLink: hoàn thành sự di chuyển trên OffMeshLink hiện tại
  + FindClosestEdge: xác định khoảng cách gần nhất tới cạnh của NavMesh
  + GetAreaCost: lấy giá cho việc tính toán đường đi khi đi qua một địa điểm đặc biệt
  + Move: áp dụng di chuyển tương đối lên vị trí hiện tại
  + RayCast: chiếu một đường thẳng tới vị trí của đối tượng trong NavMesh mà không di chuyển vị trí của agent
  + ResetPath: xoá bỏ đường đi hiện tại
  + SamplePathPosition: lấy một vị trí trên đoạn đường hiện tại
  + SetAreaCost: quy định giá trị khi đi qua khu vực của một loại khu vực
  + SetDestination: quy định hay cập nhật địa điểm cần đến vì vậy dẫn đến việc tính toán cho đường mới
  + SetPath: gán đường mới cho agent
  + Warp: dịch chuyển agent đến vị trí cho trước
* NavMeshObstacle: có thể di chuyển xung quanh bề mặt của NavMesh với một gia tốc xác định. Mặc định, vật cản sẽ chỉ ảnh hưởng đến sự tránh né của agent hơn là đến việc tìm đường, tức là agent sẽ bỏ qua vật cản khi vẽ đường đi nhưng sẽ di chuyển xung quanh khi di chuyển dọc theo đường đi. Nếu “carving” được bật, vật cản sẽ tạo ra một hố trong NavMesh. Hố đó sẽ được nhận diện bởi việc tìm đường, nên đường sẽ được vẽ để tránh vật cản. Nghĩa là nếu một vật cản cản một rãnh hẹp, đường đi sẽ tìm một con đường khác dẫn đến mục tiếu. Nếu không có “carving” agent sẽ đến vị trí của rãnh mà không thể đi qua cho đến khi vật cản được phá huỷ hoặc di chuyển
  + Thuộc tính:
    - carveOnlyStationary: liệu nó có được carved khi nó đang di chuyển
    - carving: liệu vật cản có để lại một bỏ đi trong NavMesh
    - carvingMoveThreshold: ngưỡng khoảng cách cho việc cập nhật một hố di chuyển
    - carvingTimeToStationary: thời gian để có thể xác định vật cản có đứng yên hay không
    - center: trung tâm của vật cản
    - height: chiều cao của vật cản
    - radius: bán kính của vật cản
    - shape: hình dạng của vật cản
    - size: độ lớn của vật cản
    - velocity: vận tốc khi di chuyển xung quanh NavMesh
* NavMeshPath: đường được thể hiện bằng một dãy các điểm được lữu trữ
* OffMeshLink: đường dẫn cho phép sự di chuyển bên ngoài mặt phẳng của navigation mesh
  + Thuộc tính:
    - activated: đường dẫn có bật
    - area: chỉ số khu vực của NavMesh cho OffMeshLink
    - autoUpdatePositions: tự động cập nhật điểm kết thúc
    - biDirectional: đường dẫn có thể di chuyển qua cả hai chiều
    - costOverride: thay đổi giá trị của đường dẫn
    - endTransform: transform đại diện cho vị trí cuối cùng của liên kết
    - occupied: liệu đường dẫn có đang được sử dụng
    - startTransform: transform đại diện cho vị trí đầu tiên của liên kết

1. UI

* RectTransform: khác với các Object khác sử dụng Transform, UI Objects sử dụng RectTransform để có thể kiểm soát nhiều hơn vị trí của nó liên quan đến các vật khác. Một số các tính toán liên quan đến RectTransform được thực hiện ở cuối của frame, ngay trước khi tính toán UI verticy, để cho chúng luôn đảm bảo chúng luôn cập nhật với tất cả các thay đổi gần nhất xuyên suốt trong frame
* Thuộc tính:
  + Pos (X, Y, Z): vị trí pivot của hình chư nhật tương ứng với anchors. Vị trí của pivot là nơi mà hình chữ nhật quay
  + Width/Height: chiều rộng/ chiều cao của hình chữ nhật
  + Left, Top, Right, Bottom: vị trí của các cạnh của hình chữ nhật tương ứng với anchor của chúng
  + Anchors: điểm của anchor cho góc trái dưới và góc trên phải của hình chữ nhật
    - Min: điểm cho góc trái dưới thấp hơn của hình chữ nhật định dạng dưới một bộ phận của hình chữ nhật cha
    - Max: điểm cho góc phải trên cao hơn của hình chữ nhật định dạng dưới một bộ phận của hình chữ nhật cha
  + Pivot: vị trí của pivot
  + Rotation: góc quay của Object quanh điểm pivot của nó theo các trục X, Y, Z
  + Scale: độ lớn áp dụng nên vật theo các chiều X, Y và Z
  + Blueprint Mode: thay đổi RectTransform như chúng không bị xoay và thay đổi độ lớn
  + Raw Edit Mode: khi bật, thay đổi giá trị của pivot và anchor không thay đổi vị trí và kích cỡ của hình chữ nhật để khiến nó ở nguyên một chỗ
* Canvas: các phần tử của UI trong Canvas được thể hiện theo chúng xuất hiện trên Hierarchy. Con đầu tiền sẽ được thể hiện trước sau đó là con thứ hai và cứ như vậy. Nếu hai UI nằm trồng lên nhau thì cái sau sẽ xuất hiện ở bên trên cá trước
* Screen Space – Overlay: UI được thể hiện ở trên tất các các thứ khác được thể hiện. Nếu kích thước hoặc độ phân giải thay đổi thì Ui cũng được thay đổi để phù hợp với màn hình
* Screen Space – Camera: Thay đổi theo kích thước và độ phân giải của màn hình nhưng nó cũng thay đổi theo Camera được đặt trong Render Camera
* World Space: Cho phép đặt 2D UI vào trong 3D
* Canvans Scaler: được sử dụng để kiểm soát tổng thể kích thước và độ dày của pixel của UI. Việc thay đổi độ lớn sẽ ảnh hưởng đén mọi thứ dưới Canvas bao gồm cả font size hay viền của Image
* Constant Pixel Size: một phần của UI sẽ giữ nguyên kích cỡ pixel bất kể độ phân giải của màn hình. Ta có thể kiểm soát kích cỡ của UI bằng cách thay đổi Scale Factor
* Scale With Screen Size: được sử dụng để UI có thể thích ứng với các cách bố trí màn hình khác nhau với cùng một thiết kế
* Reference Resolution: độ phân giải của thiết kế
* Screen Match Mode: cách UI thay để thích ứng với các thiết bị có độ phân giải khác với Reference Resolution
* Match Width or Height: UI sẽ giữ lại chiều dài hay chiều rộng hay thay đổi cả hai để có thể thích ứng
* Expand: mở rộng phạm vi của Canvas theo chiều dài hay chiều rộng để cho kích thước của Canvas không bao giờ nhỏ hơn giá trị cho trước
* Shrink: thu nhỏ phạm vi của Canvas theo theo chiều dài hay chiều rộng để cho kích thước của Canvas sẽ không bao giờ lớn hơn giá trị cho trước
* Constant Physical Size: giữ các thành phần UI ở một kích thước vật lý cố định không phụ thuộc và độ phân giải của màn hình
* Graphic Raycaster: được sử dụng để thực hiện raycast nên Canvas. Raycaster nhìn tất cả UI trên cavas và xác định nếu trong chúng xem có bị chạm. Graphic Raycaster có thể cấu hình để bỏ qua các UI hướng ra sau và các 2D hay 3D Object tồn tại phía trước chúng
* Visual Component
* Text: có một vùng Text để hiện thị văn bản được nhập. Ta có thể thay đổi font, style, size hay nó có hay không khả năng rich text. Có các lựa chọn để căn chỉnh vị trí, overflow khi văn bản lớn hơn chiều dài hay chiều rộng của hình chữ nhật và Best Fit khiến cho văn bản thay đổi kích cỡ để phù hợp nhật với không gian hiện tại
* Image: một sprite có thể áp dụng cho Image component. Một material cũng có thể áp dụng cho Image component. Loại của Image sẽ thể hiện cách mà sprite sẽ xuất hiện
  + Simple: kéo tất cả sprite bằng nhau
  + Sliced: thay đổi kích thước của sprite không làm méo đi các góc chỉ bộ phân trung tâm bị kéo giãn
  + Tiled: tương tự cới Sliced nhưng thay vì kéo dãn thì nó sẽ lặp lại vị trí trung tâm. Với các sprite không có viên cả sprite sẽ lặp lại
  + Filled: thể hiện sprite tương tự như Simple nhưng nó điền vào sprite từ một gốc theo một hướng phương thức với một số lượng được xác định

Set Native Size thứ sẽ xuất hiện khi Simple hay Filled được lựa chọn sẽ đặt lại hình ảnh về kích cỡ ban đầu của chúng

* Raw Image: Image component sử dụng sprite còn RawImage sử dụng texture
* Mask: là một UI không thể thấy thay vào đó nó có thể thay đổi hình dạng của các phần tử con của nó. Vì vậy nếu một con lớn hơn cha của nó thì chỉ phần của con nằm trong cha mới có thể trông thấy
* Effects: shadow, outline
* Các UI:
* Button: có OnClick UnityEvent được khai báo nó sẽ làm gì khi được chọn
* Toggle: có IsOn checkbox để xác đinh Toggle hiện tại là bật hay tắt. Nó cũng có OnValueChanged UnityEvent để thể hiện nó sẽ làm gì khi giá trị bị thay đổi
* Toggle Group: được sử dụng cho một nhóm các Toggle .Các Toggle ở cùng một nhóm được hạn chết thì chỉ một trong số chúng được chọn và các lựa chọn khác sẽ bị tắt
* Slider: có giá trị thập phân mà người dùng có thể kéo giữa giá trị tối thiểu và giá trị tối đa. Nó có kéo theo chiều dọc hoặc chiều ngang. Nó cũng có OnValueChanged UnityEvent để xác định nó sẽ làm gì khi giá trị của nó bị thay đổi
* Scrollbar: có giá trị giữa 0 và 1. Khi người dùng kéo scrollbar, giá trị của nó sẽ thay đổi tương ứng. Nó có thể di chuyển theo chiều dọc hoặc chiều ngang. Nó cũng có OnValueChanged UnityEvent để xác định nó sẽ làm gì khi giá trị bị thay đổi
* Dropdown: là một dãy các lựa chọn từ văn bản đến hình ảnh có thể chỉ định cho từng lựa chọn có thể tạo từ Editor hay từ code. Nó cũng có OnValueChanged Unity Event dể xác định nó sẽ làm gì khi thay đổi giá trị
* Input Field: sử dụng để khiến cho Text Element có thể thay đổi bởi người dùng. Nó có UnityEvent để xác định nó sẽ làm gì khi giá trị thay đổi và một UnityEvent khác đẻ xác định nó làm gì khi người dùng kể thúc thay đổi nó
* Scroll Rect (Scroll View): được dùng khi thể hiện các nội dung chiếm nhiều diện tích nhưng cần thể hiện trong một khoảng không gian nhỏ

1. Asset

Asset là các item có thể sử dụng trong Unity để tạo ra các sản phẩm. Asset có thể là thành phần âm thanh hay hình ảnh trong project như là 3D models, texture,… Asset còn có thể là thành phần trừu tượng hơn như gradients, animation mask hay dữ liệu văn bản.