Báo cáo thực tập

Game: Space Shooter – From: Unity tutorial

Thái Quang Minh

## 1. Giới thiệu:

Game Space Shooter là 1 trò chơi trong đó bạn nhân vật chính là người điều khiển phi thuyền di chuyển trong vũ trụ vô tận để chiến đấu lại với những kẻ địch cũng là những phi thuyền chiến đấu khác và phải tránh né những chướng ngại vật thiên thạch đầy nguy hiểm. Người chơi sẽ điều khiển phi thuyền với khả năng tấn công vật lí bắn ra những viên đạn để tiêu diệt kẻ thù và thiên thạch, đồng thời người chơi có thể lựa chọn tránh né những chướng ngại này. Với mỗi chướng ngại tiêu diệt được người chơi sẽ ghi được điểm.

Các thành phần trong lập trình game Space Shooter:

- Player : Phi thuyền – người chơi sẽ điều khiển phi thuyền:
    - Player: prefab chiếc phi thuyền
    - Engines_player: prefab động cơ của phi thuyền
    - Player Controller:
        - speed(tốc độ).
        - tilt(độ nghiêng).
        - Boundary(để thiết lập phạm vi hoạt động của player không ra khỏi màn hình chơi)
        - shot – viên đạn do player bắn ra – đây là prefab do chúng ta tạo ra.
        - shotPawnn – vị trí viên đạn bắn ra,
        - TouchPad-để điều khiển player trên điện thoại, areaButton – vị trí khi người dùng nhấn vào thì phi thuyền sẽ bắn đạn
        - fireRate – tốc độ bắn đạn
- Chướng ngại vật:
    - Asteroid – được tạo thành prefab với các script mô phỏng chuyển động của asteroid : tên script: mover – asteroid sẽ chuyển động về phía người chơi từ trên xuống
    - Enemy ship - được tạo thành prefab với các script mô phỏng chuyển động của Enemy ship bao gồm: chuyển động từ trên xuống với khả năng thay đổi hướng đi và khả năng bắn ra đạn
- Game Controller:
    - bộ điều khiển game với nhiệm vụ sinh ra các chướng ngại vật là asteroid và enemyship ở các vị trí random khác nhau và số lượng các chướng ngại vật này.

    - StartWait : thời gian đợi để bắt đầu trò chơi.
    - WaveWait: thời gian đợi màn chơi kế tiếp.
    - SpawnWait: thời gian để sinh ra các chướng ngại vật.
    - ScoreText: thể hiện điểm.
    - GameOverText hiện lên khi người chơi thua cuộc.
    - RestartText – hiện thông báo để restart lại game khi người chơi thua cuộc.
- Lighting:
    - Sử dụng directional Light, không phụ thuộc vào vị trí chiếu sáng, chỉ phụ thuộc theo góc chiếu:
    - Main light, Fill light, Rim light: có góc chiếu và cường độ chiếu khác nhau
- Camera
- Background: thể hiện anh nền của trò chơi
- Canvas - hiển thị Score, gameOver, Restart

## 2.Các khái niệm cơ bản trong Unity:

**1. GameObject**

Một đối tượng cụ thể trong game gọi là một game object, có thể là nhân vật, đồ vật nào đó

Ví dụ: cây cối, xe cộ, nhà cửa, người...

**2. Component**

Một GameObject sẽ có nhiều thành phần cấu tạo nên nó như là hình ảnh (sprite render), tập hợp các hành động (animator), thành phần xử lý va chạm (collision), tính toán vật lý (physical), mã điều khiển (script), các thành phần khác... mỗi thứ như vậy gọi là một component của GameObject.

**3. Sprite**

Là một hình ảnh 2D của một game object có thể là hình ảnh đầy đủ, hoặc có thể là một bộ phận nào đó.

**4. Animation**

Là tập một hình ảnh động dựa trên sự thay đổi liên tục của nhiều sprite khác nhau.

**5. Frame**

Frame là một trạng thái của một animation. Có thể được tạo nên từ 1 sprite hay nhiều sprite khác nhau.

**6. Prefabs**

Là một khái niệm trong Unity, dùng để sử dụng lại các đối tượng giống nhau có trong game mà chỉ cần khởi tạo lại các giá trị vị trí, tỉ lệ biến dạng và góc quay từ môt đối tượng ban đầu.

Ví dụ: Các đối tượng là đồng tiên trong game Mario đều có xử lý giống nhau, nên ta chỉ việc tạo ra một đối tượng ban đầu, các đồng tiền còn lại sẽ sử dụng prefabs.

Hoặc khi ta lát gạch cho một cái nền nhà, các viên gạch cũng được sử dụng là prefabs.

**7. Sounds**

Âm thanh trong game.

**8. Script**

Script là tập tin chứa các đoạn mã nguồn, dùng để khởi tạo và xử lý các đối tượng trong game.

Trong Unity có thể dùng C#, Java Script, BOO để lập trình Script.

**9. Scenes**

Quản lý tất cả các đối tượng trong một màn chơi của game.

## 3. Các khái niệm cần thiết trong Unity:

Như trong game nói trên chúng ta có một player. Vậy thì để cách để tạo ra player này là như thế nào?

Tạo một game Object sẽ chứa prefag :

- player ship – có sẵn trong bộ tutorial này
- engine cho player ship – có sẵn trong bộ tutorial này
- Tạo component rigid body để tạo tương tác vật lý
- Tạo collider cho object
- Tạo script để điều khiển player

Để có thể làm được điều này chúng ta sẽ cần tìm hiểu về một số lí thuyết cần thiết. Trước đo chúng ta cần phân biệt một số hàm đặc trưng trong script được kế thừa từ lớp MonoBehavior:

**Awake()-chúng ta không gọi hàm này trong script nhưng sẽ phân biệt nó với Start()**: Awake là phương thức được tự động gọi khi Script component được nạp. Awake chỉ được gọi duy nhất 1 lần. Phương thức này được dùng để khởi tạo bất cứ biến nào hoặc trạng thái game trước khi game bắt đầu. Phương thức Awake được gọi sau khi tất cả các đối tượng được khởi tạo vì thế bạn có thể giao tiếp với các đối tượng khác hoặc truy vấn chúng. Mỗi hàm Awake của đối tượng trong game được gọi với thứ tự ngẫu nhiên giữa các đối tượng. Bởi vì điều này, bạn có thể sử dụng Awake để thiệt lập tham số giữa các Script component và dùng Start để chuyển bất kì tham số nào qua lại giữa chúng. Awake luôn được gọi trước Start, điều này cho phép bạn có sắp xếp thứ tự khởi tạo của Script. Awake không thể là 1 co-routine.

**Start()** : Start được gọi trước lần gọi Update đầu tiên ngay tại frame Script component được bật. Cũng giống như Awake, Start được gọi duy nhất 1 lần. Tuy nhiên, Awake được gọi khi đối tượng được khởi tạo, bất kể Script component có

được bật hay tắt. Start có thể không được gọi trên cùng 1 frame với Awake nếu Script component không được bật ở thời điểm khởi tạo.

**Update()**: Hàm Update được gọi theo từng frame trong Unity (thời gian xử lý giữa các frame không cố định), và được dùng để xử lý những đối tượng non-physic.

**FixedUpdate()** : FixedUpdate được gọi theo từng khoảng thời gian cố định (mặc định là 0.02s) và được dùng để xử lý những đối tượng có physic (đối tượng được gắn rigidbody chẳng hạn)

Chính xác những gì chúng ta cần làm trong script này là gì?

Để duy chuyển được đối tượng sau mỗi frame chúng ta sẽ lập trình di chuyển trong hàm FixedUpdate():

float moveHoruzontal = Input.GetAxis(“Horizontal”);

float moveHoruzontal = Input.GetAxis(“Vertical”);

Chúng ta sẽ tìm hiểu về hàm Input.GetAxis trong Unity:

> Điểm khác biệt giữa GetAxis so với các phương thức GetKey hay GetButton là nó không trả về giá trị true/false như thông thường mà sẽ trả về một giá trị float có phạm vi nằm trong khoảng [-1;1] phụ thuộc vào thời gian nhấn giữ của một phím.
>
> Input.GetAxis sẽ trả về giá trị **từ** -1 đến 1. Nếu không bấm gì sẽ trả về 0. Vì thế mà khi sử dụng GetAxis để làm chuyển động cho nhân vật sẽ mượt mà hơn. Ví dụ khi ta dùng:
>
> float move = Input.GetAxis("Horizontal");
>
> thì khi ta chạy game và bấm phím A move không phải có giá trị -1 liền mà nó kéo từ 0 xuống từ từ đến 1. Tương tự bấm D thì giá trị của move sẽ tăng từ từ lên đến 1. Và như thế thì nhân vật sẽ chuyển động mượt mà hơn nhiều.

GetComponent<Type>() sẽ trả về component đầu tiên cần tìm của đối tượng, nếu không tìm được sẽ trả về null hoặc mảng không có phần tử tương ứng. Cú pháp này sử dụng template để lấy ra các component. Đây là cách thông dụng nhất được lập trình viên sử dụng trong **C# Script**.

GetComponent<Rigidbody>() sẽ trả về rigidbody component của object đó với đầy đủ các thuộc tính, rigidbody là một thực thể vật lý có đầy đủ các tính chất vật

lý cơ học. Nó sẽ điều khiển mọi vấn đề vật lí của đối tượng đó. *Chú ý: Một đối tượng chỉ có một Rigidbody Component:*

| Property | Function |
|---|---|
| Mass | Khối lượng của đối tượng |
| Linear Drag | Hệ số ma sát trượt (lực ma sát trượt) |
| Angular Drag | Hệ số ma sát lăn (lực ma sát lăn) |
| Gravity Scale | Mức độ ảnh hưởng của trọng lực |
| Fixed Angle | Nếu check thì đối tượng này không quay dưới tác dụng lực |
| Is Kinematic | Nếu check thì đối tượng này không chịu ảnh hưởng của trọng lực cũng như các lực tác động lên nó |
| Interpolate | Nội suy chuyển động từ cập nhật các va chạm vật lí |
| None | Không áp dụng |
| Interpolate | Làm smooth chuyển động dựa trên cập nhật vị trí của đối tượng ở frame trước đó |
| Extrapolate | Làm smooth chuyển động dựa trên việc tính toán trước vị trí đối tượng trong frame tiếp theo |
| Sleeping Node | Cách Object ở chế độ sleep khi nó nghỉ ngơi (đứng yên) để tiết kiệm chi phí xử lí |
| Never Sleep | Không bật sleep |
| Start Awake | Đối tượng được đánh thức vào lúc khởi tạo |
| Start Asleep | Đối tượng khởi tạo ở trạng thái sleep nhưng có thể kích hoạt bằng 1 va chạm |
| Collision Detection | Cách thức bắt va chạm với các đối tượng |
| Discrete | Bắt va chạm khi collider của đối tượng có 1 va chạm với các đối tượng khác trong quá trình update physics |
| Continous | Giông như discrete, nhưng nó chỉ áp dụng trong trường hợp va chạm các đối tượng chuyển động nhanh (frame sau object có thể đi xuyên qua vật thể cần va chạm dẫn đến sai trong vật lý). Continous cần một số lượt tính toán nhiều hơn nên chỉ dùng khi cần thiết |

Các thuộc tính trong RigidBody:

| | |
|---|---|
| angularDrag | Góc kéo của đối tượng có thể hiểu là sức cản không khí khi vật quay, được sử dụng để làm giảm tốc độ quay của vật thể. Angular drag càng lớn thì tốc độ quay càng chậm |
| angularVelocity | Đây là một vector vận tóc gốc có đơn vị là radian per second |
| centerOfMass | Điểm trọng tâm mà trọng lực tác dụng đến |

| | |
|---|---|
| collisionDetectionMode | Chế độ nhận biết va chạm của rigidbody |
| contraints | Điều khiển DOF (degrees of freedom) nào được sử dụng để mô phỏng trong rigidbody |
| detectCollision | Enabled hoặc disabled nhận biết va chạm |
| drag | Lực cản của một object |
| freezeRotation | Cho phép tác động vật lý có thể thay đổi góc quay của một vật không |
| inertiaTensor | ? |
| inertiaTensorRotation | ? |
| interpolation | ? |
| isKinematic | Cho phép vật lý tác động lên rigidbody không? |
| mass | Khối lượng của object |
| maxAngularVelocity | Giá trị lớn nhất của angularVelocity. Mặc định là : 7 |
| maxDepenetrationVelocity | ? |
| position | Vị trí của rigidbody |
| rotation | Góc quay của rigidbody |
| sleepThreshold | ? |
| solverVelocityIteration | ? |
| useGravity | Cho phép trọng lực tác dụng lên rigidbody hay không |
| velocity | Vector  vận tóc của rigidbody |
| worldCenterOfMass | Trọng tâm của rigidbody in world Space |

Các phương thức:

| | |
|---|---|
| AddExplosionForce | Áp dụng một lực tương tự lực nổ lên rigidbody |
| AddForce | Áp dụng một lực lên rigidbody |
| AddForceAtPosition | Áp dục lực lên một vị trí |
| AddRelativeForce | Lực chỉ có tác dụng đến active object. Nếu object là inactive thì lực không có tác dụng |
| AddRelativeTorque | Tương tự như addReltiveForce nhưng đây là mô men |
| AddTorque | Thêm mô-men cho rigidbody |
| ClosestPointOnBounds | Điểm gần nhất đến collider bao quanh object |
| GetPointVelocity | Điểm của vector Vận tốc của rigidbody trong world space |
| GetRelativePointVelocity | ? |
| IsSleeping | Trả về rigibody ở chế độ sleeping hay không? |
| MovePosition | Di chuyển rigidbody |
| MoveRotation | Quay rigidbody |

| | |
|---|---|
| ResetCenterOfMass | Reset lại trọng tâm |
| ResetInertiaTensor | Reset lại mô men quán tính |
| SetDensity | ? |
| Sleep | Thiết lập chế độ ngủ cho rigidbody |
| SweepTest | ? |
| SweepTestAll | ? |
| WakeUp | Gọi rigidbody thức dậy |

Message:

| | |
|---|---|
| OnCollisionEnter | Hàm OnCollisionEnter được gọi khi collider/rigidbody này bắt đầu chạm collider/rigidbody khác |
| OnCollisionExit | Hàm OnCollisionExit được gọi khi collider/rigidbody này kết thúc việc chạm collider/rigidbody khác |
| OnCollisionStay | Hàm OnCollisionStay được gọi ở mỗi frame khi collider/rigidbody này còn chạm collider/rigidbody khác |

Tìm hiểu về Collider: Va chạm và xử lí va chạm là những thành phần không thể thiếu khi lập trình game. Va chạm trong game là xảy ra khi chúng ta có 2 object đi vào không gian của nhau. Ví dụ như : trúng đạn, trúng bom, chạm phải quái vật, xuất phát, tới đích, trúng mũi tên ….. đều là các sự kiện va chạm và khi lập trình game chúng ta cần phải xử lí các va chạm đó. Chúng ta sẽ có 2 quá trình xử lí với va chạm như sau:

Quá trình 1 : phát hiện và thông báo sự kiện va chạm.

Qúa trình 2 : xử lí sự kiện va chạm.

Trong Unity có 2 loại va chạm đó là : Collision : là loại va chạm mà 2 đối tượng sẽ không đi xuyên qua nhau, khi đối tượng này gặp đối tượng kia thì sẽ bị cản lại, bật lại tùy theo tính chất vật lý mà chúng ta xét cho đối tượng. Ví dụ : Qủa bóng rơi từ trên cao rơi xuống sân cỏ sẽ bật lên … Trigger : là loại va chạm mà các đối tượng này có thể đi xuyên qua đối tượng kia, chúng ta sẽ sử dụng trigger trong các hoạt cảnh như làm cho tiếng nhạc bật lên khi đối tượng đi qua loa hay làm cho cây đổ khi người chơi đi tới, hay lửa …

Tạo một Collider cho đối tượng. Colliders là vật mà engine vật lý sẽ dung để có thể nhận ra sự va chạm, không giống như các mesh, chúng nhận biết được mỗi khi va chạm với nhau. Đa số collider có hình dạng đơn giản nhằm mục đích tính toán đơn

giản và dễ dàng hơn, phần lớn các object trong Unity sẽ được gắn collider mỗi khi tạo ra, với Cube là Box Collider, Sphere là Sphere Collider, Cylinder là Capsule Collider.. Các loại Collider:

Class Collider (nhưng collider khác như box collider, sphere collider,… đều được kế thừa từ class Collider này):

Các thuộc tính:

| | |
|---|---|
| AttachedRigidbody | Rigidbody mà collider này được gắn vào |
| bounds | Khoảng không gian của collider |
| contactOffset | Khoảng cách được xem là đã va chạm với collider |
| Enabled | Đã kích hoạc collider hay không |
| IsTrigger | Trả về collider đã ở chế độ trigger không? |
| Material | Trả về material sử dụng bởi collider |
| SharedMaterial | ? |
| GameObject | Gameobject mà collider này được gắn vào |
| Tag | Tag của game object |
| Transform | Transform của game object |
| HideFlags | ? |
| Name | Tên của object |

Các phương thức:

| | |
|---|---|
| ClosestPoint | Trả về điểm gần nhất với vùng được cho |

| | |
|---|---|
| ClosestPointOnBounds | Trả về điểm gần nhất với bounding box |
| Raycast | ? |

Message:

| | |
|---|---|
| OnCollisionEnter | Tượng tự ở trên |
| OnCollisionExit | Tượng tự ở trên |
| OnCollisionStay | Tượng tự ở trên |
| OnTriggerEnter | Hàm OnTriggerEnter được gọi khi collider vào trong một trigger |
| OnTriggerExit | Hàm OnTriggerExit được gọi khi collider ra khỏi một trigger |
| OnTriggerStay | Hàm OnTriggerStay được gọi ở mỗi frame khi collider còn ở trong một trigger |

Quaternion trong Unity: Quaternion được xây dựng dựa trên số phức toán học và được biểu diễn bởi 4 thành phần x, y, z, w với công thức:

$$Q = w + xi + yj + zk$$

Với i, j, k là các thành phần ảo, được xem như ba vector đơn vị của các trục toạ độ tương ứng. Để hiểu sâu sắc về Quaternion, bạn cần có một lượng kiến thức toán học về số phức, lượng giác, …

Trong Unity, mọi phép quay của đối tượng đều được lưu trữ và thể hiện bằng Quaternion với 4 thành phần kể trên. Các thành phần này không nên được chỉnh sửa độc lập đối với lập trình viên thông thường.

Để truy xuất các thành phần đó, bạn có thể sử dụng qua các kênh x, y, z, w tương ứng.

Ngoài ra còn những cách khác để thiết lập giá trị cho các kênh thông tin của Quaternion như sử dụng **constructor** hay hàm **Set**. Các cách này chỉ có thể chỉnh sửa cùng lúc cả 4 kênh của Quaternion.

Những hàm Quaternion mà bạn sẽ dùng khoảng 99% trong lúc lập trình: Quaternion.LookRotation, Quaternion.Angle, Quaternion.Euler, Quaternion.Slerp, Quaternion.FromToRotation, Quaternion.identity

Static Properties:

| | |
|---|---|
| Identity | Quaternioin đơn vị. (không thực hiên quay) |

Properties:

| | |
|---|---|
| eulerAngles | Trả về euler angle biểu diễn của phép quay -Thuộc tính này trả về một Vector3, tương ứng với góc quay của từng trục trên đối tượng. Đây là cách thức biểu diễn thông thường và dễ hiểu hơn của phép quay. Ví dụ (90,0,0) nghĩa là quay theo trục x 90 độ |
| This[int] | |
| w | Giá trị w của Quaternion |
| x | Giá trị x của Quaternion |
| y | Giá trị y của Quaternion |
| z | Giá trị z của Quaternion |

Public Methods:

| | |
|---|---|
| Set | Thiết lập giá trị x,y,z,w cho quaternion |
| SetFromRotation | Tạo một phép quay mà quay từ fromDirection đên ToDirection |
| SetLookRotation | Tạo một phép quay được xác định theo forward and upwards directions |
| ToAngleAxis | Chuyển đổi phép quay thành dạng biểu diễn góc quay trên mỗi trục |
| ToString | Định dạnh theo string |

Static Methods:

| | |
|---|---|
| Angle | Góc giữa 2 phép quay |
| AngleAxis | Tạo một phép quay theo góc quay quanh trục |

| | |
|---|---|
| Dot | Tích vô hướng giữa 2 phép quay |
| Euler | hàm giúp ta thao tác và điều chỉnh góc quay của đối tượng thông qua chính ba trục toạ độ của đối tượng đó |
| FromToRotation | Tạo một phép quay mà quay từ fromDirection đên ToDirection |
| Inverse | Trả về phép quay đảo ngược |
| Lerp | Lerp và Slerp là hai hàm nội suy phép quay giữa hai Quaternion và hiển thị việc quay ra màn hình theo thời gian. Cách sử dụng và chức năng cả hai hàm là tương đối giống nhau. Với các phép quay lớn, góc quay rộng, hàm Lerp sẽ cho chất lượng hiển thị kém hơn Slerp. Nhưng bù lại, về mặt tốc độ thì Lerp nhanh hơn do không thực hiện nội suy một cách phức tạp. Bạn đọc theo dõi cách sử dụng hai hàm này trong demo đính kèm.<br><br>Gía trị tham số t được kẹp giữa từ 0 đến 1 |
| LerpUnclamped | Tương tự như lerp nhưng giá trị tham số t không được kẹp giữa từ 0 đến 1 |
| LookRotation | Tạo một phép quay được xác định theo forward and upwards directions |
| RotateTowards | Quay một góc từ toward đến. |
| Slerp | Tương tự |
| SlerpUnclamped | Tương tự |