

# CÔNG NGHỆ MỚI TRONG CNTT

## PHÁT TRIỂN ỨNG DỤNG

Các tập lệnh trong Unity

# khung

---

- Khung là một thuật ngữ đư ợc kế thừa từ hoạt hình.
- 24, 30 và 60 khung hình mỗi giây (FPS).
  - Ví dụ, nó đang chạy ở tốc độ 60 FPS, điều đó có nghĩa là có 60 hình ảnh mới trong thứ hai đư ợc trình bày cho ngư ời chơi, gọi là tốc độ khung hình.
- khoảng thời gian khung hình là thời gian xảy ra giữa mỗi khung hình
  - Ví dụ về 60 FPS, 60 sẽ là tốc độ khung hình và khoảng thời gian giữa các khung hình sẽ là  $\frac{1}{60} \times 1000 = 16,67$  mili giây.

# khung trong Unity

---

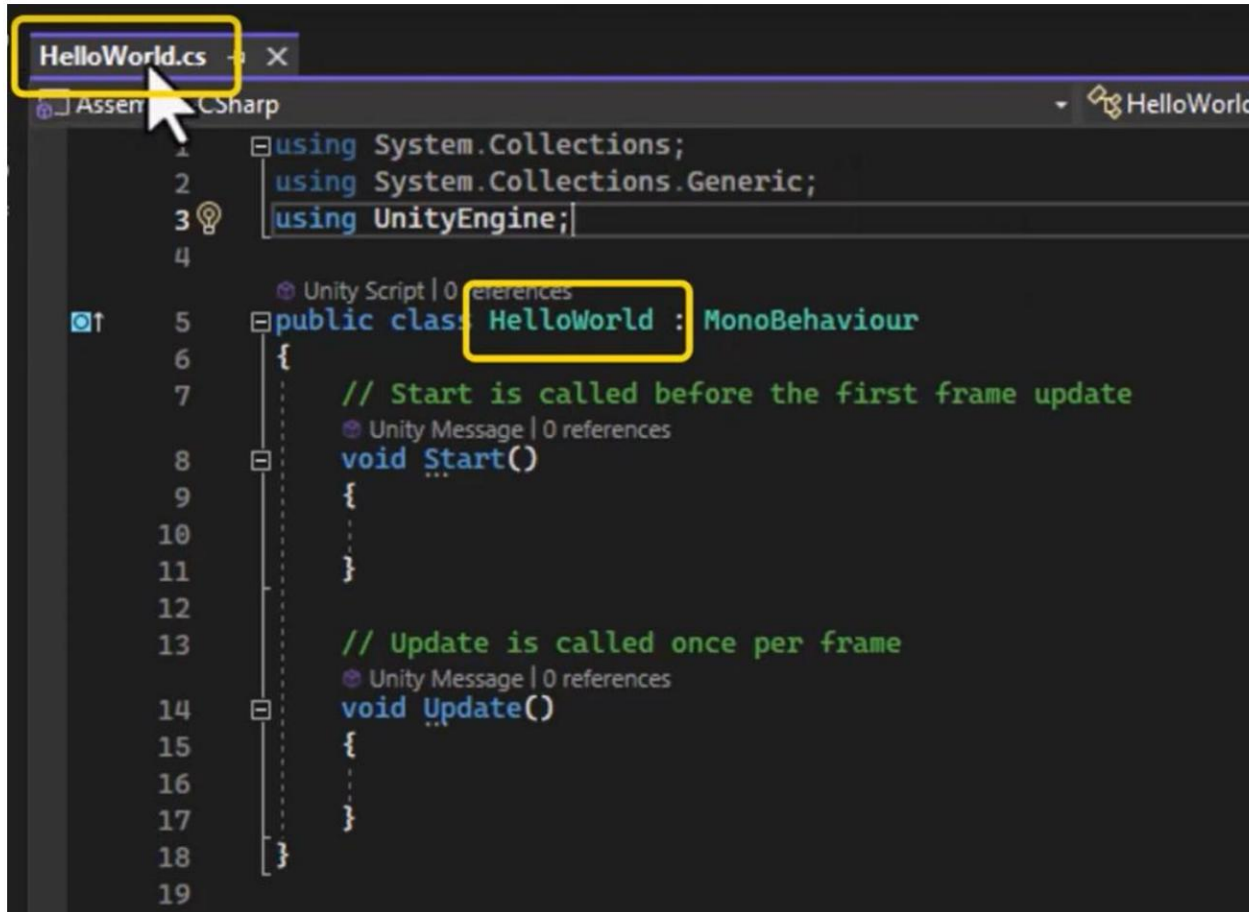
- Khung đư ợc coi là hình ảnh đư ợc hiển thị cho ngư ời chơi màn hình
- Quan tâm đến thuộc tính `Time.DeltaTime` của khung tính toán thời gian giữa các khung
- Nếu xử lý nhiều tác vụ trong khi chơi game, thời gian giữa các khung hình sẽ chậm lại, điều này làm giảm tốc độ khung hình và khiến ngư ời chơi có cảm giác trò chơi đang chạy chậm.

## Các tập lệnh trong Unity

---

- Các tập lệnh như các thành phần hành vi
- Giống như các thành phần khác của Unity, chúng có thể được áp dụng cho các đối tượng.
- Sử dụng “Thêm thành phần” của Đối tượng để đính kèm tập lệnh

# Tập lệnh C#



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class HelloWorld : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
19
```

# Xin chào thế giới

---

sử dụng System.Collections;

sử dụng System.Collections.Generic;

sử dụng UnityEngine;

lớp công khai HelloWorld : MonoBehaviour

```
{  
    // Start được gọi trước khi cập nhật khung đầu tiên  
    trỏng Bắt đầu()  
    {  
  
    }  
  
    // Cập nhật được gọi một lần cho mỗi khung  
    void Cập nhật()  
    {  
  
    }  
}
```

Chạy một lần khi điều này

lớp học đã được khai giảng

Chạy một lần mỗi khung khi  
dự án chạy, nghĩa là 30 lần  
một giây

# Xin chào thế giới

---

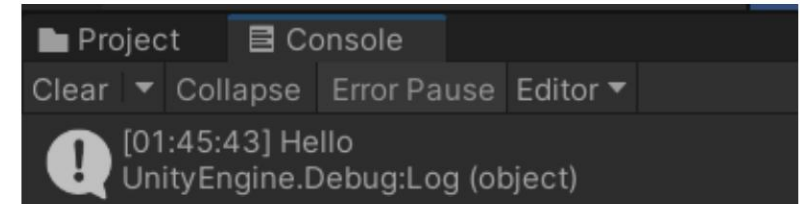
```
sử dụng System.Collections; sử  
dụng System.Collections.Generic; sử dụng  
UnityEngine;
```

```
lớp công khai HelloWorld : MonoBehaviour {
```

```
    // Start được gọi trước khi cập nhật khung đầu tiên void  
    Start() {
```

```
        Debug.Log("Xin chào");  
    }
```

```
    // Cập nhật được gọi một lần cho mỗi khung  
    void Update() {
```



# Xin chào thế giới

```
sử dụng System.Collections; sử  
dụng System.Collections.Generic; sử dụng  
UnityEngine;
```

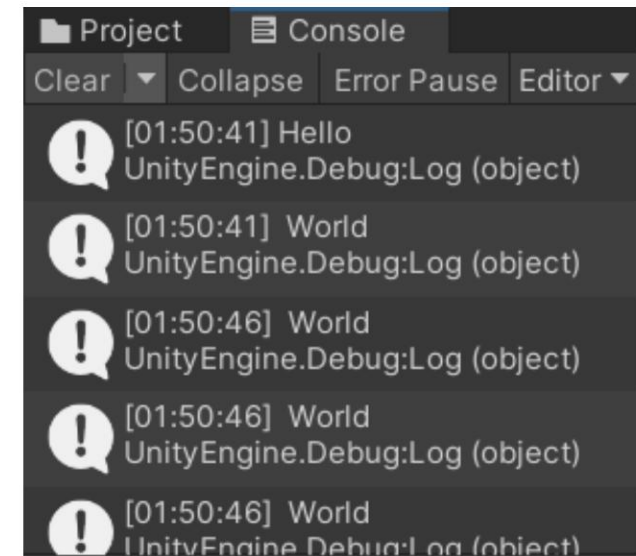
```
lớp công khai HelloWorld : MonoBehaviour {
```

```
    // Start được gọi trước khi cập nhật khung đầu tiên void  
    Start() {
```

```
        Debug.Log("Xin chào");  
    }
```

```
    // Cập nhật được gọi một lần cho mỗi khung  
    void Update() {
```

```
        Debug.Log(" Thế giới");
```





# Xin chào Chương trình

---

```
sử dụng System.Collections; sử  
dụng System.Collections.Generic; sử dụng  
UnityEngine;  
  
lớp công khai HelloWorld : MonoBehaviour {  
  
    // Start được gọi trước khi cập nhật khung đầu tiên void  
    Start() {  
  
    }  
  
    // Cập nhật được gọi một lần cho mỗi khung  
    void Update() {
```

# Kiểu dữ liệu

---

## Kiểu giá trị

- Int
- Trôi nổi
- Gấp đôi
- Lơ
- Nhân vật
- Cấu trúc
  - Vectơ
  - Quaternion

## Loại tham chiếu •

### Các lớp

- Biến đổi
- Đối tượng trò chơi

# Kiểu dữ liệu

```
sử dụng UnityEngine; sử  
dụng System.Collections;  
  
lớp công khai DatatypeScript : MonoBehaviour {  
  
    void Bắt đầu ()  
    {  
        //Biến kiểu giá trị Vector3  
        pos = transform.position; pos = new  
        Vector3(0, 2, 0);  
  
        //Biến kiểu tham chiếu Transform  
        tran = transform; tran.position =  
        new Vector3(0, 2, 0);  
    }  
}
```

# Biến số

---

- Biến là một vị trí được đặt tên trong bộ nhớ nơi lập trình viên có thể lưu trữ dữ liệu và sau đó truy xuất dữ liệu bằng cách sử dụng biến “tên”
- Người lập trình được chọn tên của các biến
- Bạn có thể thay đổi nội dung của một biến trong một câu lệnh sau

x     12,2

y     14

x12.2

và 14

# Chức năng

---

- một hàm là một số mã có thể tái sử dụng lấy **đối số** làm đầu vào, thực hiện một số phép tính và sau đó trả về một hoặc nhiều kết quả

# Biến và hàm

sử dụng `System.Collections;`

sử dụng `System.Collections.Generic;`

sử dụng `UnityEngine;`

lớp công khai `EmptyBV : MonoBehaviour`

`int`: kiểu số nguyên

số nguyên `x = 5;`

biến `x`

khoảng trống `Bắt đầu() {`

`x = 70;`

`Gỡ lỗi.Log(x);`

`}`

`void`: hàm không trả về giá trị

`void Cập nhật() {`

`}`

`}`

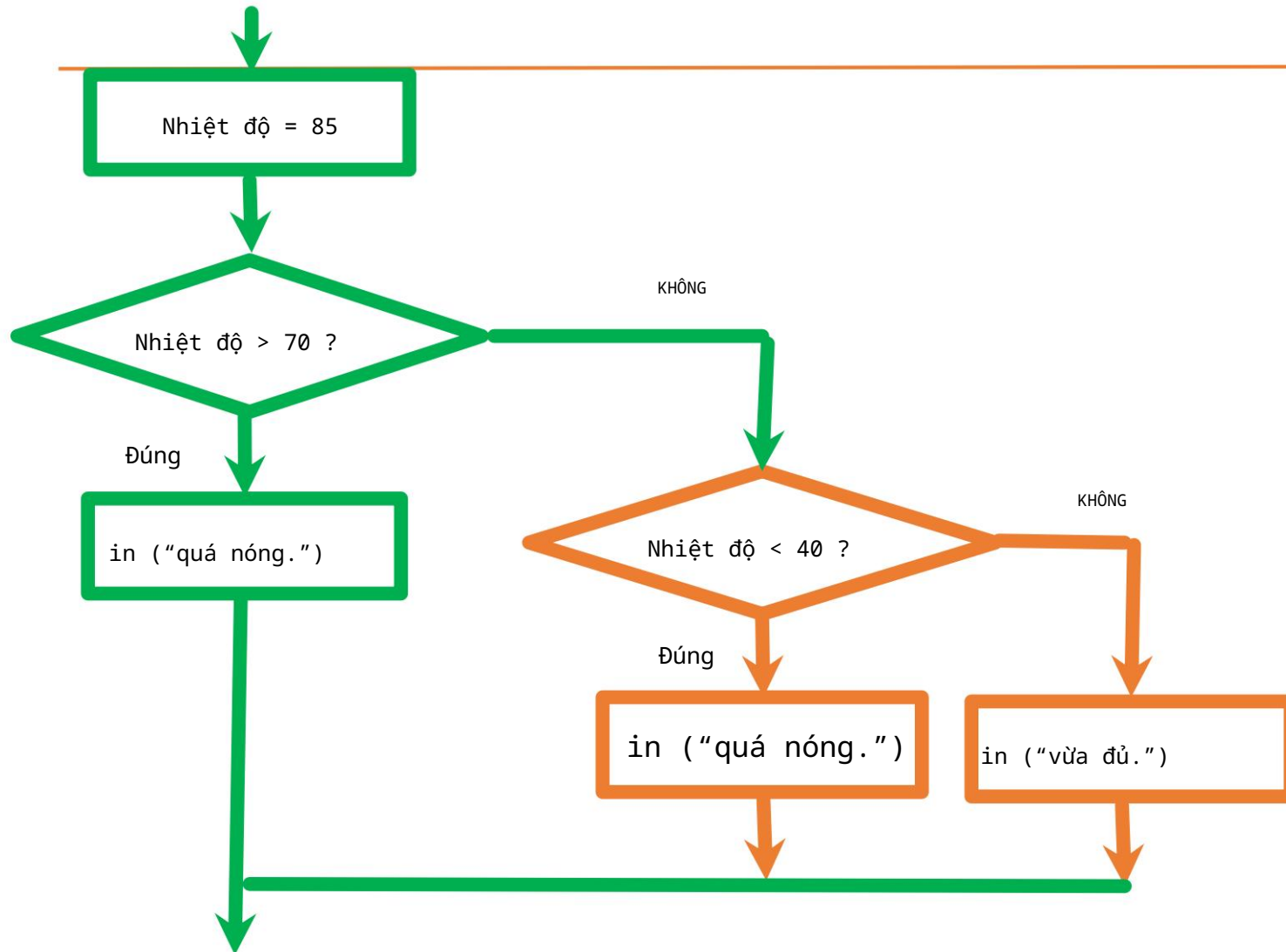
# Biến và hàm

```
sử dụng System.Collections;  
sử dụng System.Collections.Generic;  
sử dụng UnityEngine;  
  
lớp công khai EmptyBV : MonoBehaviour {  
  
    int x = 5;  
  
    int MultiplyByTwo(int num) { int res =  
        x*2; trả về res;  
  
    } void Start() {  
        Start() x = MultiplyByTwo  
        (x); Debug.Log(x);  
    }  
}
```

Giá trị trả về của hàm  
MultiplyByTwo , kiểu của nó là int

Gọi MultiplyByTwo

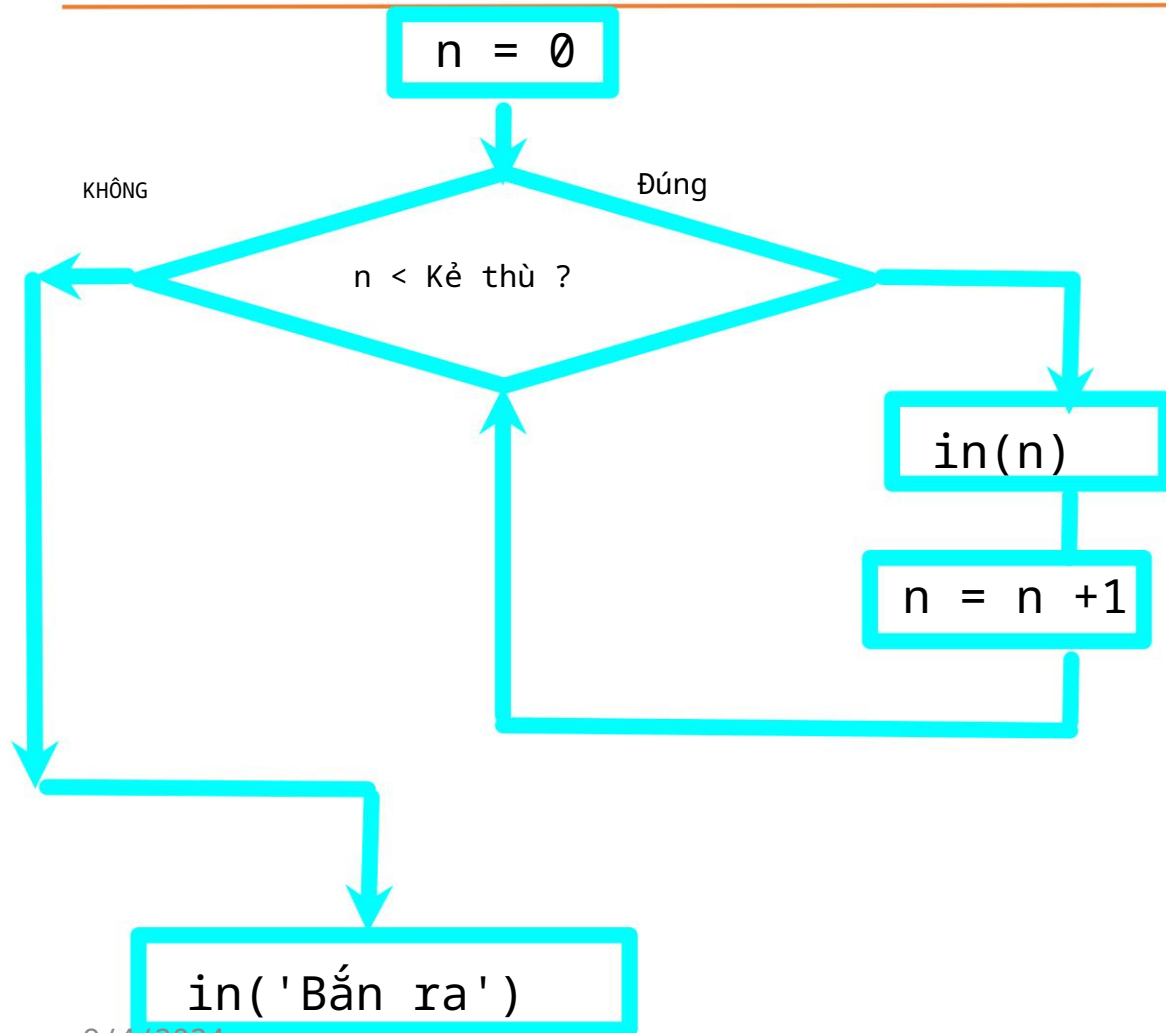
# Định nghĩa nếu



```
Nhiệt độ = 85;  
nếu(Nhiệt độ > 70)  
{  
    print("quá nóng.");  
}  
nếu không thì nếu (Temp <  
Temp) {  
    print("quá lạnh.");  
}  
khác {  
    print("vừa đúng.");  
}
```

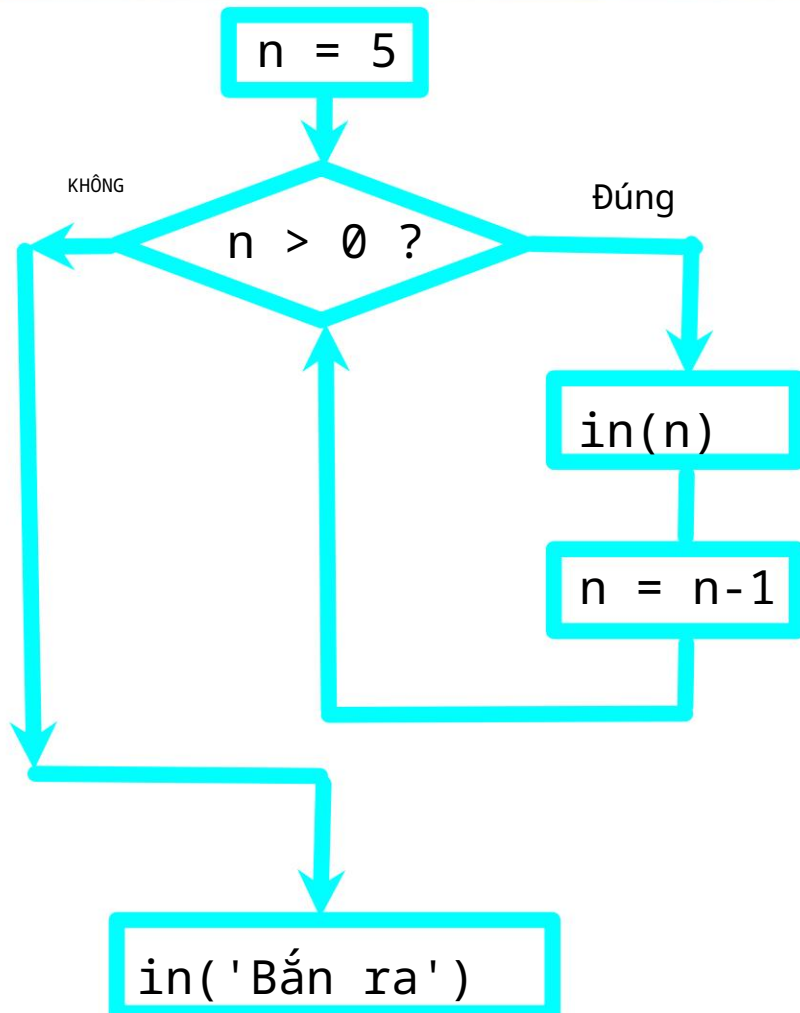


# Định nghĩa cho



```
void Bắt đầu ()  
{  
    for(int n = 0; n < Kẽ thù; n++) {  
        Debug.Log("Đang tạo số kẻ thù: " + n);  
    }  
    Debug.Log("BẮn ra");  
}
```

# Định nghĩa trong khi



```
Chương trình: while(n > 0) {  
    in (n); n = n - 1;  
}  
print("Bắn đi!");  
print(n);
```

Đầu ra:

5

4

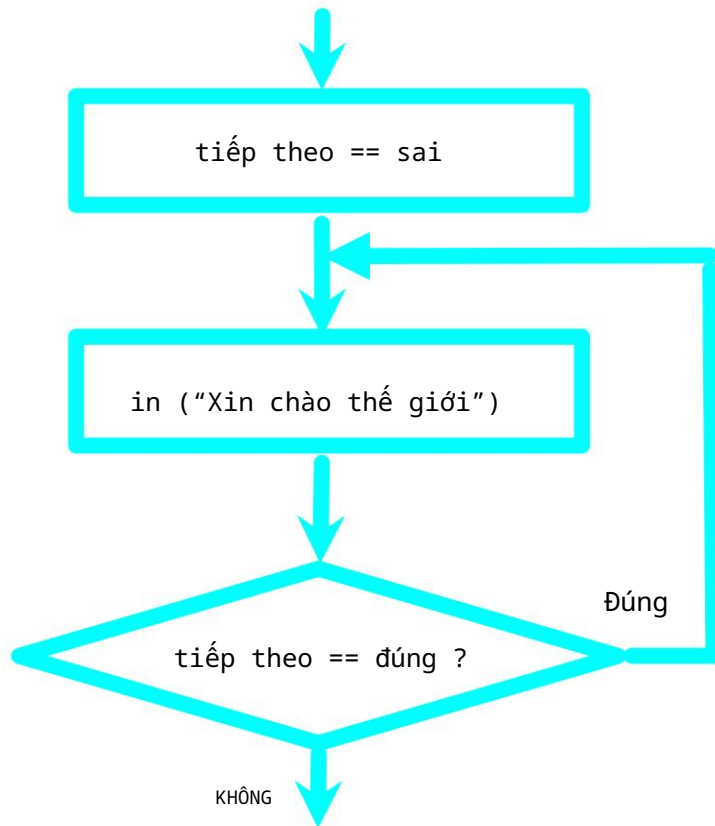
3

2

1 Cất cánh!

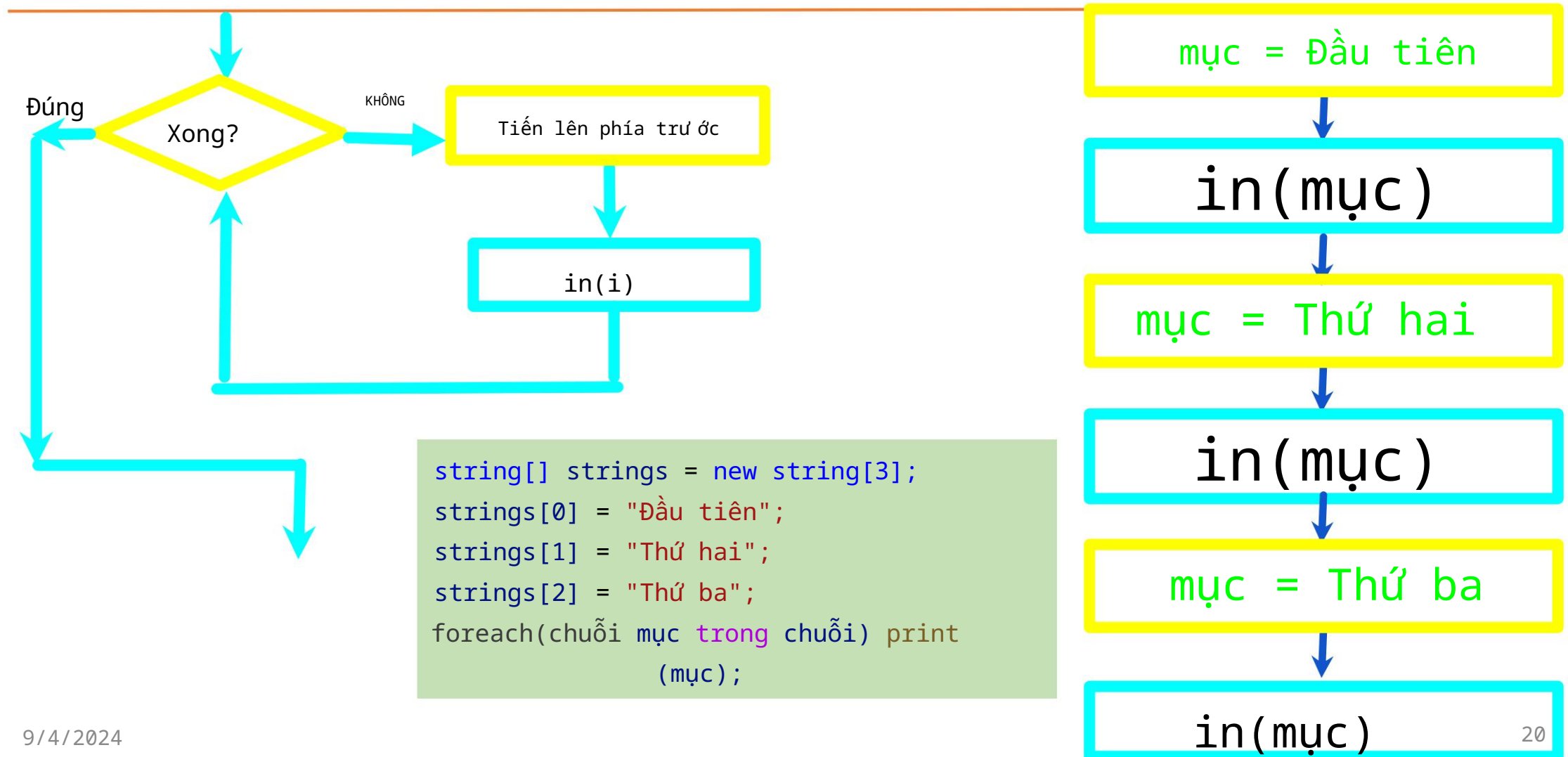
0

# Định nghĩa DoWhile



```
bool tiếp theo = sai;  
LÀM  
{  
    in ("Xin chào thế giới"); }  
while(next == true);
```

# Định nghĩa foreach



# Phạm vi và Bộ điều chỉnh quyền truy cập

lớp công khai ScopeModifier : MonoBehaviour

{

    công khai int alpha = 5;

    int beta riêng tư = 0;

    int gamma riêng tư = 5;

alpha, beta,  
gamma đối phó  
trong lớp

void Ví dụ(int bút, int bút chì màu)

{

    câu trả lời int ;

    trả lời = bút\*bút chì màu\*chữ alpha;

    Debug.Log(câu trả lời);

}

bút, bút chì màu,  
trả lời đối phó  
trong chức năng

// Cập nhật được gọi một lần cho mỗi khung

void Cập nhật()

{

    Debug.Log("Alpha được đặt thành: } " + anpha);

}

Bộ điều chỉnh quyền truy cập

Alpha có thể  
được nhìn thấy từ  
ngoài lớp học

## Phạm vi và Bộ điều chỉnh quyền truy cập

---

- Alpha sẽ không bị ghi đè khi chương trình bắt đầu

```
trống Bắt đầu()  
{  
    alpha = 79;  
}
```

```

lớp công khai AnotherClass : MonoBehaviour {

    công khai int táo;
    công khai int chuối;

    máy bấm kim riêng ; bằng
    dính riêng ;

    công khai void FruitMachine (int a, int b) {

        int answer;
        answer = a + b;
        Debug.Log(" Tổng số trái cây: " + answer);
    }

    riêng tư void OfficeSort (int a, int b) {

        int answer;
        answer = a + b;
        Debug.Log(" Tổng vật dụng văn phòng: " + answer);
    }
}

```

```

khoảng trống Bắt đầu()
{
    alpha = 79;
    myOtherClass = lớp
    AnotherClass
    mới (); myOtherClass.
}

```

# Thức dậy và bắt đầu

---

- Sử dụng Awake để khởi tạo các biến hoặc trạng thái trước khi ứng dụng bắt đầu
- Unity chỉ gọi Awake một lần trong suốt thời gian tồn tại của phiên bản tập lệnh.
  - Thời gian tồn tại của một tập lệnh kéo dài cho đến khi Cảnh chứa tập lệnh đó được dỡ tải
- Nếu Scene được tải lại, Unity sẽ tải lại phiên bản tập lệnh và gọi lại Awake. Awake được gọi một lần cho mỗi phiên bản.
- Awake luôn được gọi trước bất kỳ hàm Start nào
- Sử dụng Awake để thiết lập tham chiếu giữa các tập lệnh và sử dụng Start, được gọi sau khi tất cả các cuộc gọi Awake hoàn tất, để truyền bất kỳ thông tin nào qua lại



# Cập nhật và Cập nhật cố định

---

## Cập nhật

- Gọi mọi khung hình
- Được sử dụng để cập nhật thường xuyên như :
  - Di chuyển các vật thể phi vật lý
  - Bộ đếm thời gian đơn giản
  - Nhận đầu vào
- Thời gian cập nhật khoảng thời gian thay đổi

## Đã sửa Cập Nhật

- Gọi là mọi bước vật lý
- Khoảng thời gian FixedUpdate là nhất quán
- Được sử dụng cho các bản cập nhật thường xuyên như BẢNG:
  - Điều chỉnh các đối tượng vật lý (Rigidbody)

# Cập nhật và Cập nhật cố định

---

Cập

nhật `void Update ()`

```
{ Debug.Log(" Thời gian cập nhật :"  
+ Time.deltaTime); }
```

FixedUpdate

`void FixedUpdate ()`

```
{ Debug.Log(" Thời gian cập nhật cố  
định:" +  
Time.deltaTime); }
```

## Trình tự các chức năng sự kiện của Unity được trực quan hóa

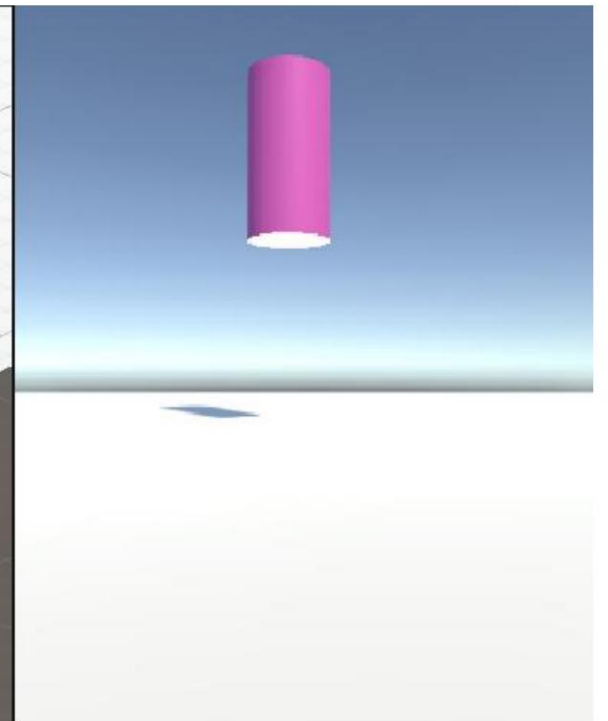
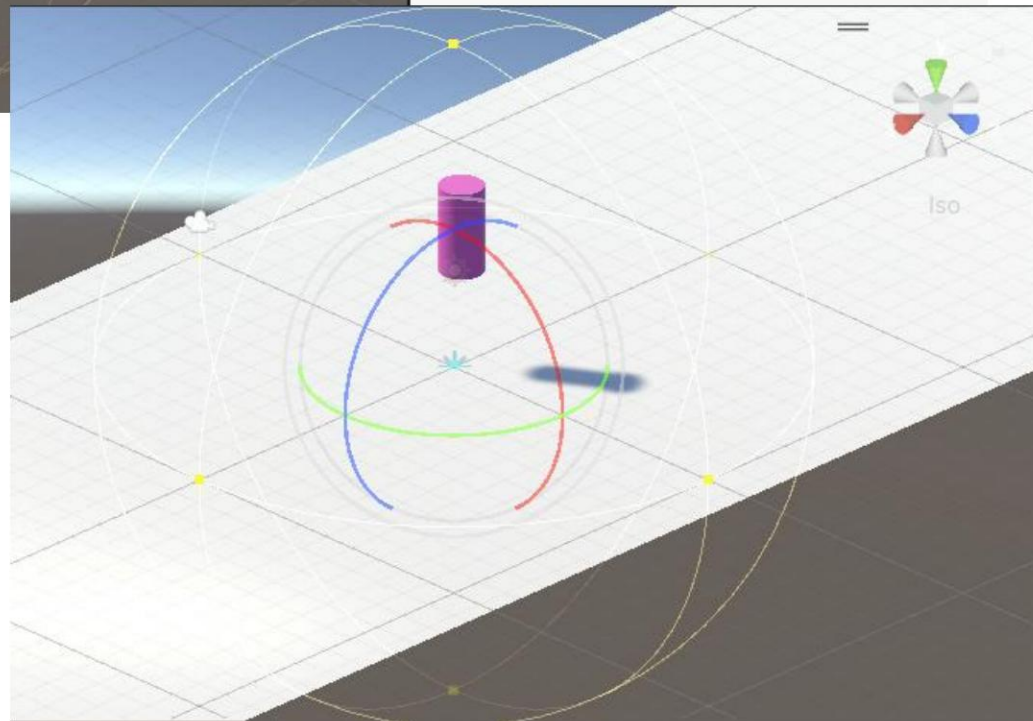
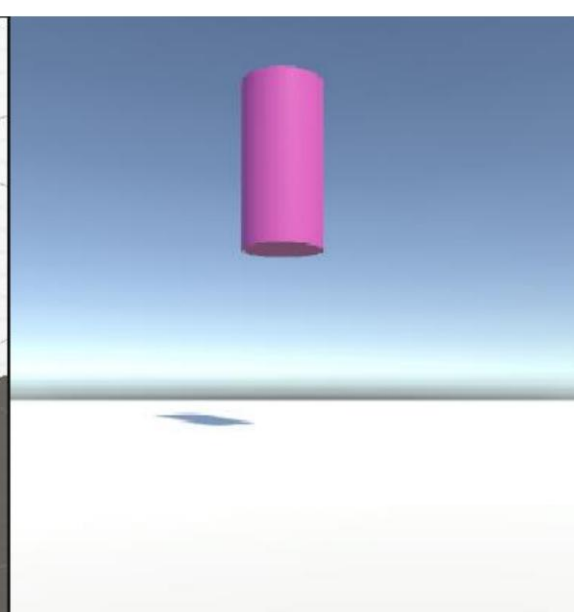
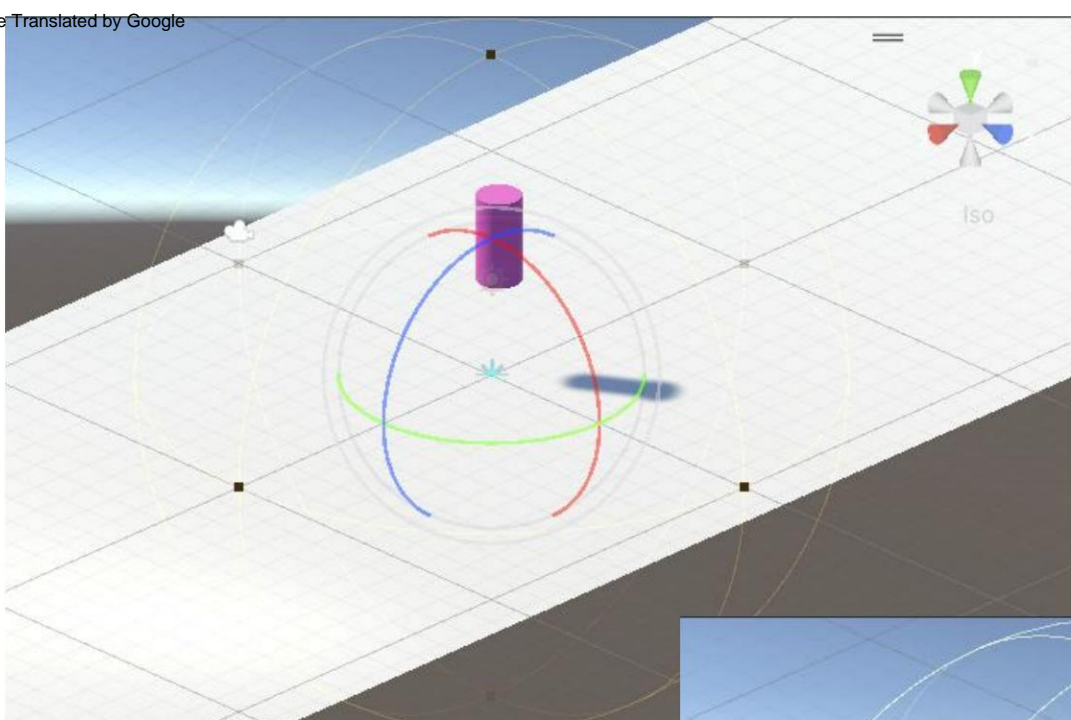


Nguồn: <https://gamedevbeginner.com/start-vs-awake-in-unity/>

# Bật và Tắt các Thành phần

---

```
lớp công khai EnableComponents : MonoBehaviour {  
  
    riêng tư Light myLight;  
    void Start() {  
  
        myLight = GetComponent<Ánh sáng>();  
    }  
  
    void Cập nhật()  
    {  
        nếu(Đầu vào.GetKeyUp(KeyCode.Space)) {  
  
            myLight.enabled = !myLight.enabled;  
        }  
    }  
}
```



# Dịch và Xoay

---

- `public void Translate(Vector3 translation);` •

`public void Translate(Vector3 translation, Space relativeTo = Không gian.  
Bản thân);`

- `relativeTo = Space.Self`: Chuyển động đư ợc áp dụng t ư ơng đ ối v ớ i biến đ ổi tr ực c ục b ộ.
- `relativeTo = Space.World`: Chuyển động đư ợc áp dụng t ư ơng đ ối v ớ i h ệ t ọa đ ộ th ế gi ới

# Dịch và Xoay

```
biến đổi. Dịch(0, 0, 1);
```

```
biến đổi. Dịch (vector3 mới (0,0,1));
```

Di chuyển một dọc theo trục z,  
mỗi khung hình

```
Transform.Translate(0, 0, Time.deltaTime);
```

```
tốc độ float công khai = 10f;
```

```
biến đổi. Dịch (Vector3. chuyển tiếp * tốc độ * Thời gian. deltaTime);
```

Di chuyển một theo trục  
z, đơn vị/giây

```
Transform.Translate(0, Time.deltaTime, 0, Space.World);
```

```
tốc độ di chuyển của float công khai = 10f;
```

```
biến đổi.Dịch(-Vector3.up*tốc độ di chuyển*Thời gian.deltaTime, Không gian.Thế  
giới);
```

Di chuyển một đơn vị theo trục  
y, đơn vị/giây

# Dịch và Xoay

---

```
công khai float turn = 10f;
```

```
biến đổi. Xoay (Vector3.up, -turn * Time.deltaTime);
```

```
công khai float turn = 10f;
```

```
biến đổi. Xoay (Vector3.up, turn * Time.deltaTime);
```



# Nhìn vào

---

```
sử dụng System.Collections; sử  
dụng System.Collections.Generic; sử dụng  
UnityEngine;
```

```
lớp công khai CameraLookAt : MonoBehaviour {
```

```
    công khai Biến đổi mục tiêu;
```

```
    void Cập nhật()
```

```
    {
```

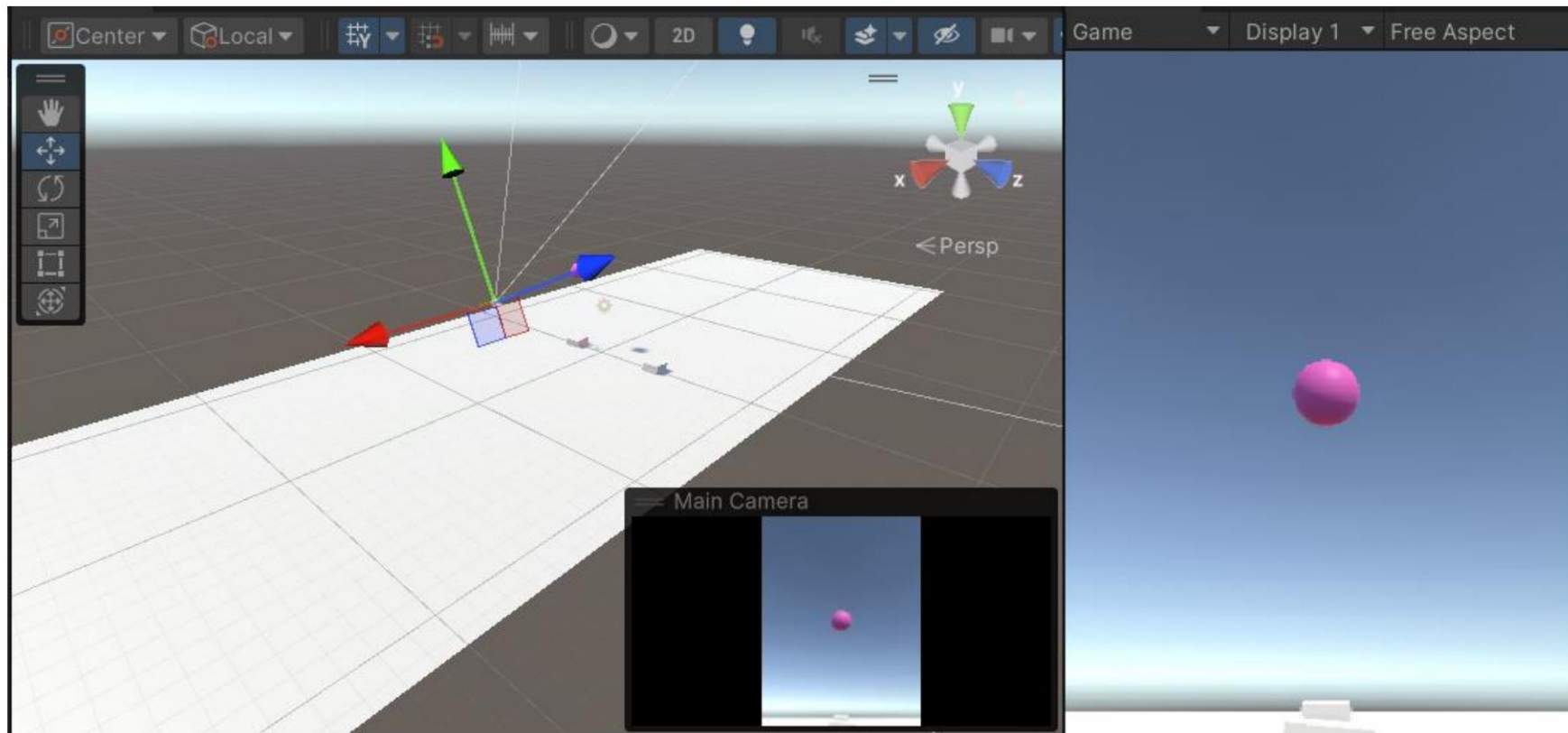
```
        transform.LookAt(mục tiêu);
```

```
    }
```

```
}
```

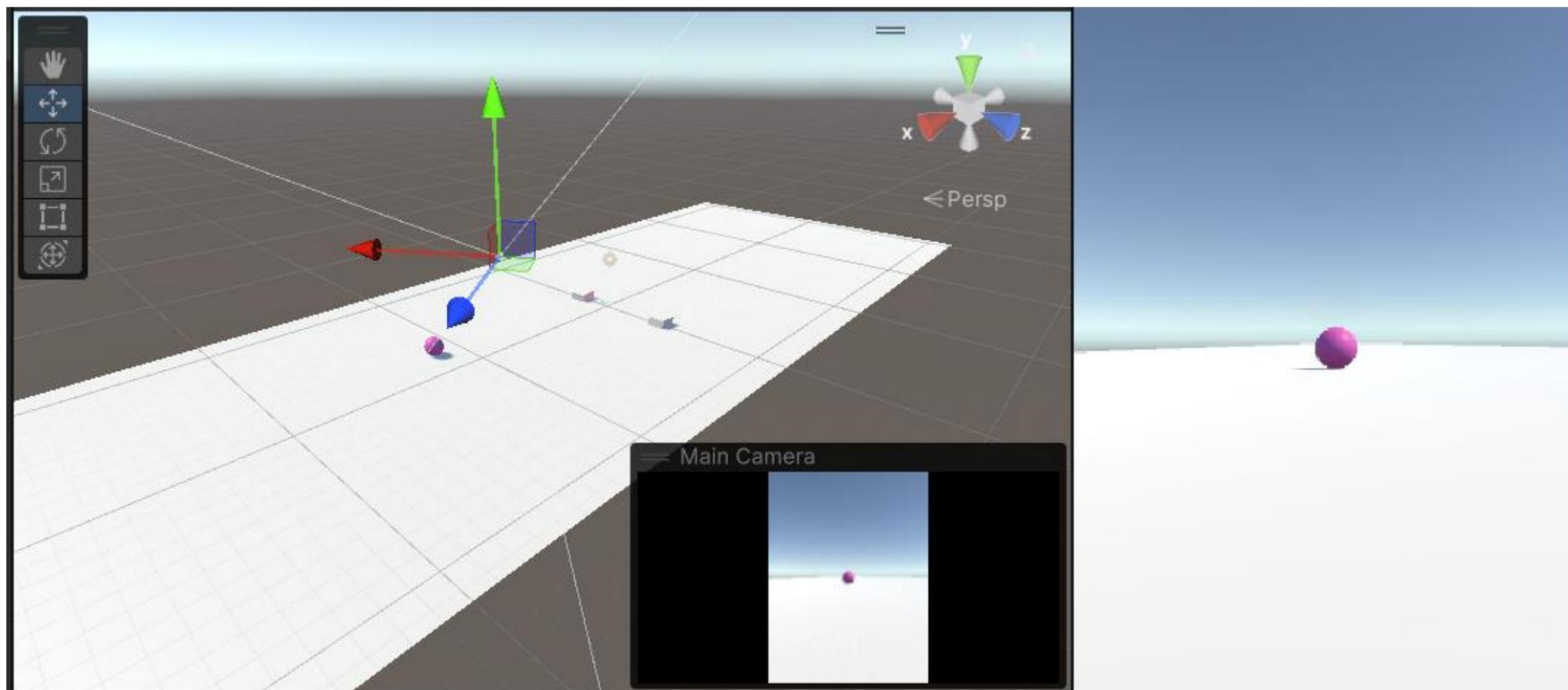
Mục tiêu là đối tượng được quan sát

# Nhìn vào



# Nhìn vào

---



# Nội suy tuyến tính

---

- Nội suy tuyến tính là tìm một giá trị là một phần trăm nào đó giữa hai giá trị đã cho. `Mathf.Lerp(từ, đến, phần trăm)`

```
// Trong trường hợp này, result  
= 4 float result = Mathf.Lerp (3f, 5f, 0.5f);
```

```
Vector3 từ = new Vector3 (1f, 2f, 3f);  
Vector3 thành = Vector3 mới (5f, 6f, 7f);
```

```
// Ở đây kết quả = (4, 5, 6)  
Kết quả Vector3 = Vector3.Lerp (từ, đến, 0,75f);
```

# Nội suy tuyến tính

---

- Trong cấu trúc Color, màu sắc đư ợc biểu diễn bằng 4 số thực đại diện cho đỏ, xanh lam, xanh lục và alpha.

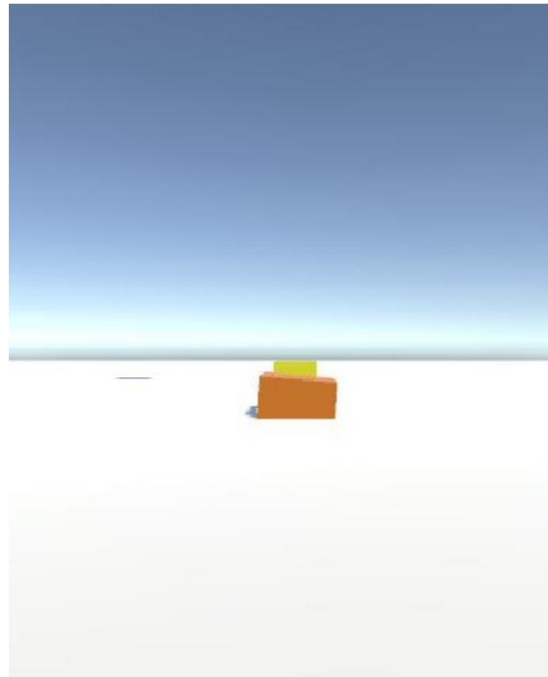
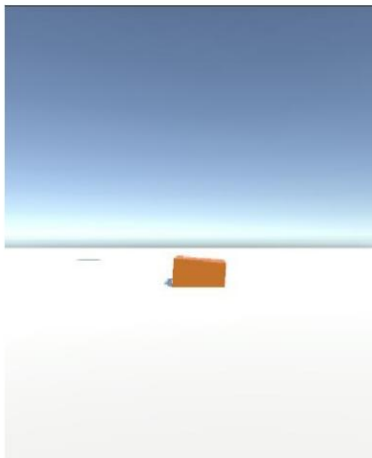
```
void Cập nhật()  
{  
    cường độ ánh sáng = Mathf.Lerp(cường độ ánh sáng, 8f, 0,5f); cường  
    độ ánh sáng = Mathf.Lerp(cường độ ánh sáng, 8f, 0,5f * Thời gian.deltaTime);  
}
```

# Hủy hoại

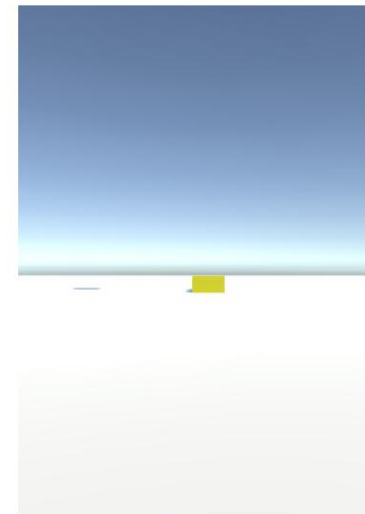
- Sử dụng hàm `Destroy()` để xóa `GameObject` và `Components` khi chạy. Một tập lệnh có tên `DestroyObject` được đính kèm vào đối tượng màu vàng.

Hủy(đối tượng trò chơi);

Hủy(gameObject,10f);



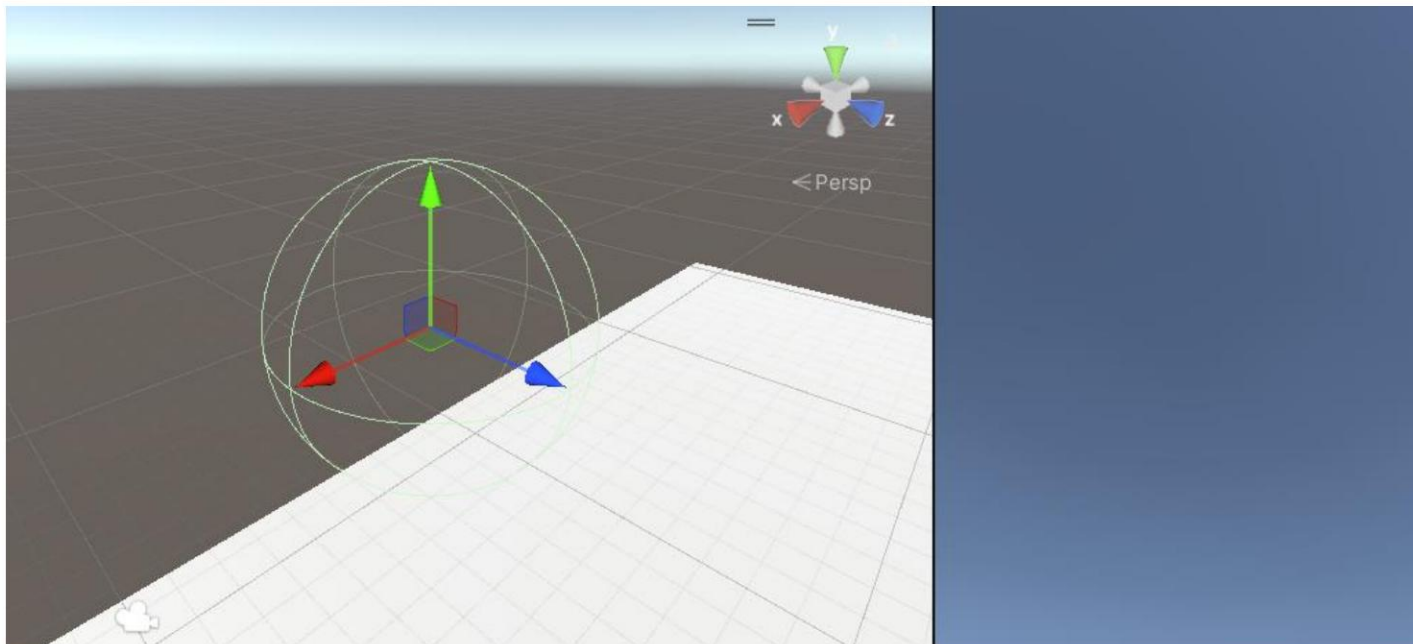
```
công khai GameObject màu cam;  
void Start()  
{  
    Phá hủy(màu cam,5f);  
}
```



# Phá hủy thành phần

- Phá hủy thành phần kết xuất lưu ở để đối tượng không còn được kết xuất một cách rõ ràng nữa

Hủy (`GetComponent<MeshRenderer>()`);



# Quản lý đầu vào

---

Trình quản lý đầu vào sử dụng các loại điều khiển sau:

- Phím là bất kỳ phím nào trên bàn phím vật lý, chẳng hạn như W, Shift hoặc phím cách.
- Nút là bất kỳ nút nào trên bộ điều khiển vật lý (ví dụ: tay cầm chơi game), chẳng hạn như nút X trên điều khiển từ xa.
- Một trục ảo (số nhiều: axes) được ánh xạ tới một điều khiển, chẳng hạn như một nút hoặc một phím. Khi người dùng kích hoạt điều khiển, trục nhận được một giá trị trong phạm vi  $[-1..1]$ . Bạn có thể sử dụng giá trị này trong các tập lệnh của mình.



Nhận chìa khóa

---

**GetKey/Down/Up**

**KeyCode.Space**



**GetButtonDown: True**

**GetButton: True**

**GetButtonUp: False**

Nhận chìa khóa

---

**GetKey/Down/Up**  
**KeyCode.Space**



**GetButtonDown: False**

**GetButton: True**

**GetButtonUp: False**

Nhận chìa khóa

---

**GetKey/Down/Up**  
**KeyCode.Space**



**GetButtonDown:** **False**

**GetButton:** **False**

**GetButtonUp:** **True**

Nhận chìa khóa

---

**GetKey/Down/Up**  
**KeyCode.Space**



**GetButtonDown: False**

**GetButton: False**

**GetButtonUp: False**

## Nhận chìa khóa

---

- Đầu vào. `GetKeyUp(KeyCode.Space)`
- Đầu vào. `GetKeyDown(KeyCode.R)`
- Đầu vào. `GetKeyDown(KeyCode.B)`

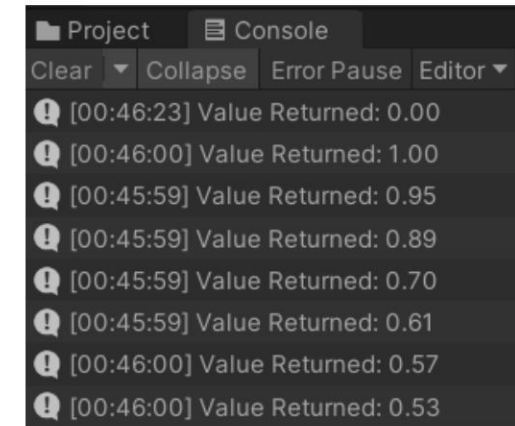
# Nhận Axis

---

- Trả về giá trị float giữa âm một và dư ơng một  $[-1,1]$
- Truy cập Chính sửa > Cài đặt dự án > Đầu vào > Trục
- Trọng lực: hành vi tiêu chuẩn của GetAxis theo chiều ngang, ảnh hưởng đến tốc độ tỷ lệ trở về số không sau khi nút đư ợc nhả ra.
- Dead: sử dụng cần điều khiển để biểu diễn trục của chúng ta. Giá trị dead càng cao thì vùng dead càng lớn.
- Nhạy cảm: kiểm soát tốc độ giá trị trả về của đầu vào đạt đến một hoặc trừ một. Số càng cao thì phản hồi càng tốt. Số càng thấp thì càng mư ợt mà
- Snap: cho phép trả về số không nếu cả số dư ơng và số âm đều đư ợc giữ.

# Nhận Axis

```
phạm vi float công khai = 2;  
công khai Văn bản textOutput ;  
void Cập nhật()  
{  
    float h = Input.GetAxis("Ngang");  
    float xPos = h * phạm vi;  
    biến đổi.vị trí = vector3 mới (xPos, 2f, 0);  
    Debug.Log("Giá trị trả về: " + h.ToString("F2"));  
}
```



# Nhận Bài tập Trục

---

- Trục thẳng đứng
- Trục kép



# Nhận thành phần

---

- Script như một thành phần tùy chỉnh để GetComponent có thể truy cập các script và thành phần khác
- GameObject.GetComponent: **T** Tham chiếu đến thành phần có kiểu **T** nếu tìm thấy, nếu không thì trả về giá trị null.

```
lớp công khai UsingComponents : MonoBehaviour {  
  
    công khai GameObject otherGameObject;  
    riêng tư AnotherScript another;  
    riêng tư AnotherTwoScript anothertwo;  
    riêng tư BoxCollider boxCol;  
  
    void Awake () {  
  
        một cái khác = GetComponent<AnotherScript>();  
        một cái khác = otherGameObject.GetComponent<AnotherTwoScript>(); boxCol  
        = otherGameObject.GetComponent<BoxCollider>();  
  
    } void Bắt đầu()  
    {  
  
        boxCol.size = new Vector3(3,3,3);  
  
        Debug.Log(" Điểm của ngư ời chơi là " + another.playerScore); Debug.Log("   
        Ngư ời chơi đã chết " + anothertwo.numberOfPlayerDeaths + "lần");  
  
    }  
}
```

# Nhận thành phần

---

```
lớp công khai AnotherScript : MonoBehaviour {  
    công khai int playerScore = 9001;  
}
```

AnotherScript.cs

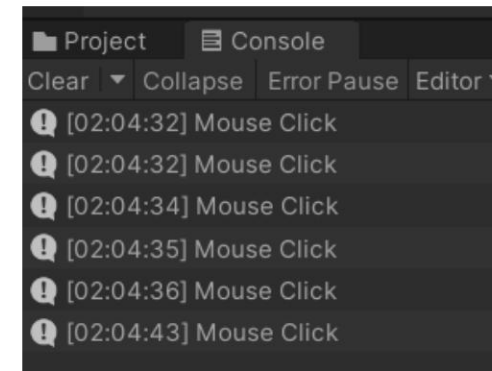
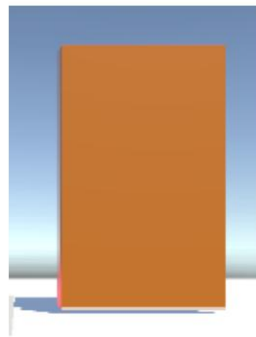
```
lớp công khai AnotherTwoScript : MonoBehaviour {  
    công khai int numberOfPlayerDeaths = 3;  
}
```

AnotherTwoScript.cs

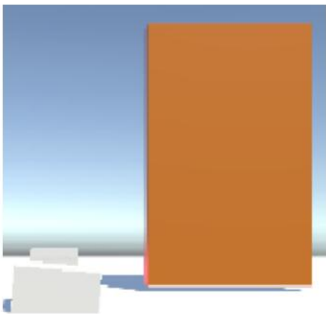
# Nhấp chuột

---

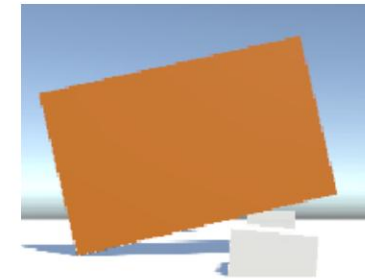
```
void OnMouseDown () {  
  
    Debug.Log(" Nhấp chuột");  
}
```



# Nhấp chuột



```
lớp công khai MouseClick : MonoBehaviour {  
  
    Rigidbody riêng tư rb;  
  
    riêng tư void Awake()  
    {  
        rb = GetComponent<Thân cứng>();  
    }  
  
    void OnMouseDown () {  
  
        rb.AddForce(-transform.forward * 200f);  
        rb.useGravity = đúng;  
    }  
}
```

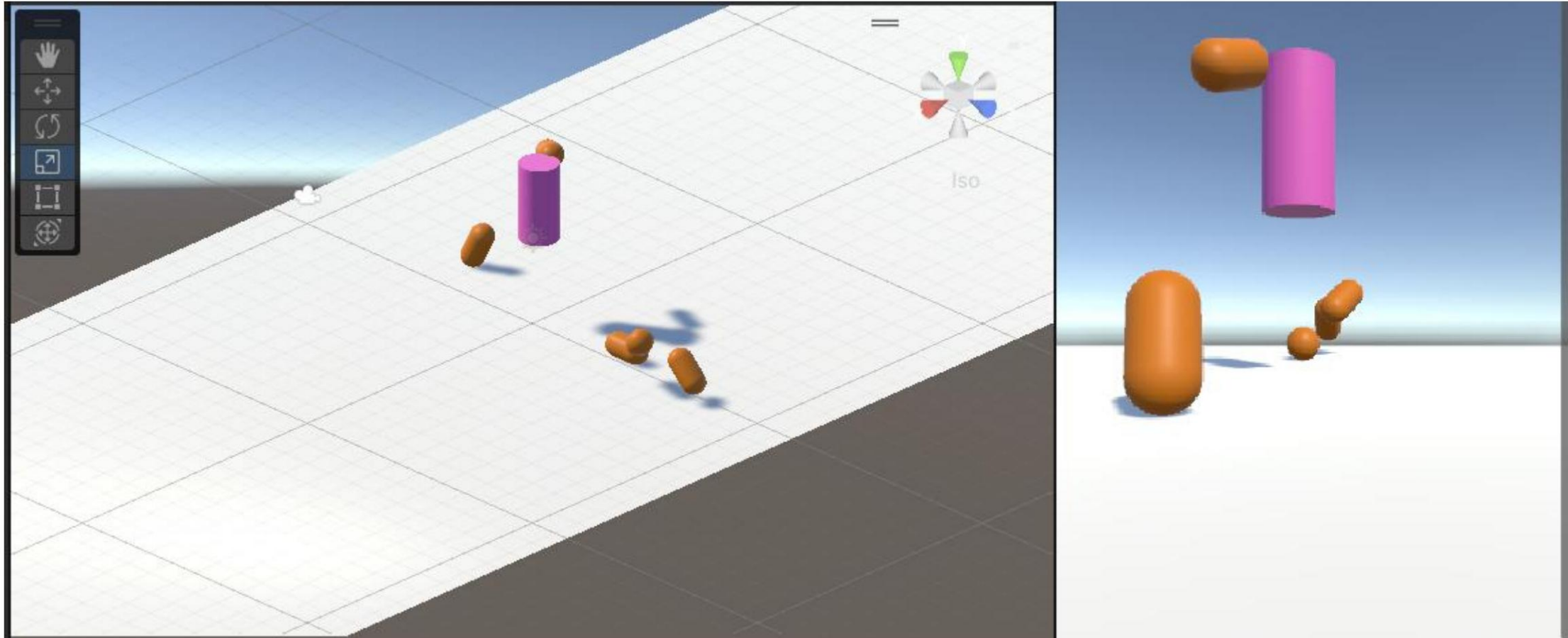


# Khởi tạo

---

- Một chức năng để tạo bản sao của các đối tượng trò chơi. Điều này thường được sử dụng trong bối cảnh sao chép một prefab.
- `public static void Destroy(Object obj, float t = 0.0F)`
  - Obj: Đối tượng cần phá hủy
  - T: Khoảng thời gian tùy chọn để trì hoãn trước khi phá hủy đối tượng.

# Khởi tạo



# Khởi tạo

---

```
lớp công khai UsingInstantiate : MonoBehaviour {  
  
    công cộng Đạn vật rắn đúc sẵn;  
  
    void Cập nhật()  
    {  
        nếu(Input.GetButtonDown("Fire1")) {  
  
            Khởi tạo(projectilePrefab);  
        }  
    }  
}
```



# Khởi tạo

---

```
lớp công khai UsingInstantiate : MonoBehaviour {  
  
    công khai Rigidbody rocketPrefab;  
    công khai Transform barrelEnd;  
    void Update() {  
  
        nếu(Input.GetButtonDown("Fire1")) {  
  
            Rigidbody rocketInstance;  
            rocketInstance = Khởi tạo(rocketPrefab, barrelEnd.position,  
barrelEnd.rotation) là Rigidbody;  
            rocketInstance.AddForce(barrelEnd.forward * 5000);  
        }  
    }  
}
```

# Phá hủy Khởi tạo

---

lớp công khai ProjectileDestruction :

Hành vi đơn điệu

```
{  
    // Start đư ợc gọi trư ớc khi cập nhật khung đầu tiên void  
Start()  
    {  
  
        Phá hủy (gameObject, 1.5f);  
    }  
}
```

Đính kèm scripte vào đối tượng prelab

# Gọi

---

- `public void Invoke(string methodName, float time)`
  - `public void InvokeRepeating(string methodName, float time, float repeatRate);`
  - `public void CancelInvoke();`
- 
- `methodName`: Tên của phương thức cần gọi.
  - Thời gian: Bắt đầu gọi sau `n` giây.
  - `repeatRate`: Lặp lại sau mỗi `n` giây.
  - Hủy tất cả các lệnh gọi `Invoke` trên `MonoBehaviour` này.

# Gọi

---

```
lớp công khai InvokeObject : MonoBehaviour {
```

```
    công khai GameObject mục
```

```
    tiêu; void
```

```
    Start() {
```

```
        Gọi ("SpawnObject", 2);
```

```
    }
```

```
    void SpawnObject() {
```

```
        Khởi tạo(mục tiêu, Vector3 mới (0, 2, 0),
```

```
        Quaternion.identity); }
```

```
}
```

Độ trễ tính bằng giây trước khi  
gọi phương thức: 2

Tên của phương pháp gọi

# Bài tập

---

- Gọi một hàm có tên `SpawnObjects` để khởi tạo đối tượng ngẫu nhiên
  - `InvokeRepeating(?,?,?)`
  - Tạo `x, y` trong phạm vi `[1,2]`
  - Khởi tạo(`target, new Vector3(x, 2, y), Quaternion.identity`);