

# Bài 7: Hiệu ứng và tối ưu hóa đồ họa Game

*Giảng viên:*

# MỤC TIÊU

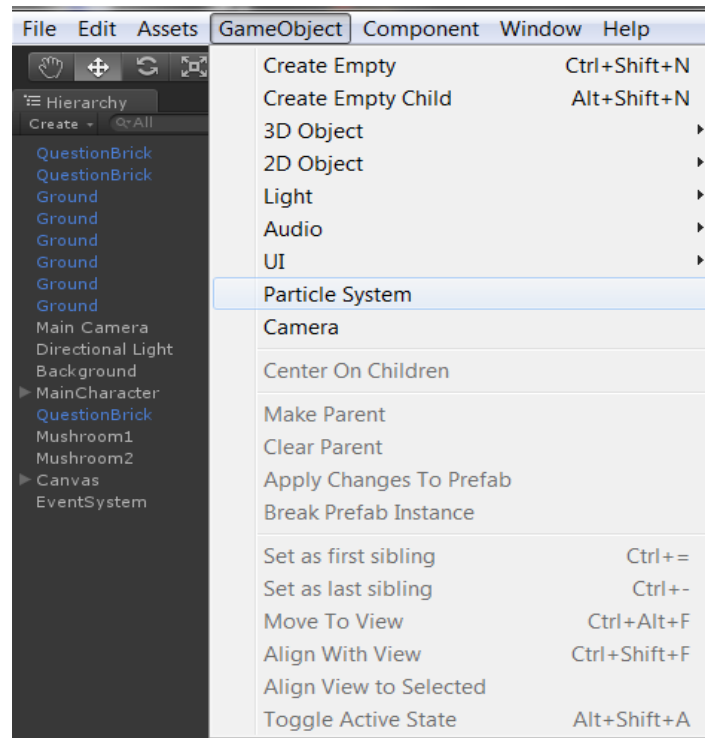
- Cấu trúc Project
- A. Khởi tạo và cấu hình dự án Game 2D
- B. Tạo các đối tượng cơ bản
  - 1. *Game Object*
  - 2. *Sprite*
  - 3. *Animation và điều khiển hành động nhân vật*
  - 4. *Prefab*
  - 5. *Script và điều khiển máy trạng thái*
  - 6. *Thành phần vật lý và xử lý va chạm*
  - 7. *Thiết kế UI*
  - **8. *Sử dụng Particle System***
  - **9. *Chuyển đổi màn chơi***
  - **10. *Sound***
  - **11. *Design Pattern trong Game***

# Nội dung

- Cấu trúc Project
- A. Khởi tạo và cấu hình dự án Game 2D
- B. Tạo các đối tượng cơ bản
  - 1. *Game Object*
  - 2. *Sprite*
  - 3. *Animation và điều khiển hành động nhân vật*
  - 4. *Prefab*
  - 5. *Script và điều khiển máy trạng thái*
  - 6. *Thành phần vật lý và xử lý va chạm*
  - 7. *Thiết kế UI*
  - **8. *Sử dụng Particle System***
  - **9. *Chuyển đổi màn chơi***
  - **10. *Sound***
  - **11. *Design Pattern trong Game***

# Sử dụng Particle System

- Particle System là một trong những kỹ thuật tạo ra các hiệu ứng cháy nổ hay sương, khói... được sử dụng rất thường xuyên trong game.
- Các bạn chọn Menu/GameObject/Particle System như hình sau:



# Sử dụng Particle System

- Tiếp theo các bạn thêm một Script cho đối tượng Particle System này, và đặt tên là ParticleSystemBehaviour.cs Và edit nội dung của nó thành như sau:

```
3
4 public class ParticleSystemBehaviour : MonoBehaviour {
5
6     // Use this for initialization
7     void Start () {
8         Destroy (gameObject, 3);
9     }
10
11    // Update is called once per frame
12    void Update () {
13
14    }
15 }
16 |
```

- Destroy(gameObject, 3) --> có nghĩa rằng đối tượng này sau khi xuất hiện 3s sẽ tự động hủy.

# Sử dụng Particle System

- Tiếp theo ta kéo đối tượng Particle System vào thư mục Prefabs để tạo prefab cho đối tượng này
- Sau đó ta có thể xóa đối tượng Particle System này ở cửa sổ Hierarchy đi, lúc nào có và chạm ta mới sinh ra một đối tượng Particle System này.

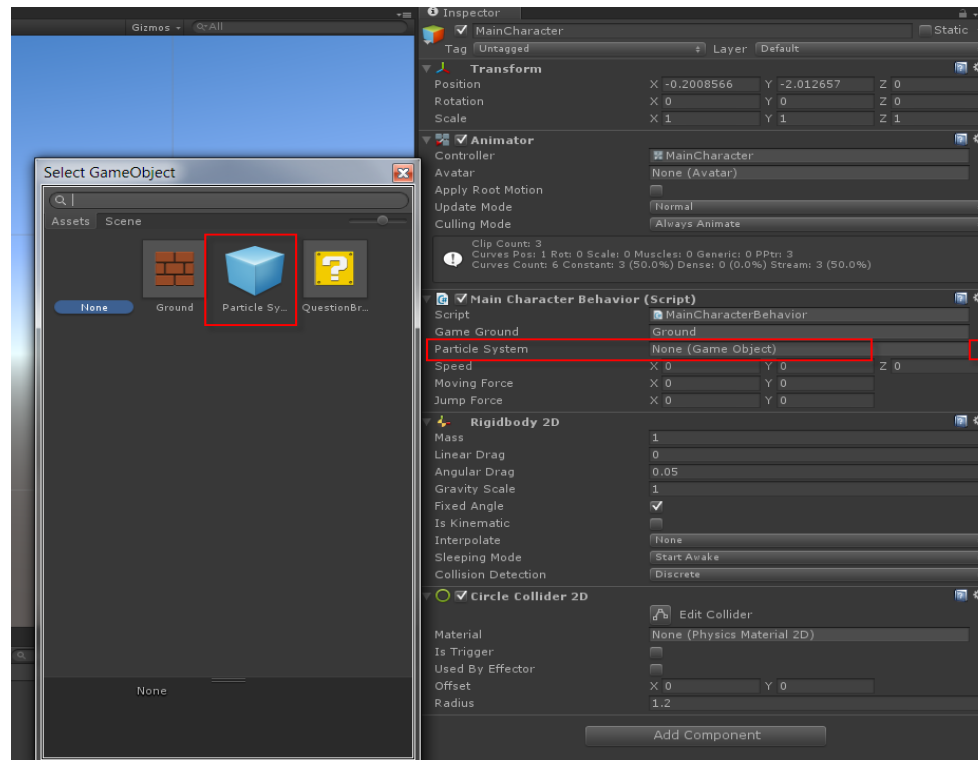
# Sử dụng Particle System

- Tiếp theo ở MainCharacterBehaviour.cs ta khai báo thêm một thuộc tính là particleSystem như sau:

```
public GameObject gameGround;  
public GameObject particleSystem;  
public Vector3 speed;  
  
public Vector2 movingForce;  
public Vector2 jumpForce;
```

# Sử dụng Particle System

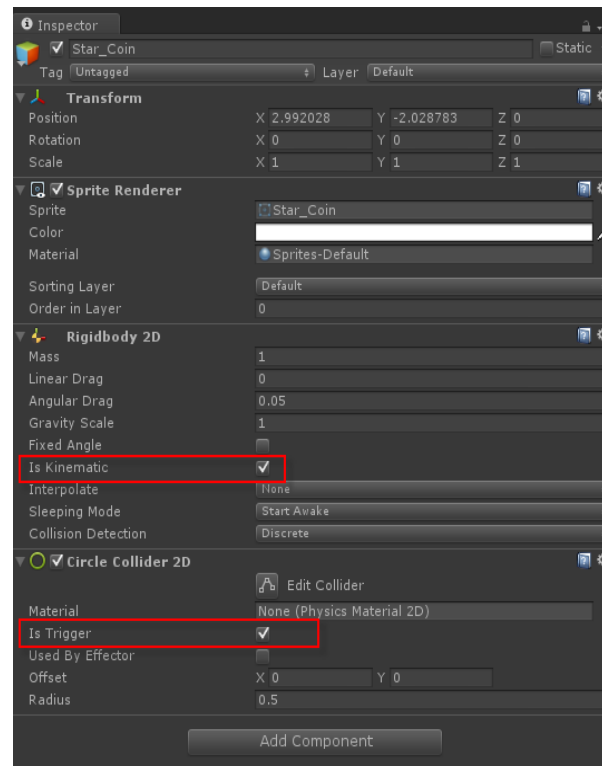
- Chọn đối tượng MainCharacter, ở cửa sổ Inspector, thành phần Script ta sẽ thấy thêm một thuộc tính đó là Particle System như hình:
- Ta chọn vào nút khoan tròn màu đỏ, sau đó một cửa sổ các Prefabs sẽ hiện ra, ta chọn cho nó là đối tượng Particle System.





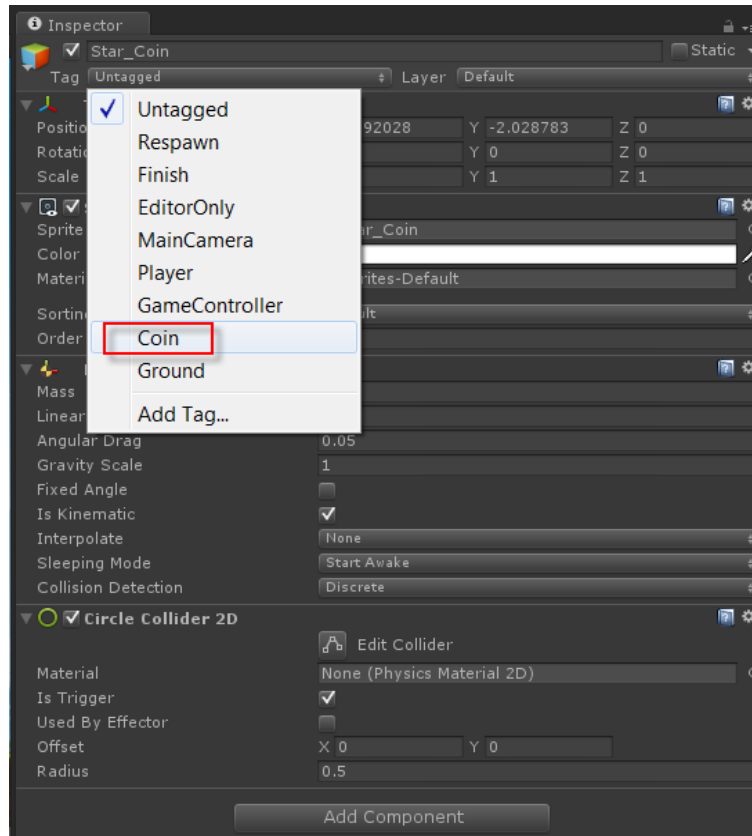
# Sử dụng Particle System

- Tiếp theo, ta sẽ thêm Sprite đồng tiền, để khi đối tượng chạm vào đồng tiền.
- Ta phải thêm đầy đủ các thành phần vật lý, xử lý va chạm và thêm tag cho đồng tiền.
- **Chú ý:** là ta sẽ chọn: Is Kinematic và Is Trigger cho đối tượng đồng tiền.



# Sử dụng Particle System

**Chú ý:** Nếu chưa có tag "Coin" ta có thể chọn Add Tag rồi thêm.



# Sử dụng Particle System

- Ta đặt đồng tiền ở một vị trí sao cho khi MC chạy tới thì có thể va chạm với đồng tiền.



# Sử dụng Particle System

- Tiếp theo, ở MainCharacterBehaviour.cs ta sẽ thêm đoạn xử lý này:

```
void OnCollisionEnter2D(Collision2D other)
{
    //other.gameObject
    //Debug.Log ("OnCollisionEnter2D with object has tag = " + other.gameObject.tag);
}
void OnTriggerEnter2D(Collider2D other)
{
    //other.gameObject
    Debug.Log ("OnTriggerEnter2D with object has tag = " + other.gameObject.tag);
    if (other.gameObject.tag == "Coin")
    {
        Destroy(other.gameObject);
        Instantiate(particleSystem,
                    other.gameObject.transform.position,
                    other.gameObject.transform.localRotation);
    }
}

void OnMouseDown()
{
    .....
```

## Sử dụng Particle System

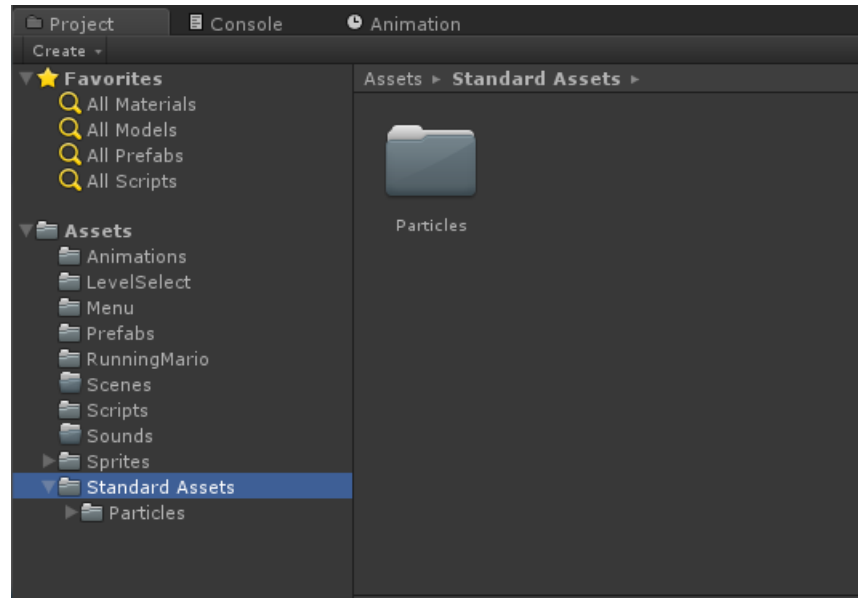
- Như vậy mỗi khi có va chạm giữa MainCharacter với một trigger nào đó, hàm này sẽ được gọi và chúng ta sẽ kiểm tra nếu đối tượng va chạm có Tag là "Coin" ta sẽ xoá đối tượng coin\_gold đi bằng lệnh Destroy(other.gameObject) và đặt vào tại đó một đối tượng Particle System bằng lệnh **Instantiate** (Đối tượng Particle System này sau 3s sẽ biến mất theo như đã thiết lập ở trên).
- Ta có thể nhấn nút Play để kiểm tra lại kết quả.

# Sử dụng Particle System

- Ngoài việc sử dụng đối tượng Particle có sẵn, để cho đẹp hơn ta có thể mua thêm các Particle System khác hoặc sử dụng các Particle System free trên Internet hoặc cộng đồng Unity chia sẻ.
- Ta import các gói assets free của Unity như sau:
- Bước 1. Import Package -> Particles
- Bước 2: chọn các asset cần, và nhấn nút Import

# Sử dụng Particle System

- Bước 3: các assets sẽ được import vào thư mục như sau:




- Sau đó bạn chỉ việc kéo thả các Asset này thay vì sử dụng đối tượng Particle System có sẵn.



# DEMO

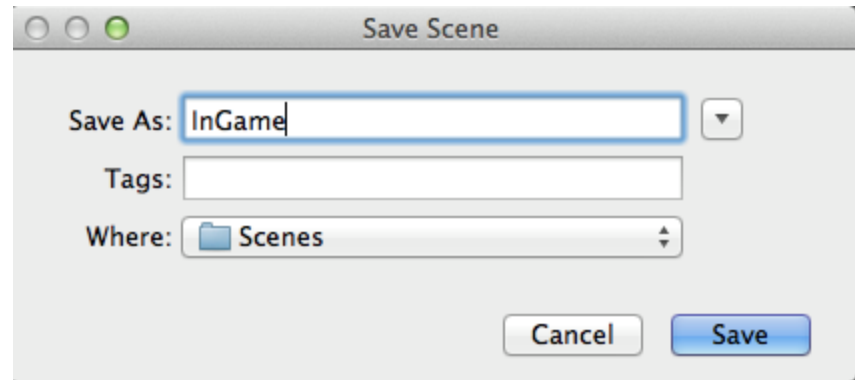
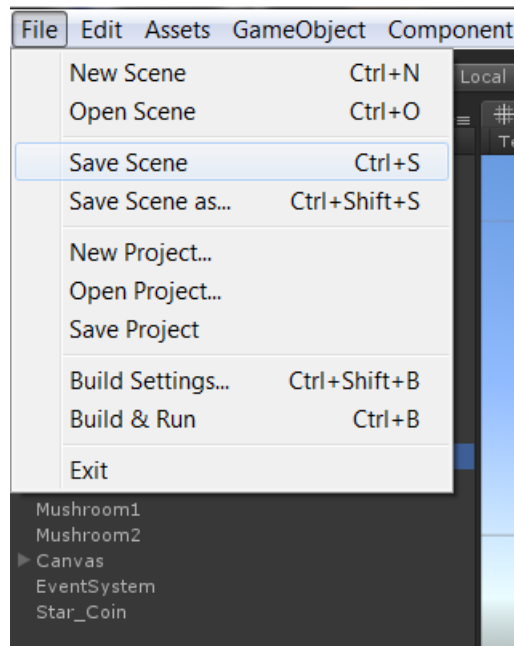
Particel  
System

A white hand cursor icon, resembling a computer mouse pointer, is pointing at the letter 'O' in the word 'DEMO'. The hand is stylized with a white outline and a white fill.



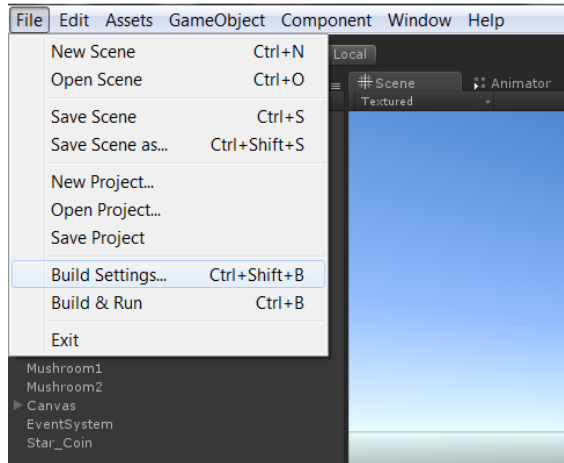
# Chuyển đổi màn chơi

- Trong một game sẽ có nhiều màn chơi, hoặc nhiều cảnh game,.
- Ví dụ đơn giản khi đối tượng rơi xuống thì game sẽ kết thúc và hiện ra màn hình thông báo là Game Over chẳng hạn.
- Đầu tiên, ta save Scenes hiện thời lại và đặt tên là InGame.
- Để dễ quản lý ta sẽ lưu trong thư mục Scenes của thư mục Assets.

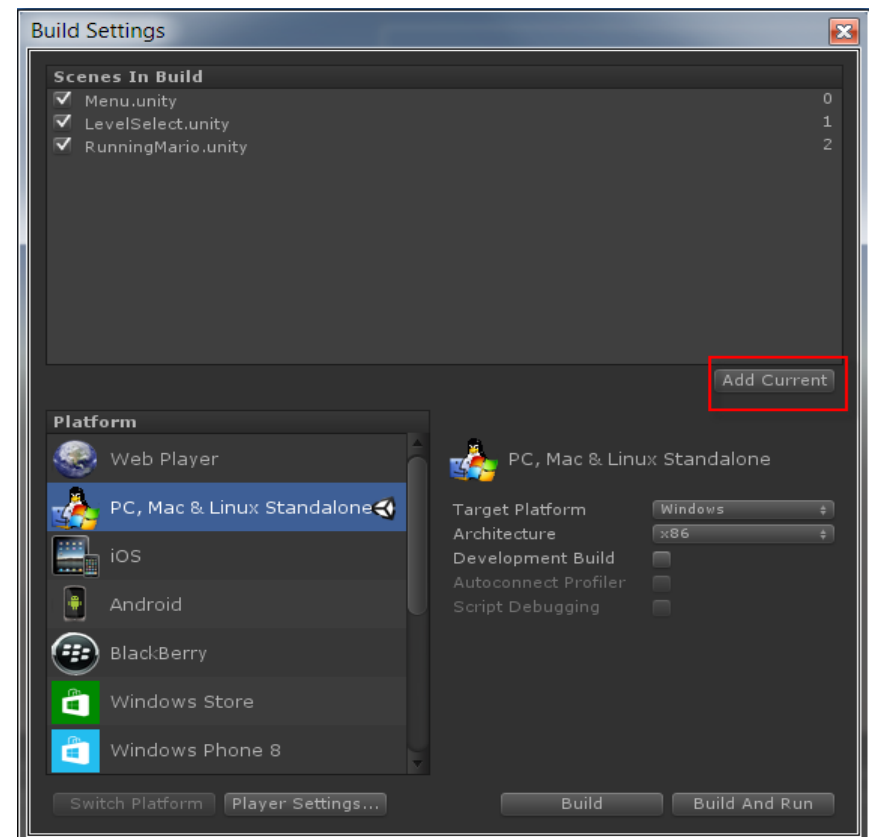


# Chuyển đổi màn chơi

- Tiếp theo ta vào Build Setting:

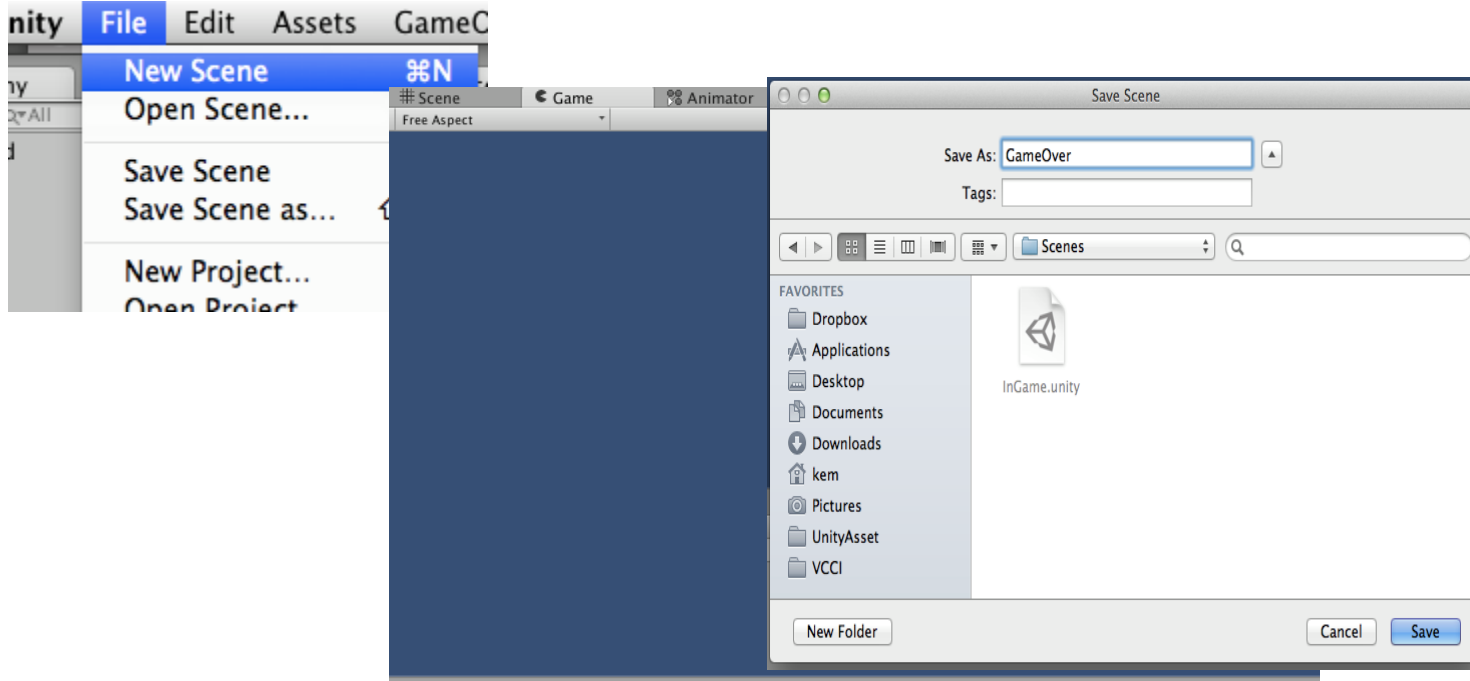


- Một cửa sổ mới hiện ra, ta chọn Add Current.



# Chuyển đổi màn chơi

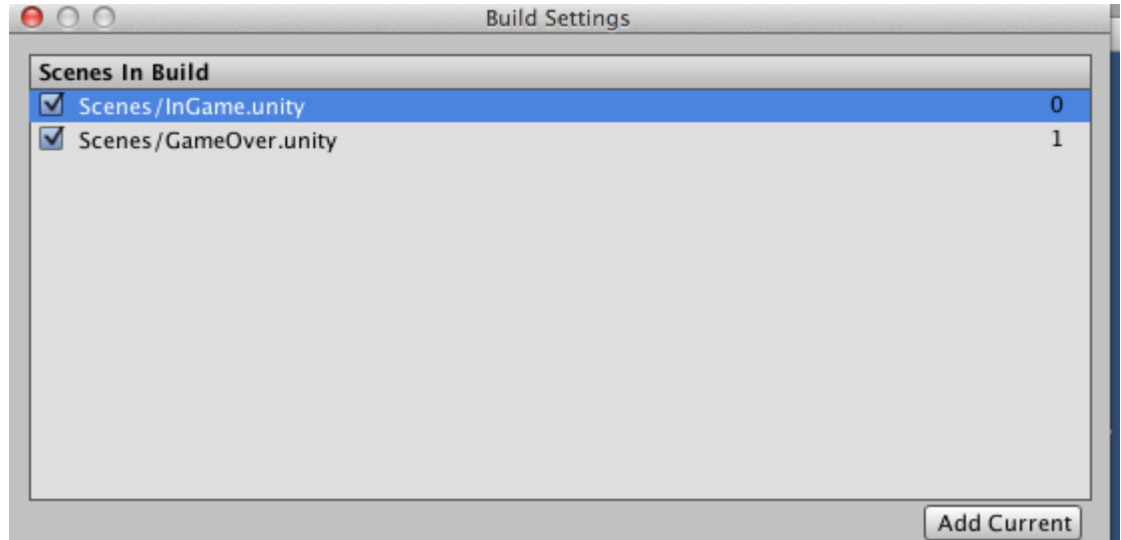
- Tắt cửa sổ mới hiện ra, tiếp theo các bạn tạo mới một Scenes, lưu lại với tên là GameOver



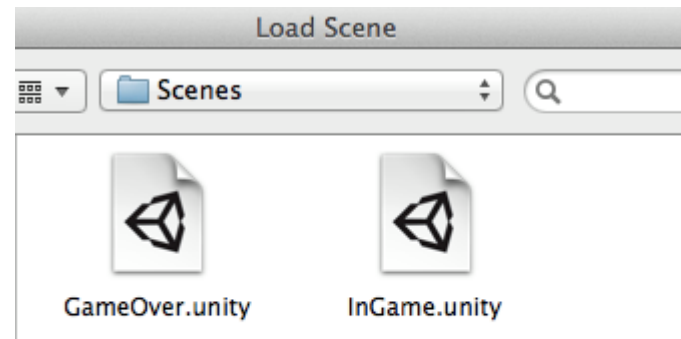
- Tiến hành vào Build Setting, thêm Scenes GameOver vào.

# Chuyển đổi màn chơi

- **Chú ý:** Scenes nào cần hiển thị đầu tiên thì ta sẽ thêm vào Build Setting đầu tiên, hoặc chúng ta có thể kéo thả ngay tại cửa sổ Build Settings.



- Bây giờ ta có hai Scenes là InGame và GameOver. Tiến hành Save lại, sau đó Open Scenes InGame lại.



# Chuyển đổi màn chơi

- Xử lý chuyển đổi màn chơi:  
Ở MainCharacter, ta sẽ thêm đoạn lệnh này ở hàm Update:

```
void Update () {  
    //Switch states  
    if (Input.GetKey(KeyCode.LeftArrow) || Input.GetKey(KeyCode.RightArrow))  
    {  
        animator.SetBool("isIdle", false);  
        animator.SetBool("isRunning", true);  
    }  
    else  
    {  
        animator.SetBool("isRunning", false);  
        animator.SetBool("isIdle", true);  
    }  
  
    if (gameObject.transform.position.y < -3.0f)  
    {  
        Application.LoadLevel("GameOver");  
    }  
}
```

- Ta sẽ điều khiển nhân vật ra ngoài nền, để đối tượng rơi xuống khi giá trị y của position < -5 game sẽ tự động chuyển qua màn hình GameOver.



**DEMO**

Chuyển đổi  
màn chơi

A white hand cursor icon, resembling a computer mouse pointer, is positioned below the word 'DEMO'. The index finger is pointing upwards towards the letter 'O'.

# Sound

- **Audio Listener:** Giống như một thiết bị Microphone. Nó nhận đầu vào từ bất cứ một nguồn âm thanh nào trong Scenes và các âm thanh thông qua máy tính. Đối với hầu hết các ứng dụng, nó là ý nghĩa nhất để gắn tai nghe lên Main Camera.
- Ranh giới/độ ảnh hưởng âm thanh nghe của **Reverb Zone** được áp dụng cho toàn Scene thì có thể nghe thấy âm thanh trên bất cứ địa điểm nào của Scene.

# Sound

## Các thuộc tính của Audio Source

Phải thêm âm thanh vào Scene và chỉnh sửa thuộc tính của âm thanh bên Inspector View.

- **Audio Clip:** Chọn file âm thanh cho Scene.
- **Mute:** bật/tắt âm thanh.
- **Bypass Effects:** Lọc nhanh hiệu ứng “by-pass” to audio source. Một cách dễ dàng nhất để bật/tắt hiệu ứng(effect).



# Sound

- **Play on Wake:** Nếu enable, âm thanh sẽ được chạy ngay khi ra mắt Scene. Nếu để Disable, khi cần chạy âm thanh chúng ta phải gọi phương thức/chức năng **Play()** từ Script.
- **Priority:** Xác định độ ưu tiên của Audio Source trong số tất cả các Audio Source có trong Scene.
  - **Priority = 0:** Rất quan trọng. Sử dụng ở mức 0 cho bài nhạc để tránh bị thường xuyên trao đổi.
  - **Priority = 256:** Độ quan trọng thấp nhất(Độ ưu tiên ở mức thấp nhất).
  - **Mặc định(default) = 128.**
- **Volume:** Làm thế nào để âm thanh lớn trên một bộ phận của thế giới Scene từ Audio Source.
- **Pitch:** Số dùng thay đổi tốc độ của âm thanh. Tốc độ bằng 1 là ở mức chạy bình thường.

# Sound

- **3D Source Setting:** Cài đặt, cái mà được áp dụng cho Audio source nếu Audio Clip là một file âm thanh 3D.
  - **Pan Lever:** Cài đặt, làm thế nào để máy 3D có được hiệu ứng từ Audio source.
  - **Spread:** Cài đặt góc ảnh hưởng tới âm thanh 3D Stereo nếu Audio Clip là một âm thanh 3D.
  - **Doppler Lever:** Xác định bao nhiêu hiệu ứng âm thanh Doppler sẽ được áp dụng cho Audio Source(Nếu cài đặt là 0 thì sẽ không có hiệu ứng nào được áp dụng).
  - **Min Distance:** Với Min Distance, âm thanh sẽ đạt mức to nhất có thể. Ngoài Min Distance, nó sẽ bắt đầu suy yếu đi. Tăng MinDistance của âm thanh để được âm thanh lớn hơn trong thế giới 3D và giảm MinDistance để cho âm thanh nhỏ hơn.

# Sound

- **Max Distance:** Khoảng cách mà âm thanh dừng suy giảm. Sau khoảng cách này nó sẽ tạm ngừng không cho MinDistance tăng nữa. Tức là đây là giá trị Max mà MinDistance sẽ đạt tới.
- **Volume Rolloff:** Làm thế nào nhanh chóng mất dần âm thanh. Giá trị cao hơn, gần gũi hơn Listener có được trước khi nghe âm thanh.(Điều này được xác định bởi một Graph).
  - **Logarithmic Rolloff:** Âm thanh là lớn khi bạn ở gần nguồn âm thanh. Nhưng khi chúng ta nhận được từ các đối tượng nó sẽ giảm đi nhanh chóng một cách đáng kể.
  - **Linear Rolloff:** Càng xa nguồn âm thanh, bạn nghe thấy càng ít.
  - **Custom Rolloff:**

# Sound

- **2D Sound Setting:** Cài đặt áp dụng cho nguồn âm thanh, nếu Audio Clip là âm thanh 2D.
- **Pan 2D:** Cài đặt hiệu ứng trên nguồn âm thanh.
- **Type Rolloff:**
- **Distance Function:**
  - **Volume:** Amplitude(0.0 - 1.0) over distance.
  - **Pan:** Left(-1.0) to Right(1.0) over distance.
  - **Spread:** Angle (degrees 0.0 - 360.0) over distance.
  - **Low-Pass** (only if LowPassFilter is attached to the AudioSource): Cutoff Frequency (22000.0-10.0) over distance.

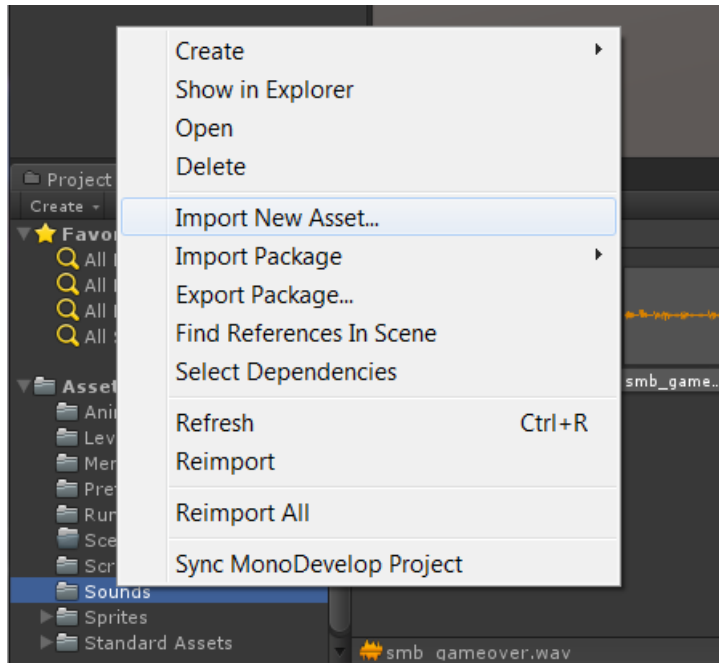
# Sound

## ■ Tạo một nguồn âm thanh:

- Import file âm thanh của bạn vào project. Nó là những Audio Clip.
- Tới **Game Object** -> **Create Empty** từ menubar.
- Trên file âm thanh đang chọn trên Game Object View, chọn Component Audio là Audio Source.
- Thuộc tính của nguồn âm thanh sẽ được hiển thị trên Inspector View.

# Sound

- Đầu tiên, ta sẽ import một tập tin mp3 vào dự án để làm nhạc nền, bằng cách click chuột phải vào thư mục Sounds, chọn Import New Assets, sau đó tìm đến tập tin MP3 của mình.

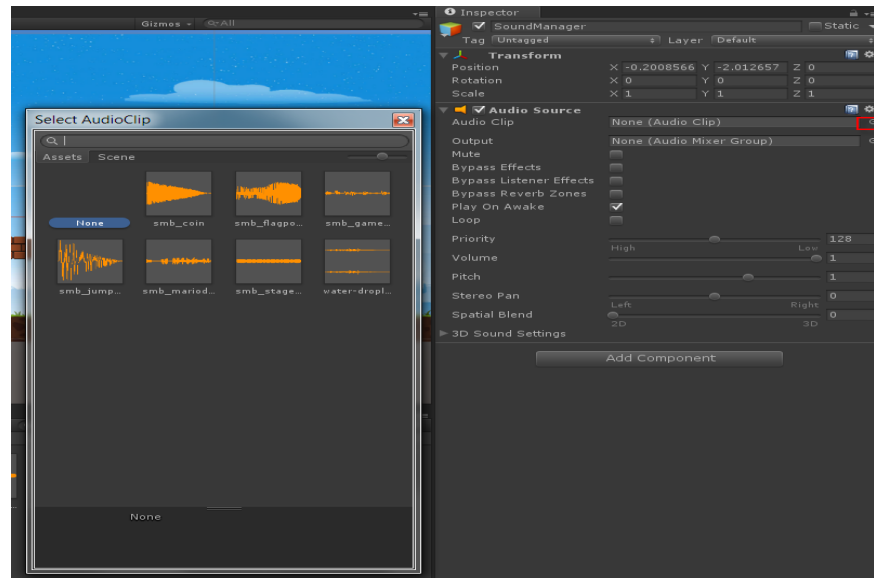


# Sound

- Để sử dụng và quản lý sound, ta sẽ quản lý thông qua một đối tượng trong game.
- Tiến hành tạo mới một Empty game object, đặt tên là SoundManager, sau đó thêm Audio Source component cho đối tượng này.

# Sound

- Ở cửa sổ Inspector, ta nhấn nút được bao tròn ô vuông màu đỏ, để chọn nhạc (Nhớ là phải import nhạc vào sẵn), chọn Start on awake để tự động chơi nhạc mỗi khi đối tượng bắt đầu khởi tạo, chọn loop nếu cho phép nhạc lặp đi lặp lại.



- Như vậy là ta đã có được nhạc nền cho game, nhấn thử nút Play để xem kết quả.



# Sound

- Tiếp theo ta sẽ tạo một script cho đối tượng SoundManager, đặt tên là SoundManagerBehaviour và viết script.  
Như vậy, khi game chạy, mỗi lần nhấn nút P là nhạc tắt, nhấn nút S là nhạc chơi lại.
- Ngoài ra các bạn có thể tìm hiểu thêm, để sử dụng linh hoạt hơn trong game tại:

<https://docs.unity3d.com/Documentation/Manual/Sound.html>.



# Design Pattern trong Game

- Design Pattern hay mẫu thiết kế, là cách xây dựng một lớp dựa trên một thiết kế nhất định nào đó. Một Pattern được sử dụng nhiều trong game đó là Singleton.
- Vậy khi nào thì sử dụng Singleton ? - Singleton sử dụng khi trong chương trình chỉ tồn tại duy nhất một instance (thể hiện or đối tượng) của một lớp nào đó.
- Ví dụ: Ta có lớp SoundManager để quản lý tất cả sound cho game, thì suốt chương trình chỉ cần duy nhất 1 thể hiện của lớp SoundManager để quản lý âm thanh trong game.
- Như vậy ta sẽ cài đặt lớp SoundManager theo mẫu Singleton.

# Design Pattern trong Game

- Về cơ bản mẫu Singleton được thiết kế như sau:
- Hàm dựng (constructor) để là **private**, để đảm bảo không tạo được đối tượng từ bên ngoài lớp, chỉ cho phép tạo đối tượng từ hàm **GetInstance** thôi.
- Câu lệnh **instance = this;** để đảm bảo biến instance được trỏ đến đối tượng duy nhất vừa mới tạo ra.

```
5 public class Singleton
6 {
7     private static Singleton instance = null;
8     private Singleton()
9     {
10         instance = this;
11     }
12     public static Singleton GetInstance()
13     {
14         //allocate instance if not exist
15         if (instance == null)
16         {
17             instance = new Singleton ();
18         }
19         return instance;
20     }
21     public static void DestroyInstance()
22     {
23         //release resources
24         instance = null;
25     }
26 }
27 }
28
29 ^^^
```

# Design Pattern trong Game

- Như vậy ở bất cứ đâu trong dự án, muốn điều khiển Sound, bật tắt bài nào đó ta chỉ việc gọi `SoundManagerBehaviour.GetInstance().Method()` là được.

```
public class BackgroundMusicBehaviour : MonoBehaviour {  
    private AudioSource[] soundList;  
  
    static private BackgroundMusicBehaviour instance = null;  
    private BackgroundMusicBehaviour()  
    {  
        instance = this;  
    }  
    public static BackgroundMusicBehaviour GetInstance()  
    {  
        if (instance == null)  
        {  
            //instance = new BackgroundMusicBehaviour();  
            Debug.Log("Need create an instance");  
        }  
        return instance;  
    }  
}
```

# Design Pattern trong Game

## Cách truyền một giá trị từ script này sang script khác.

- Nhu cầu truyền một giá trị từ script này sang script khác rất thường xuyên. Ở Unity một script được coi là một component của một GameObject, vì vậy để làm việc này ta chỉ cần xác định được đối tượng GameObject chứa script cần truyền giá trị, sau đó từ đối tượng này ta sẽ truy xuất giá trị cần thiết thông qua component script.
- Ví dụ: ở Script SoundManagerBehaviour ta muốn truy xuất đến các giá trị ở Script MainCharacterBehaviour ta làm như sau:
- Đầu tiên ta sẽ đặt cho đối tượng MainCharacter tag là: Player (Xem lại phần 2 để biết cách đặt tag).

# Design Pattern trong Game

## Cách truyền một giá trị từ script này sang script khác.

- Ở script SoundManagerBehaviour ta khai báo một đối tượng là mainCharacter kiểu GameObject.

```
// Use this for initialization
void Start () {

    mainCharacter = GameObject.FindGameObjectWithTag("Player"); // must put at Start method

    if (mainCharacter != null)
    {
        MainCharacterBehaviour script = mainCharacter.GetComponent<MainCharacterBehaviour>();
        script.speed = new Vector3(1, 0, 0); // access and change speed value of main character
    }

    soundList = GetComponents<AudioSource> ();
}
```

# Design Pattern trong Game

**Cách truyền một giá trị từ script này sang script khác.**

**Chú ý:**

- Lệnh **mainCharacter = GameObject.FindGameObjectWithTag("Player");** cần phải đặt ở hàm Start (hàm này chỉ gọi một lần khi khởi tạo xong đối tượng),
- Nếu lệnh này đặt ở ***Update/FixedUpdate/Render...*** thì sẽ làm chậm chương trình và không tối ưu. Vì quá trình tìm kiếm một đối tượng dựa vào tag hay vào một tiêu chí nào đó sẽ mất rất nhiều thời gian.



# Design Pattern trong Game

## Cách truyền một giá trị từ script này sang script khác.

- Sau khi tìm thấy đối tượng chúng ta có thể truy xuất, thay đổi các thành phần public của script đó bằng cách:


```
if (mainCharacter != null)
{
    MainCharacterBehaviour script = mainCharacter.GetComponent<
MainCharacterBehaviour>();
    script.speed = new Vector3(1, 0, 0); // access and change speed va
lue of main character
}
```

- Hoặc bạn có thể cài đặt đối tượng MainCharacter theo Singleton như hướng dẫn ở trên, vì thông thường trong một game thì chỉ có 1 MainCharacter.



# DEMO

Design  
Pattern

A white hand cursor icon, resembling a computer mouse pointer, is pointing upwards at the letter 'O' in the word 'DEMO'.

# Camera

- Camera trong Unity cũng được dùng để hiển thị game trên thế giới cho người chơi. Nó cũng được coi là một Game Object trong Unity.
- Có thể xoay, di chuyển.... tùy chỉnh nó theo ý tưởng.
- Camera được sử dụng để hiển thị cảnh trong game, chúng ta có thể làm cho game của mình trở nên độc đáo hơn nhờ tùy chỉnh Camera.
- Trong một cảnh, chúng ta có thể có một hoặc rất nhiều Camera.

# Camera

## Các thuộc tính Camera:

- ***Clear Flags:*** Xác định các bộ phận mà màn hình sẽ bị xóa. Thuận tiện khi sử dụng nhiều máy ảnh và để vẽ nhiều đối tượng khác nhau. Không xóa nó sẽ hiển thị màu đen xì.
- ***Background:*** Màu nền cho phần màn hình còn lại.

# Camera

## Các thuộc tính Camera:

- **Culling Mask:** Chỉ định các lớp đối tượng của bạn trong Inspector. Cho phép hoặc bỏ qua các đối tượng được hiển thị trong Camera.
- **Projection:**
  - **Perspective:** Camera hiển thị các đối tượng theo phối cảnh tròn vện.
  - **Orthographic:** Hiển thị các đối tượng như một thể thống nhất, không có theo nghĩa của phối cảnh(Perspective).
- **Size:** Kích thước quan sát của Camera khi chọn phép chiếu là Orthographic.
- **Field of view:** Chiều rộng của góc nhìn Camera. Được đo bằng độ dọc theo trục Local Y.

# Camera

## Các thuộc tính Camera:

### ■ ***Clipping plane:***

- ***Near:*** Khoảng cách gần nhất hiển thị trong Camera.
- ***Far:*** Khoảng cách xa nhất hiển thị được trong tầm nhìn của Camera.

### ■ ***Normalized View Port Rect:***

- ***X:*** Bắt đầu từ vị trí ngang mà Camera hiển thị.
  - ***Y:*** Bắt đầu từ vị trí thẳng đứng mà Camera hiển thị.
- ***Depth:*** Vị trí. Camera có một giá trị lớn hơn sẽ được hiển thị lên đầu. Tức là cái nào có giá trị lớn hơn thì nó sẽ được ưu tiên được hiển thị lên màn hình.

# Camera

## Các thuộc tính Camera:

- **Rendering path:** Tùy chọn cho việc xác định phương pháp vẽ những gì sẽ được hiển thị lên Camera.
  - **Use Player Setting:** Sử dụng cài đặt Player.
  - **Vertex Lit:** tất cả các đối tượng được đưa ra bởi máy ảnh sẽ được trả lại như đối tượng Vertex-Lit.
  - **Forward:** tất cả các đối tượng sẽ được trả về như một tài liệu. Giống như tiêu chuẩn Unity 2.x.
  - **Deferred Lighting** (Unity Pro only): Mất phí, không chơi. Tất cả các đối tượng sẽ được rút ra mà không có ánh sáng, sau đó tất cả ánh sáng được trả lại trong hàng đợi Render.
  - **Target Texture:** Phiên bản free 4.0 đã có. Tham chiếu đến một texture. Cho phép dựng hình High Dynamic Range cho máy ảnh này





# Kết luận

- Cấu trúc Project
- A. Khởi tạo và cấu hình dự án Game 2D
- B. Tạo các đối tượng cơ bản
  - 1. *Game Object*
  - 2. *Sprite*
  - 3. *Animation và điều khiển hành động nhân vật*
  - 4. *Prefab*
  - 5. *Script và điều khiển máy trạng thái*
  - 6. *Thành phần vật lý và xử lý va chạm*
  - 7. *Thiết kế UI*
  - **8. *Sử dụng Particle System***
  - **9. *Chuyển đổi màn chơi***
  - **10. *Sound***
  - **11. *Design Pattern trong Game***

# Chuẩn bị bài sau

- Cấu trúc Project
- A. Khởi tạo và cấu hình dự án Game 2D
- B. Tạo các đối tượng cơ bản
  - 1. *Game Object*
  - 2. *Sprite*
  - 3. *Animation và điều khiển hành động nhân vật*
  - 4. *Prefab*
  - 5. *Script và điều khiển máy trạng thái*
  - 6. *Thành phần vật lý và xử lý va chạm*
  - 7. *Thiết kế UI*
  - **8. *Sử dụng Particle System***
  - **9. *Chuyển đổi màn chơi***
  - **10. *Sound***
  - **11. *Design Pattern trong Game***



**FPT POLYTECHNIC**

THANK YOU!

[www.poly.edu.vn](http://www.poly.edu.vn)