part
2

rev3

www.gamedevpbru.com

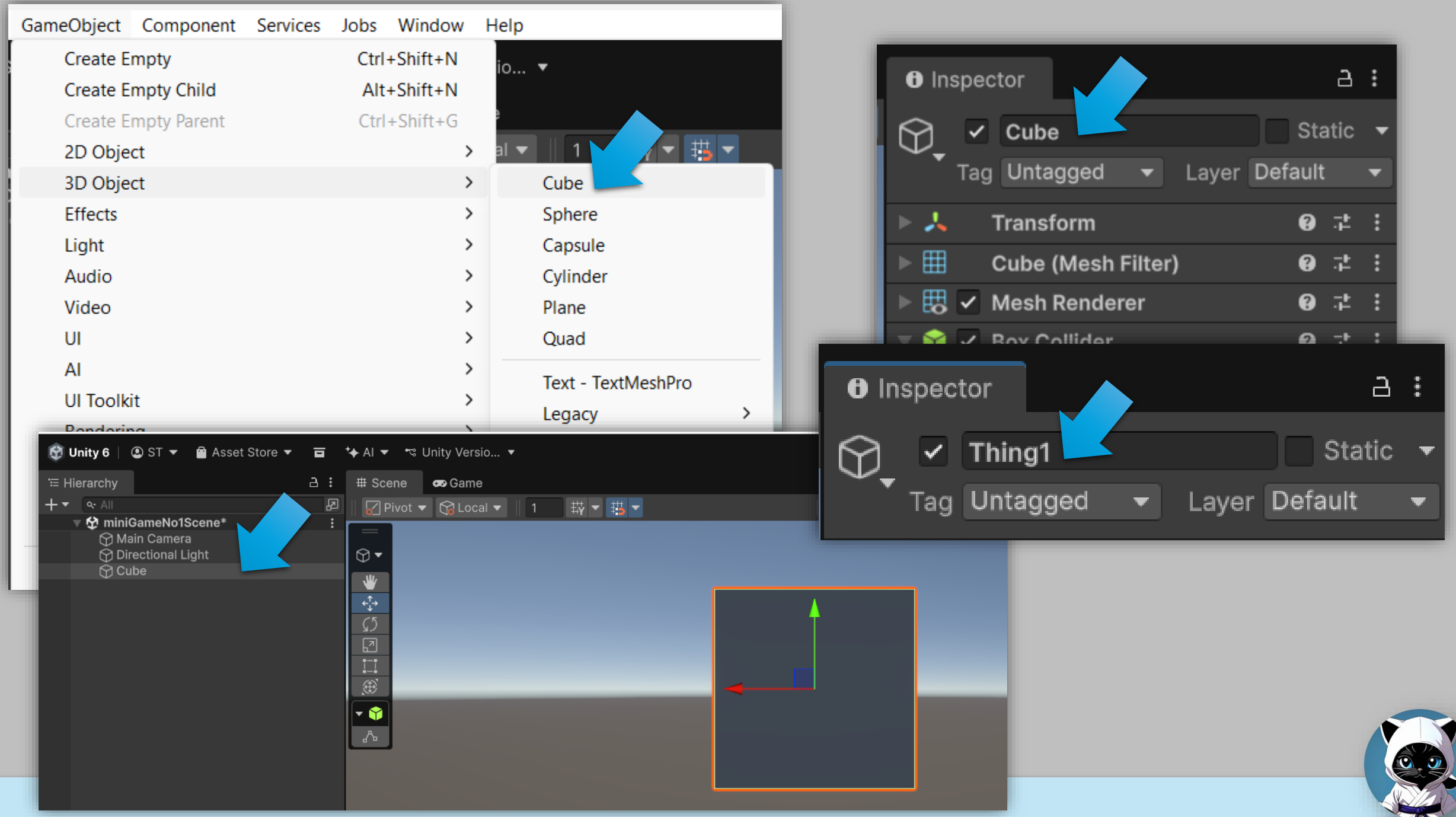
miniGameNo.1

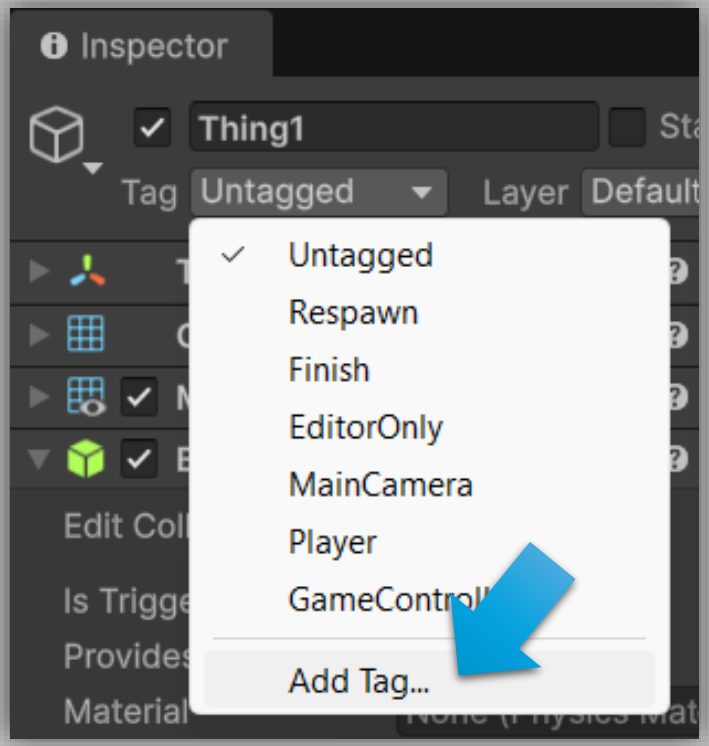
ดนัย เจษฎาฐิติกุล
จารุต บุศราทิจ
DcG.IT@PBRU

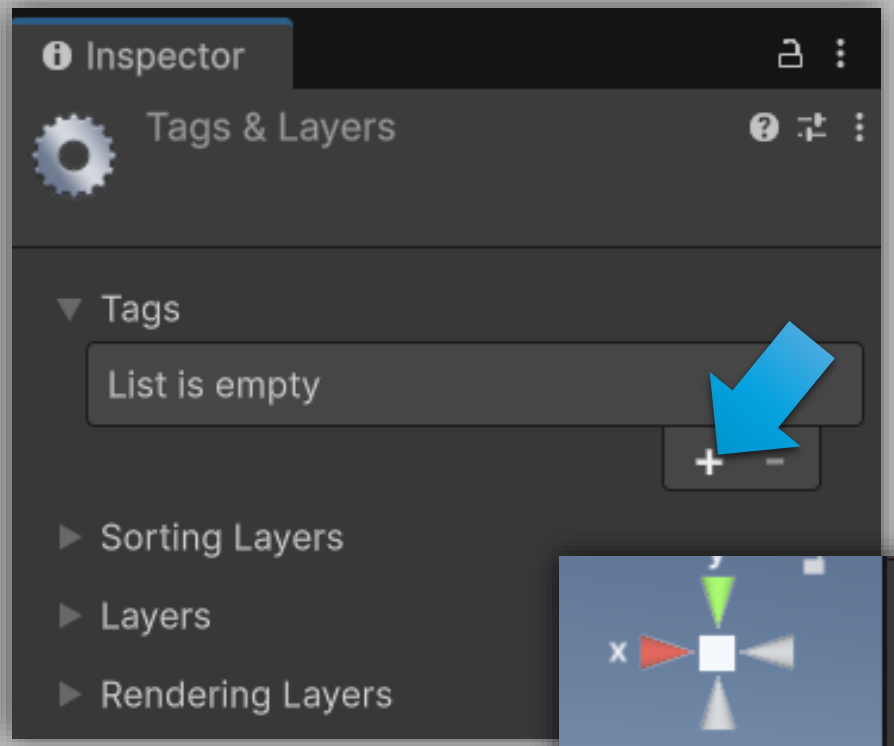
สร้าง Prefab ศัตรู

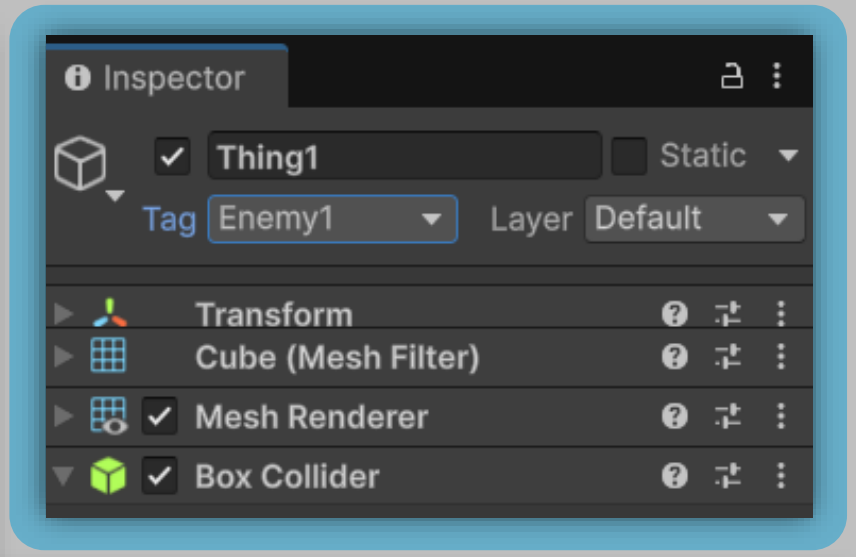
วัตถุ1

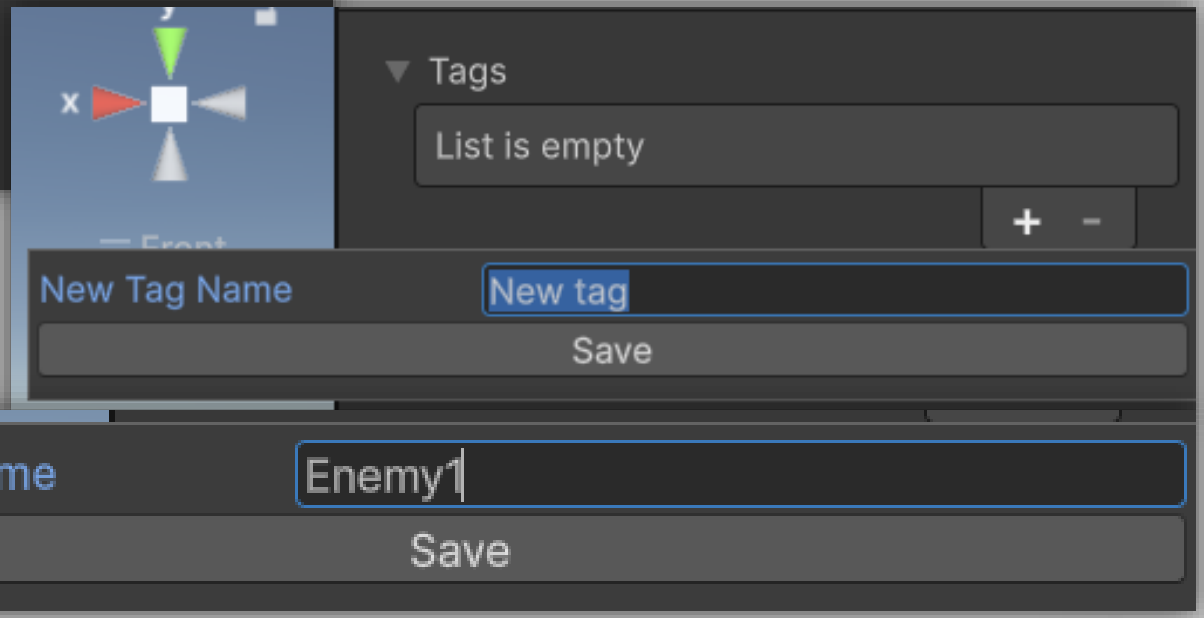
1. สร้าง Cube
2. ตั้งชื่อเป็น Thing1
3. เพิ่ม Box Collider (ถ้ายังไม่มี) และ Rigidbody
4. เปลี่ยน Tag เป็น Enemy1
5. ลากมาวางใน Asset เพื่อแปลงเป็น Prefab
6. ลบ Thing1 ออกจาก Scene

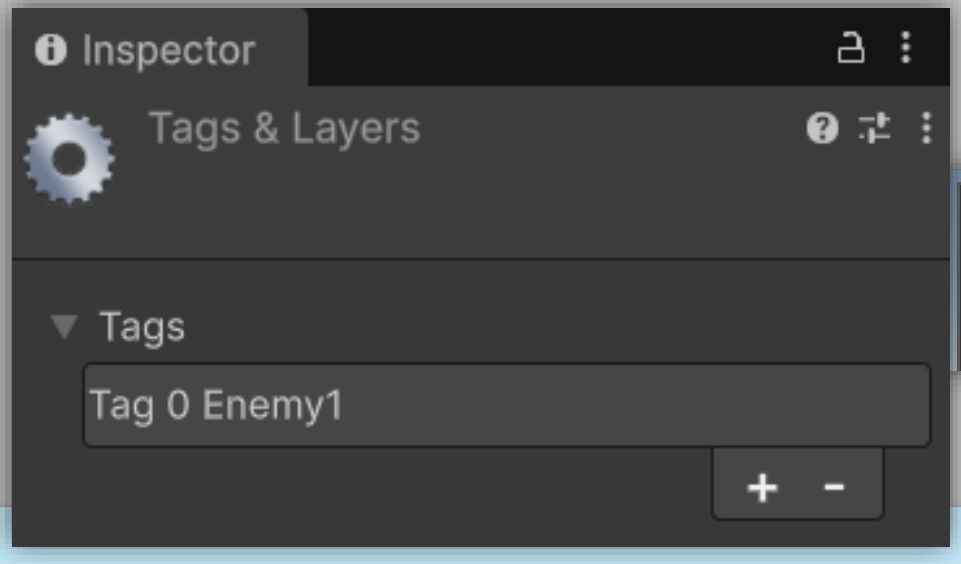

GameObject
Component
Services
Jobs
Window
Help
Create Empty
Ctrl+Shift+N
Create Empty Child
Alt+Shift+N
Create Empty Parent
Ctrl+Shift+G
2D Object
3D Object
Effects
Light
Audio
Video
UI
AI
UI Toolkit
Cube
Sphere
Capsule
Cylinder
Plane
Quad
Text - TextMeshPro
Legacy
Inspector
Cube
Static
Tag
Untagged
Layer
Default
Transform
Cube (Mesh Filter)
Mesh Renderer
Box Collider
Inspector
Thing1
Static
Tag
Untagged
Layer
Default
Unity 6
Asset Store
Unity Versio...
Hierarchy
Scene
Game
Pivot
Local
miniGameNo1Scene*
Main Camera
Directional Light
Cube

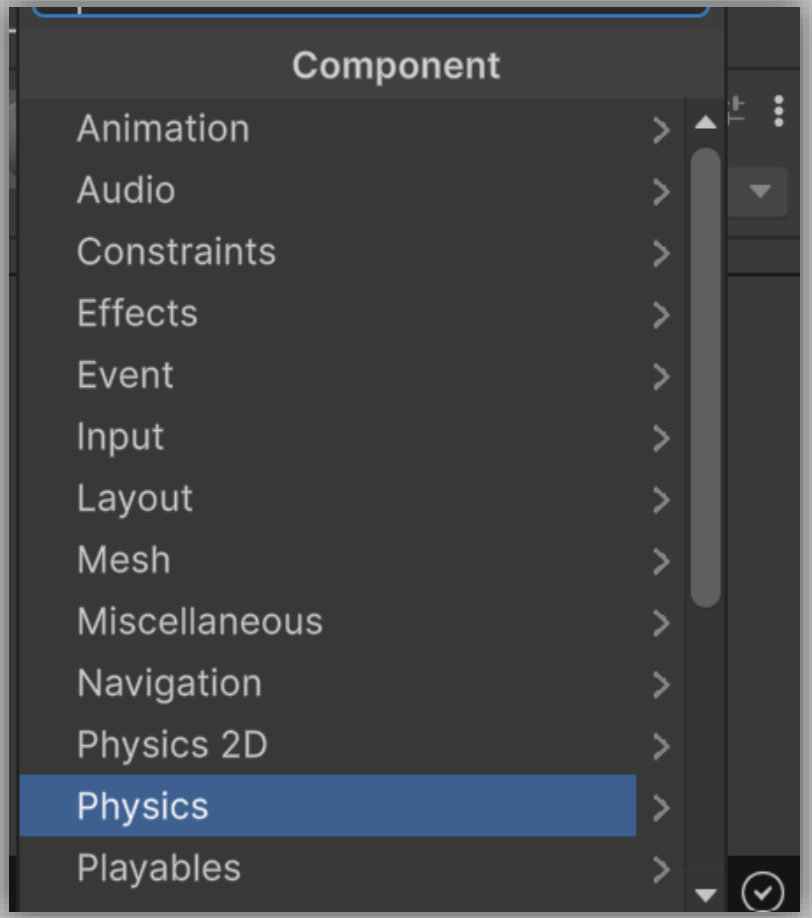

Inspector
Thing1
Tag Untagged
Layer Default
Untagged
Respawn
Finish
EditorOnly
MainCamera
Player
GameControll
Add Tag...

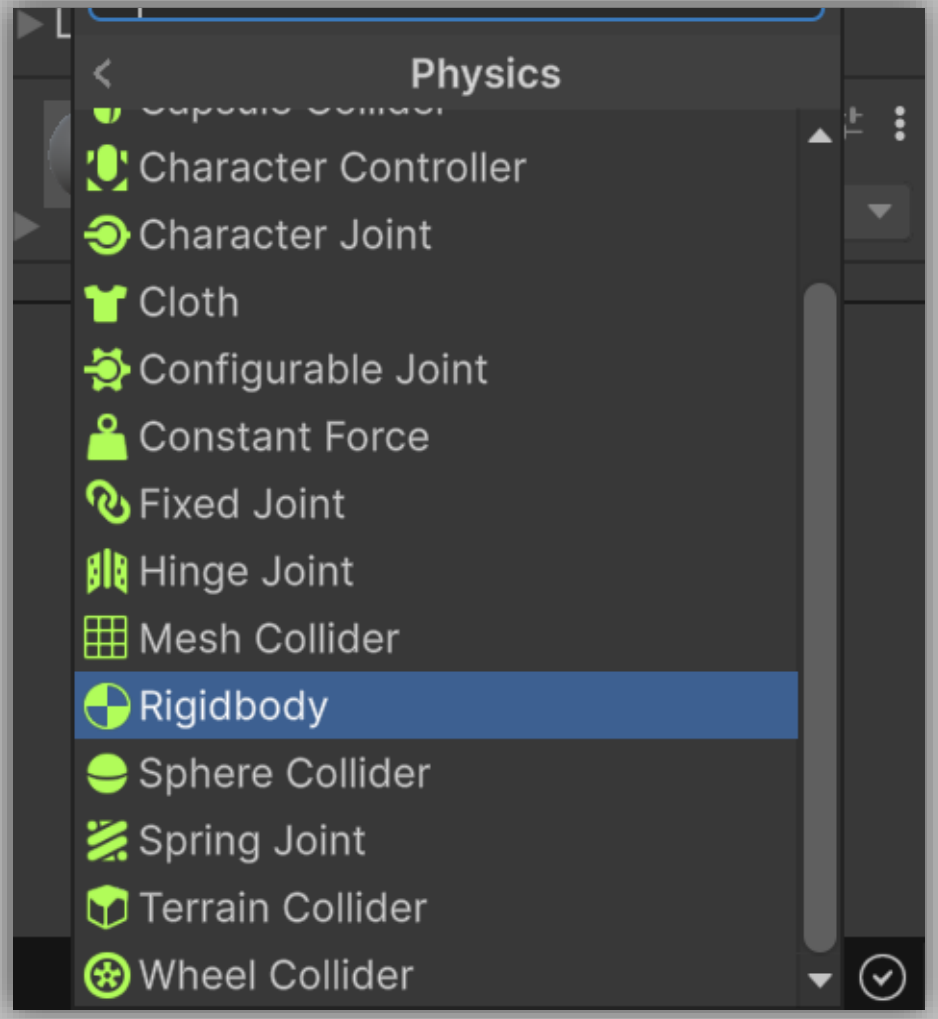

Inspector
Tags & Layers
Tags
List is empty
Sorting Layers
Layers
Rendering Layers


Inspector
Thing1
Static
Tag Enemy1
Layer Default
Transform
Cube (Mesh Filter)
Mesh Renderer
Box Collider


Tags
List is empty
New Tag Name
New tag
Save
New Tag Name
Enemy1
Save

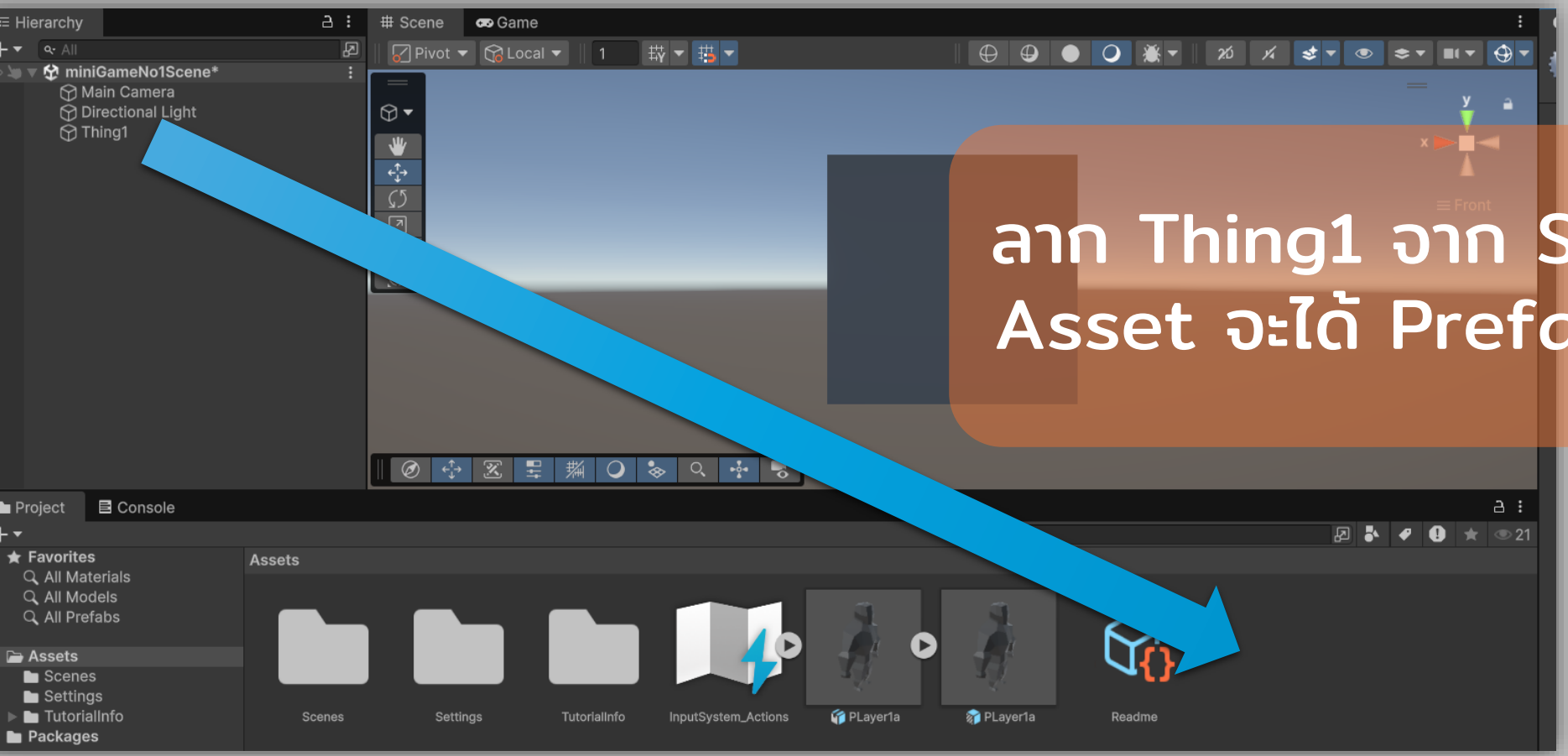

Inspector
Tags & Layers
Tags
Tag 0 Enemy1

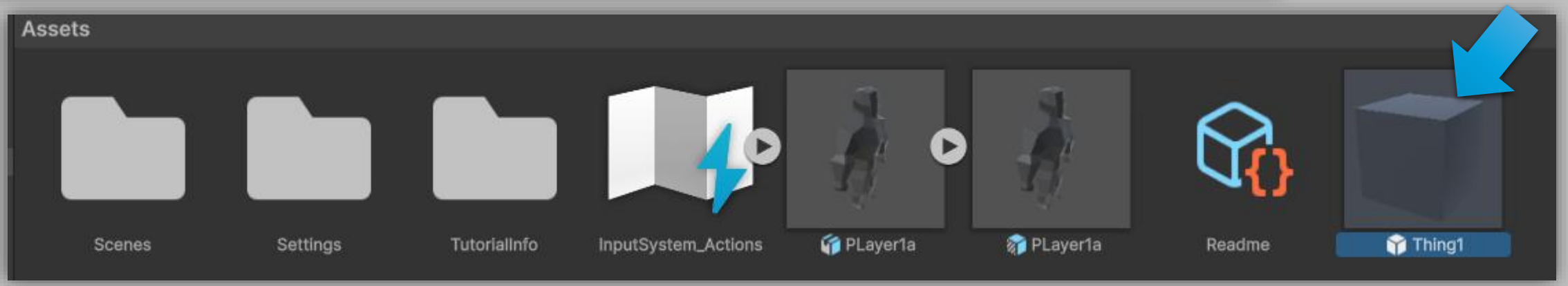




เพิ่ม Conponent ชื่อ Rigidbody จาก Physics

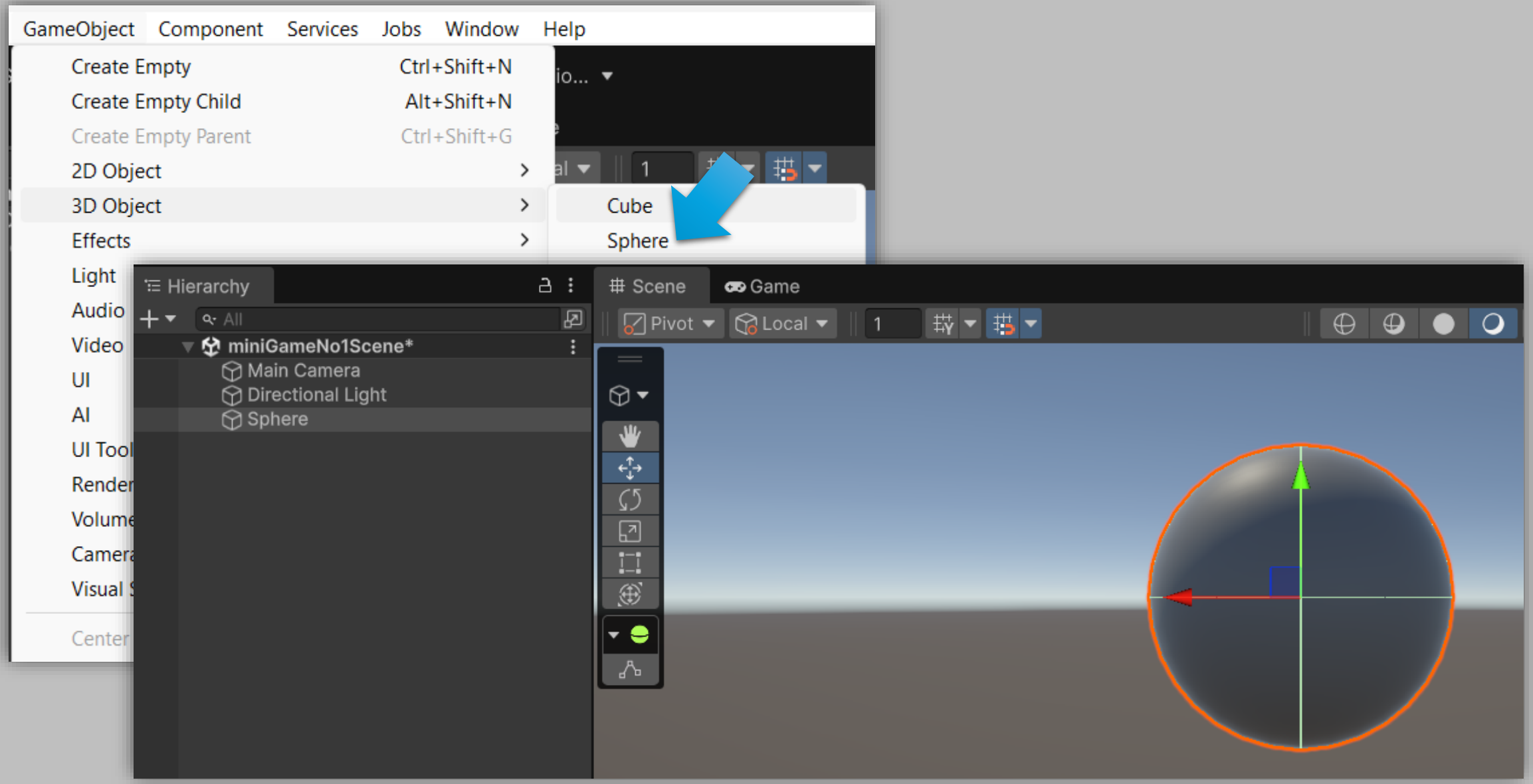
ลาก Thing1 จาก Scene มาวางใน Asset จะได้ Prefab ของ Thing1

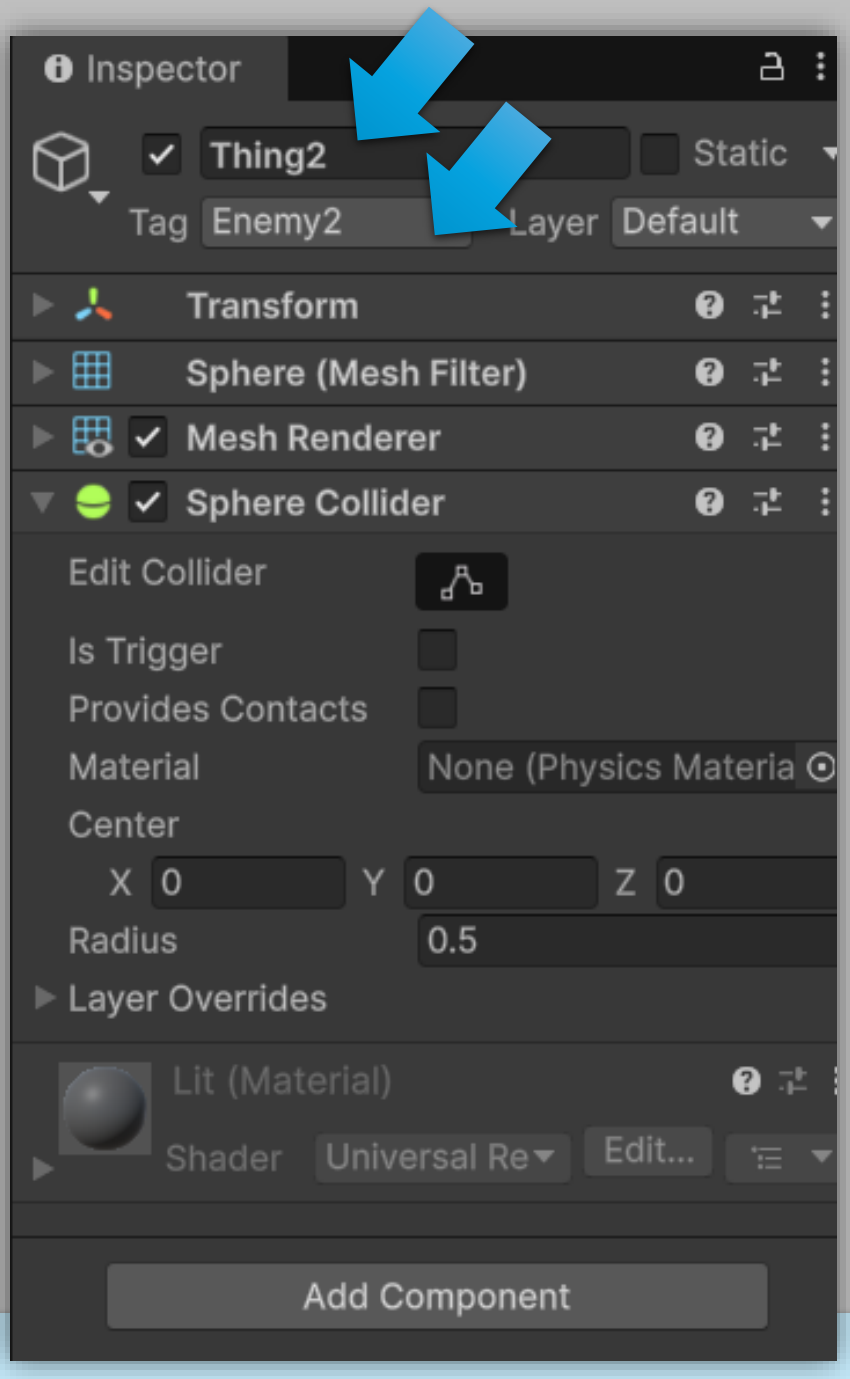




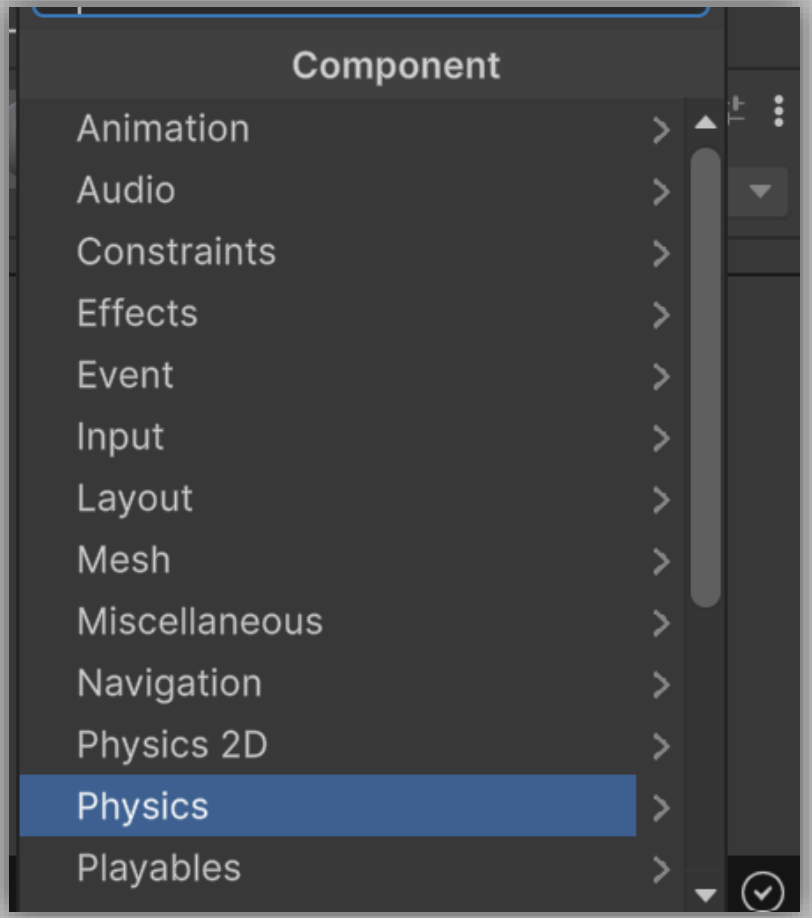
วัตถุ2

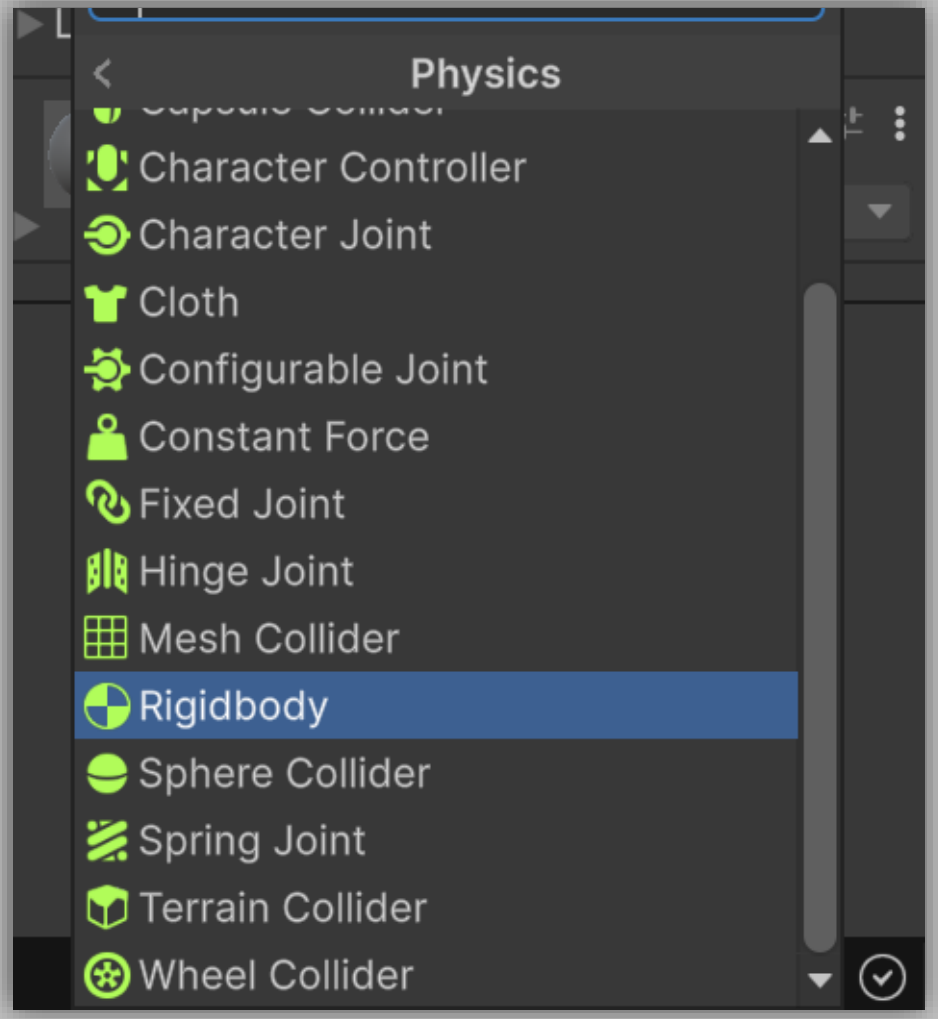
1. สร้าง Sphere
2. ตั้งชื่อเป็น Thing2
3. เพิ่ม Sphere Collider (ถ้ายังไม่มี) และ Rigidbody
4. เปลี่ยน Tag เป็น Enemy2
5. ลากมาวางใน Asset เพื่อแปลงเป็น Prefab
6. ลบ Thing2 ออกจาก Scene


GameObject
Component
Services
Jobs
Window
Help
Create Empty Ctrl+Shift+N
Create Empty Child Alt+Shift+N
Create Empty Parent Ctrl+Shift+G
2D Object
3D Object
Effects
Light
Audio
Video
UI
AI
Cube
Sphere
Hierarchy
All
miniGameNo1Scene*
Main Camera
Directional Light
Sphere
Scene
Game
Pivot
Local
1

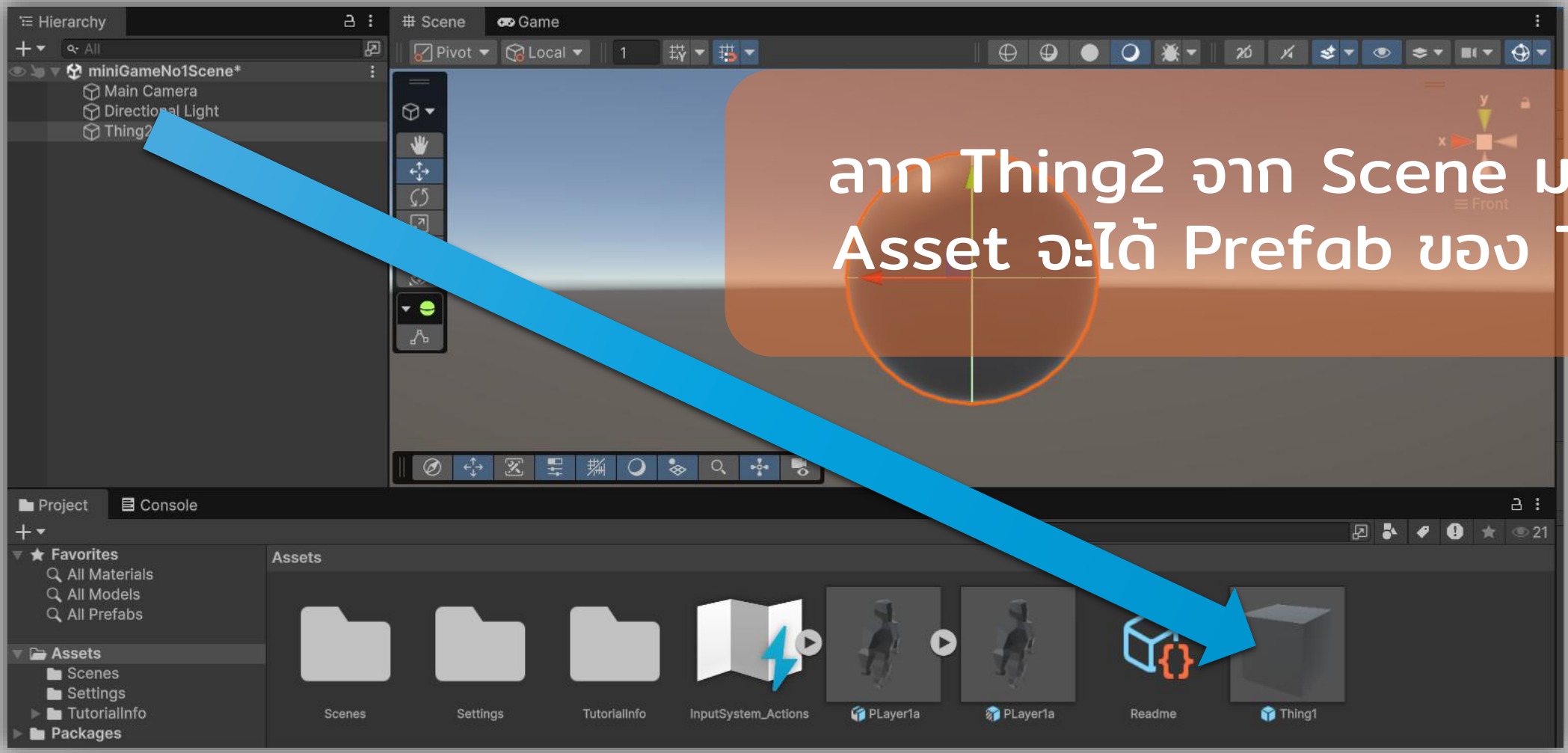


เปลี่ยนชื่อเป็นThing2 และเพิ่ม Tag ชื่อ Enemy2 แล้วกำหนดให้ Thing2 เป็น Enemy2

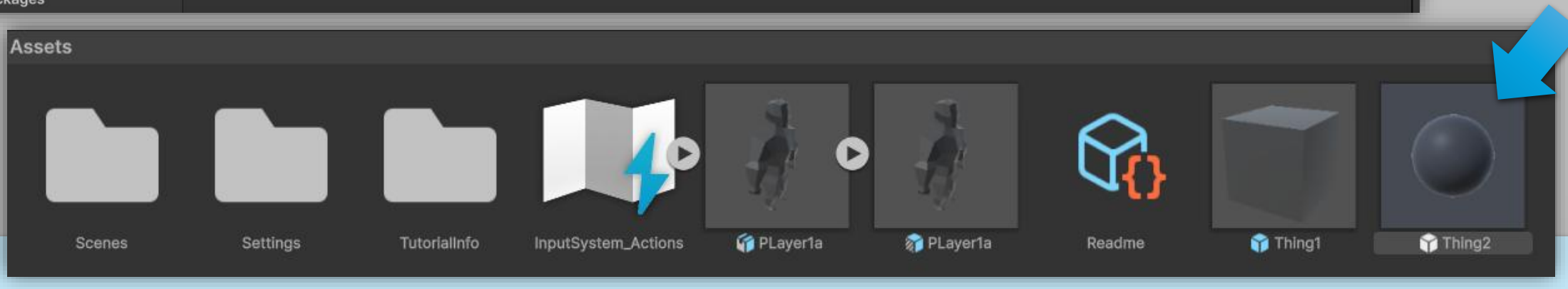




เพิ่ม Conponent ชื่อ Rigidbody จาก Physics



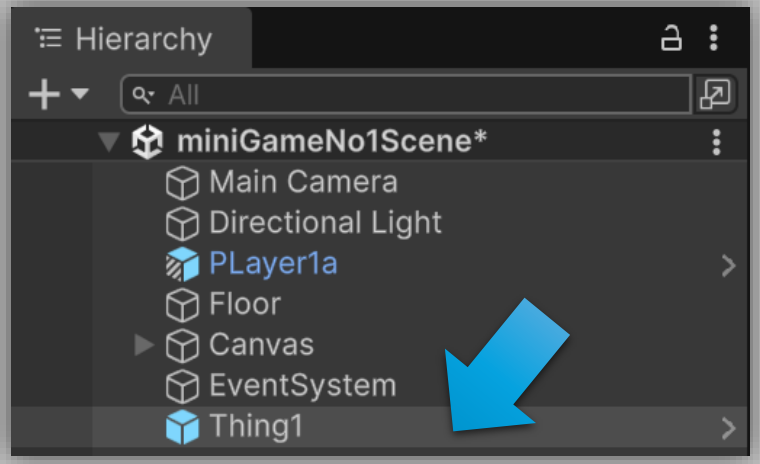
ลาก Thing2 จาก Scene มาวางใน Asset จะได้ Prefab ของ Thing2

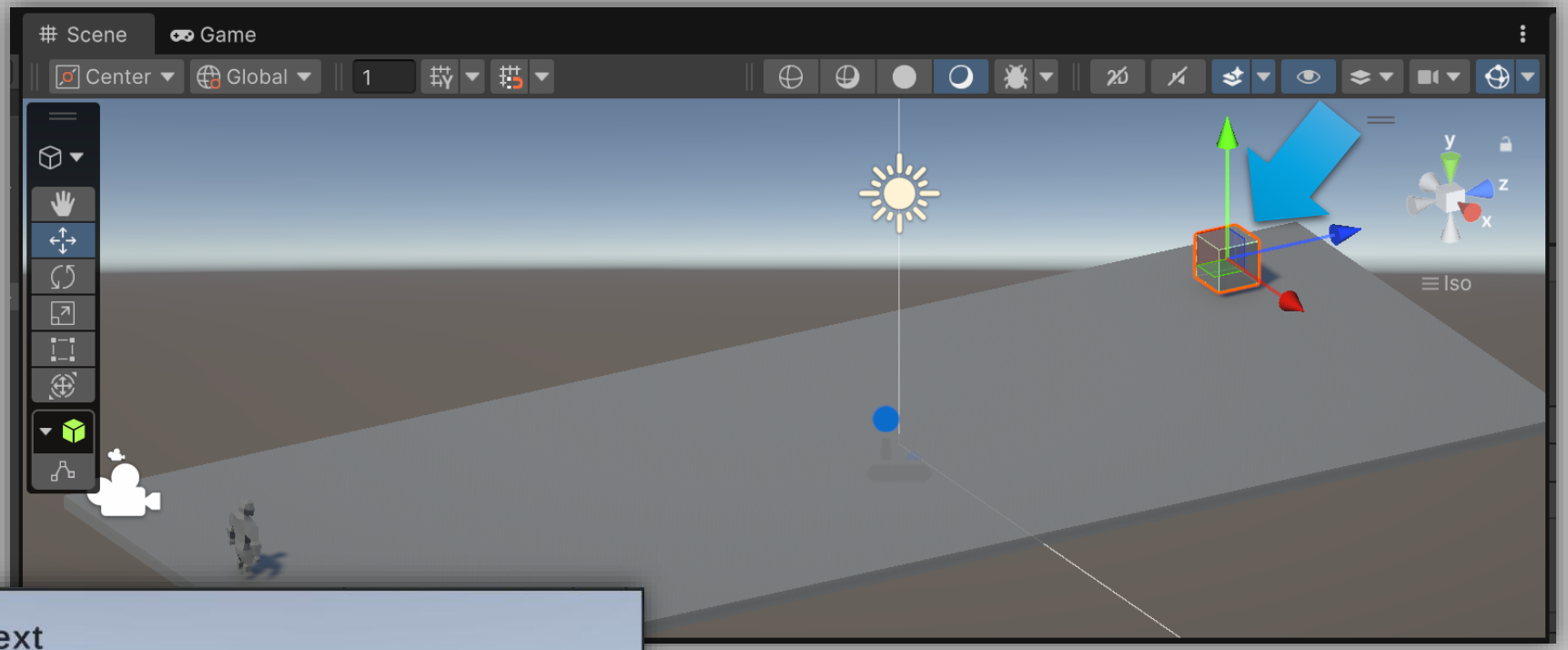


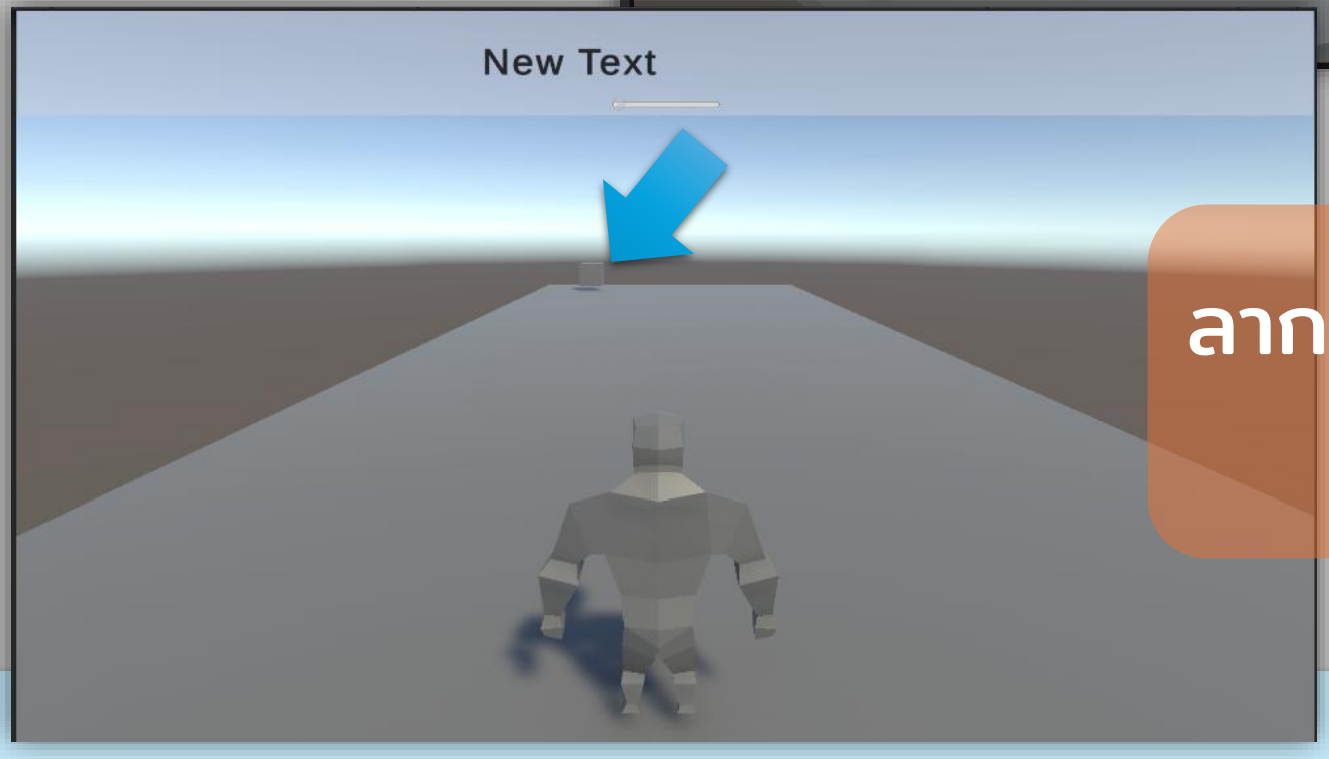
พฤติกรรมของศัตรู

วัตถุ1 (Enemy1)

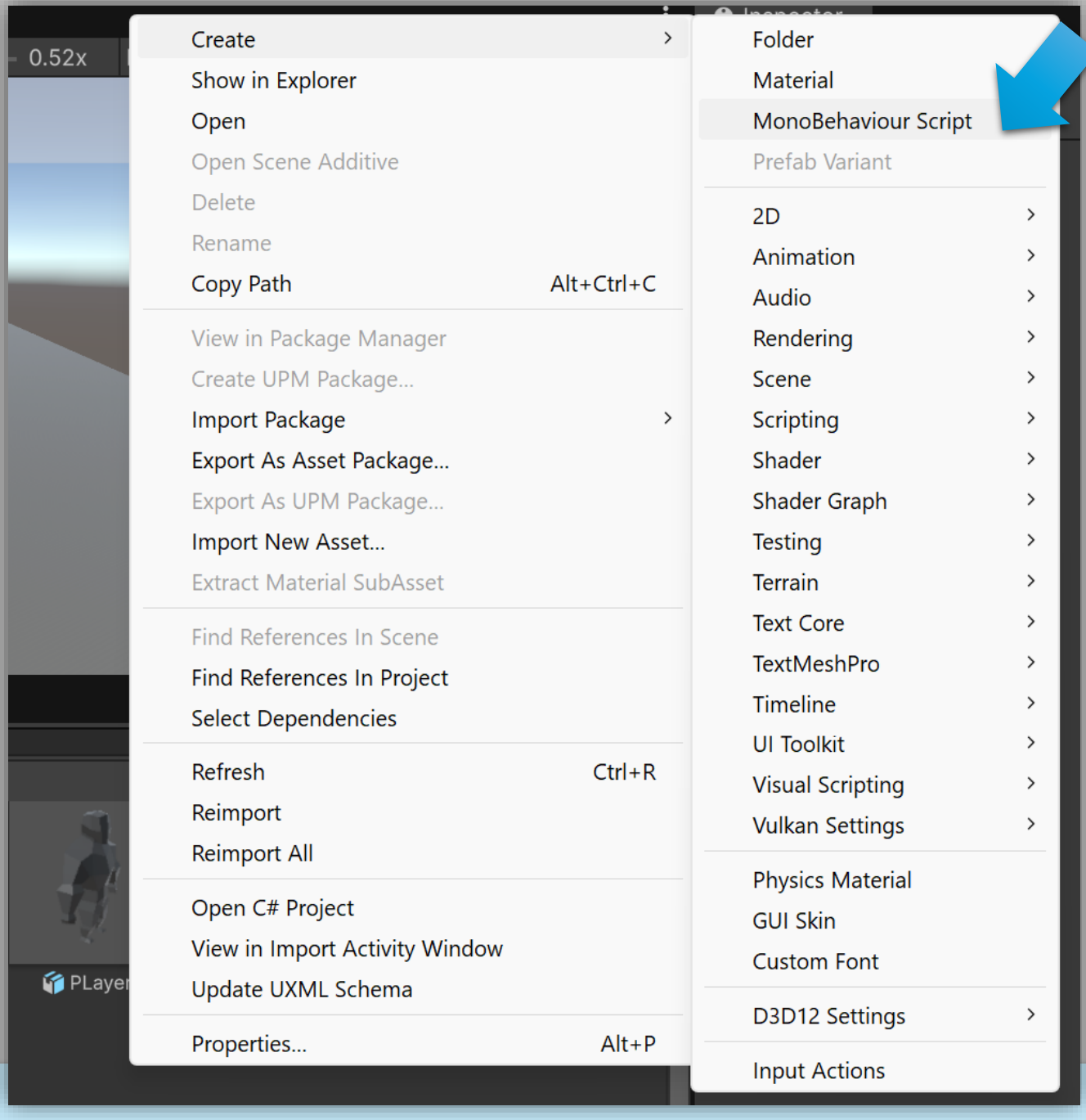
1. เลื่อนจากจุดเกิดมาหาผู้เล่น
2. ถ้าชนผู้เล่นจะเพิ่มคะแนน แล้วหายไป
3. ถ้าหลุดหน้าจอ (เทียบค่า z ของวัตถุ 1 กับค่า z ของ Player)
4. วัตถุมีคุณสมบัติ
    1. speed = 2 ถึง 10 ขึ้นอยู่กับ wave การเกิด
    2. damage = 1







ลาก Thing1 มาวางอีกด้านหนึ่งของฉาก



สร้างสคริปต์ Enemy1

Enemy2.cs* | **Enemy1.cs** | UIManager.cs | PlayerActivity.cs | PlayerProperty.cs

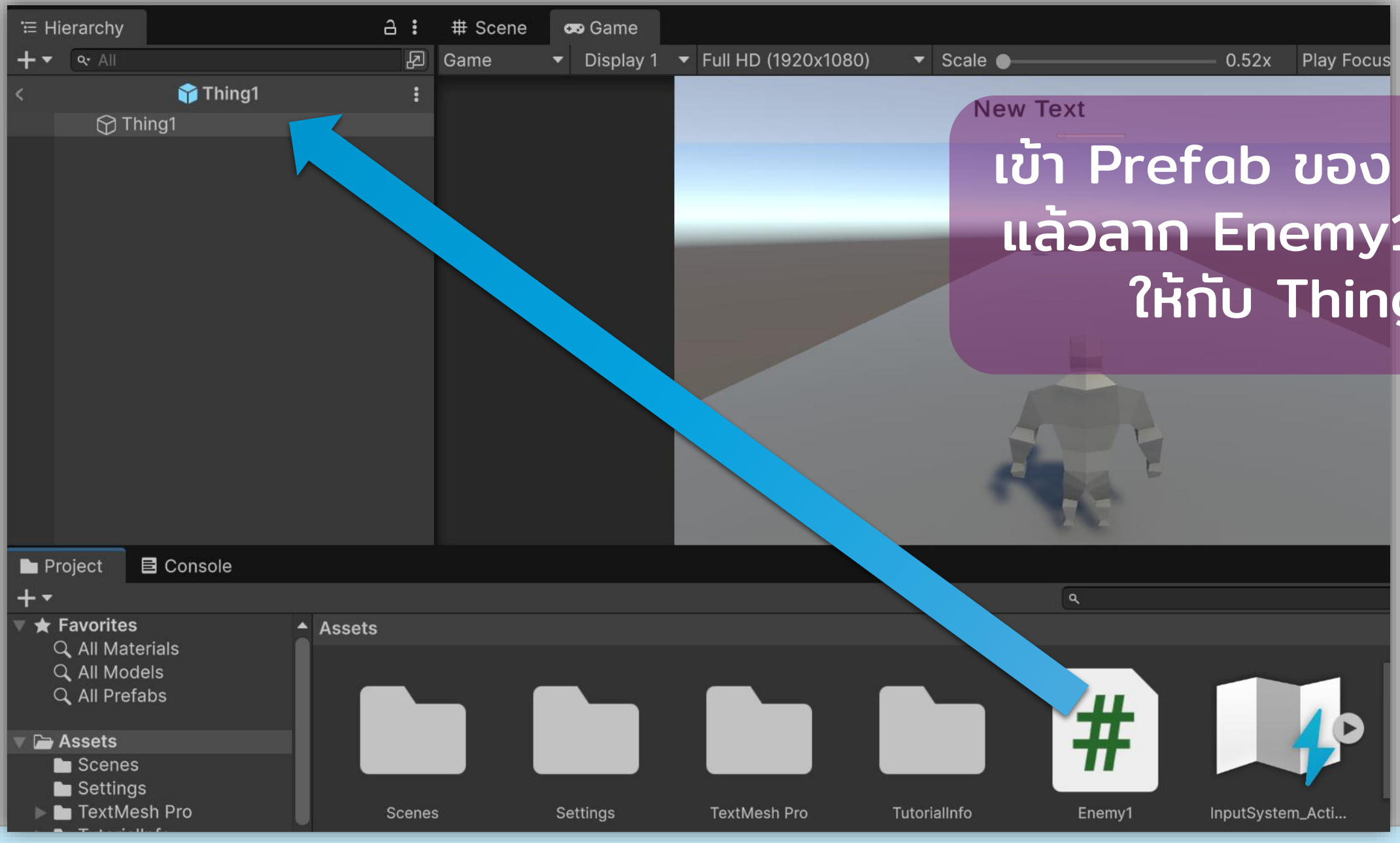
Assembly-CSharp | Enemy1 | FixedUpdate()

```
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class Enemy1 : MonoBehaviour
{
    public float speed = 2f; // ความเร็วของศัตรู
    public int damage = 1; // คะแนนที่จะเพิ่มเมื่อชน

    private Transform playerTransform; // เก็บ Reference ของผู้เล่น
```

```
 9      void Start()
10      {
11          GameObject player = GameObject.FindGameObjectWithTag("Player");
12          if (player != null)
13          {
14              playerTransform = player.transform;
15          }
16          else
17          {
18              Destroy(gameObject); // ถ้าไม่เจอผู้เล่น ให้ลบศัตรูออกไปเลย
19          }
20      }
21
```

```
22      // ใช้ FixedUpdate เมื่อทำงานกับ Rigidbody
        Unity Message | 0 references
23      void FixedUpdate()
24      {
25          if (playerTransform == null)
26          {
27              return;
28          }
29          // เคลื่อนที่ศัตรูเข้าหาผู้เล่น
30          // ใช้ Vector3.forward แทน Vector3.left ถ้าศัตรูเกิดอยู่ข้างหน้า และผู้เล่นเดินไปข้างหน้า
31          // สำหรับตัวอย่างนี้ สมมติว่าศัตรูเกิดข้างหลังผู้เล่นและเคลื่อนที่ไปข้างหน้า (แกน Z เพิ่มขึ้น)
32          transform.Translate(-1.0f*Vector3.forward * speed * Time.deltaTime);
33
34          // ตรวจสอบตำแหน่ง Z ของศัตรูเทียบกับผู้เล่น
35          // ถ้าศัตรูอยู่หลังผู้เล่นแล้ว (Z น้อยกว่าหรือเท่ากับของผู้เล่น) ให้ทำลายตัวเอง
36          if (transform.position.z <= playerTransform.position.z)
37          {
38              Destroy(gameObject);
39          }
40      }
41
```

```
42      // ฟังก์ชันตรวจสอบการชนแบบ Physical Collision
            Unity Message | 0 references
43      void OnCollisionEnter(Collision collision)
44      {
45          // ตรวจสอบว่า Object ที่เราชนมี Tag ว่า Player หรือไม่
46          if (collision.gameObject.CompareTag("Player"))
47          {
48              PlayerActivity acts = collision.gameObject.GetComponent<PlayerActivity>();
49              acts.IncScore();
50
51              // ทำลายศัตรูเมื่อชน
52              Destroy(gameObject);
53          }
54      }
55  }
```

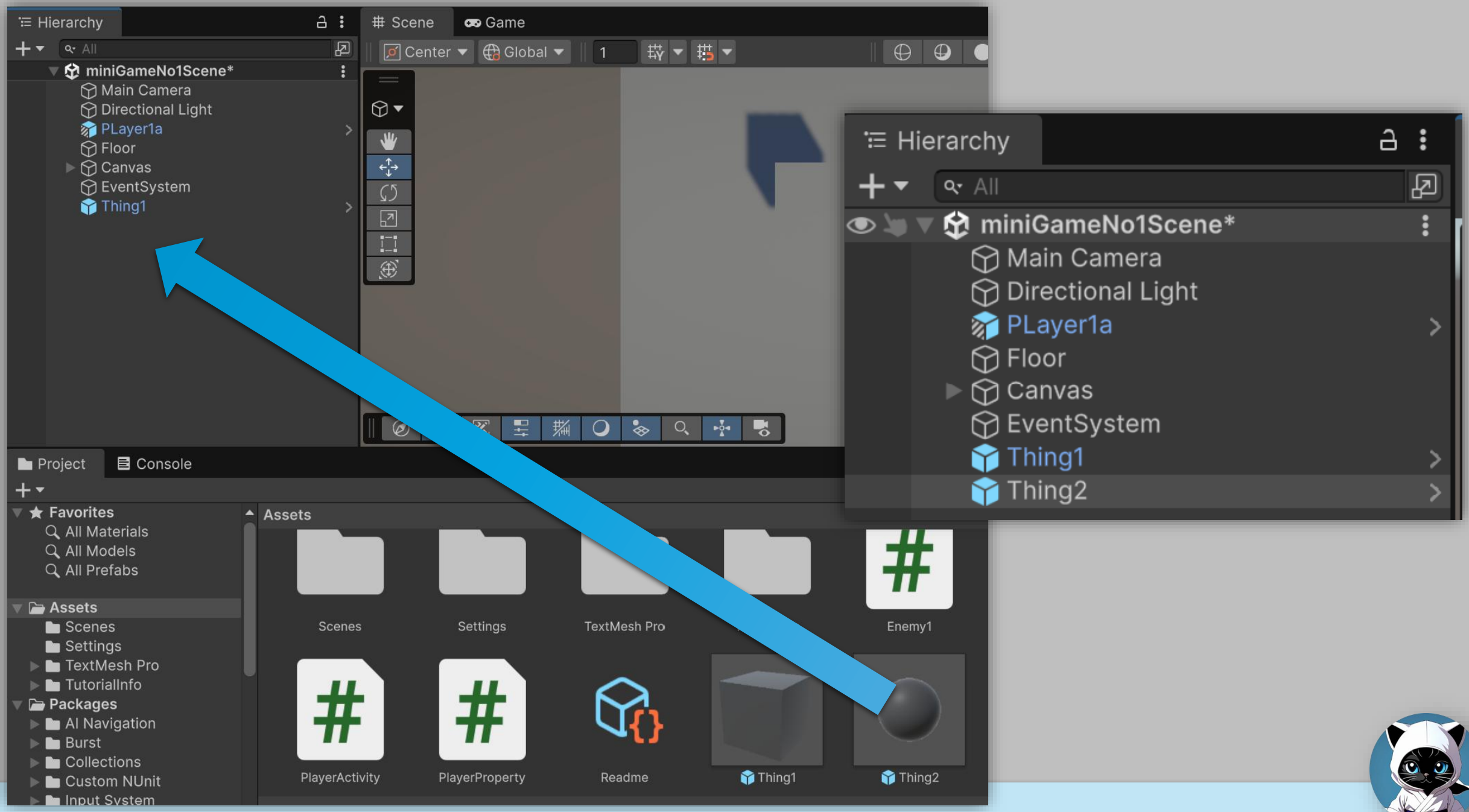

Hierarchy
Scene
Game
All
Thing1
Thing1
Game
Display 1
Full HD (1920x1080)
Scale
0.52x
Play Focus
New Text
เข้า Prefab ของ Thing1
แล้วลาก Enemy1 ไปวาง
ให้กับ Thing1
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Scenes
Settings
TextMesh Pro
Assets
Scenes
Settings
TextMesh Pro
TutorialInfo
Enemy1
InputSystem_Acti...

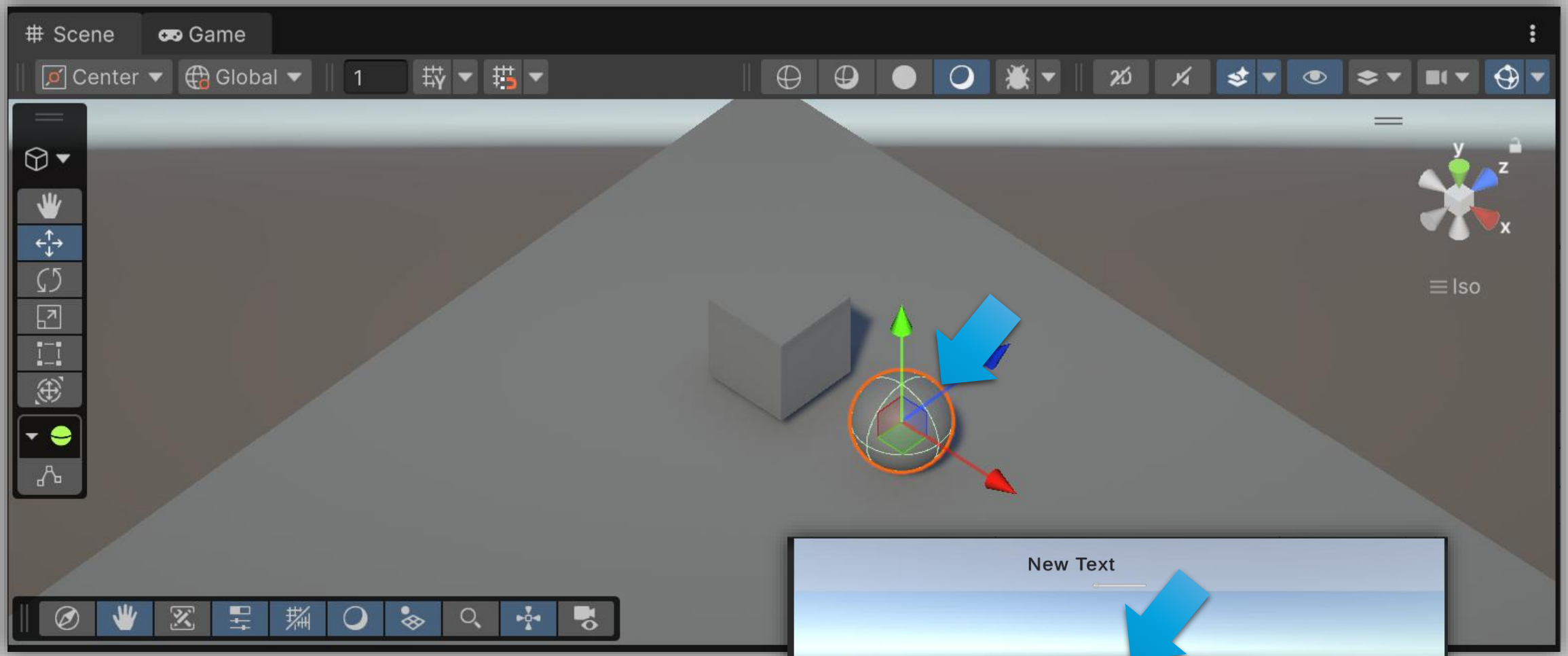

Inspector
Thing1
Static
Tag
Enemy1
Layer
Default
Transform
Position
X 0.2476
Y 3.9220
Z -10
Rotation
X 0
Y 0
Z 0
Scale
X 0.5
Y 0.5
Z 0.5
Cube (Mesh Filter)
Mesh Renderer
Box Collider
Enemy 1 (Script)
Script
Enemy1
Speed
2
Damage
1
Lit (Material)
Shader
Universal
Edit...
Add Component

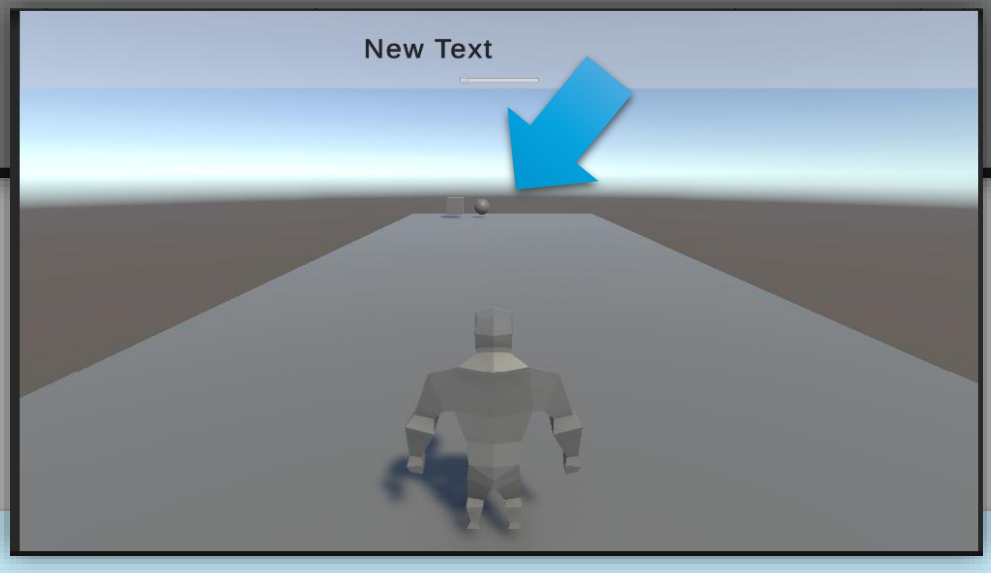
ลองเล่นจะพบว่า Enemy1 จะหายไปเมื่อชนกับผู้เล่น (จะได้คะแนนเพิ่ม) กับไม่ชนกับผู้เล่นแต่เลยตำแหน่งของผู้เล่น

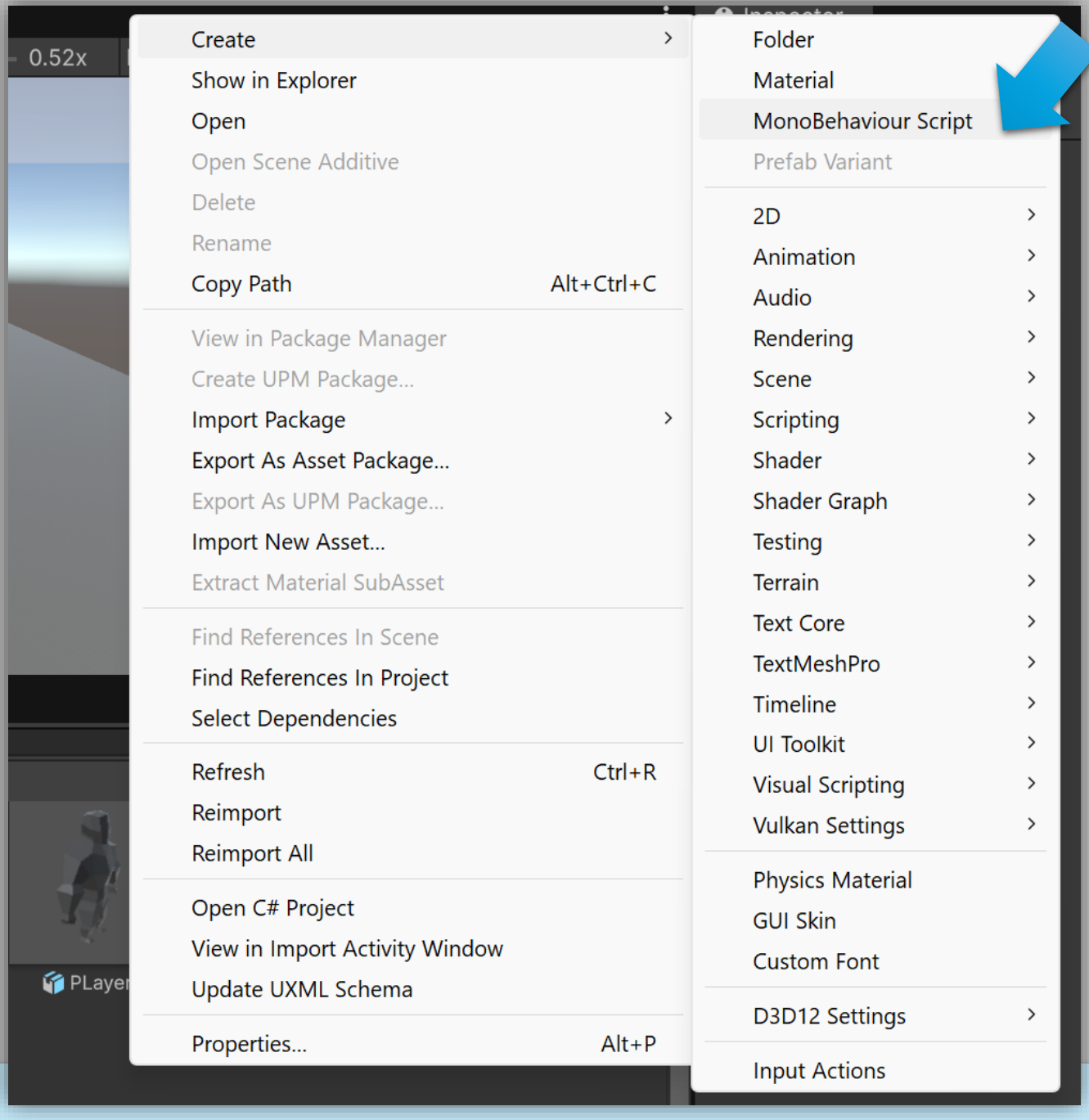
วัตถุ2 (Enemy2)

1. เลื่อนจากจุดเกิดมาหาผู้เล่น
2. ถ้าชนผู้เล่นจะลดคะแนน แล้วหายไป
3. ถ้าหลุดหน้าจอ (เทียบค่า z ของวัตถุ 1 กับค่า z ของ Player)
4. วัตถุมีคุณสมบัติ
    1. Speed = 2 ถึง 10 ขึ้นอยู่กับ wave การเกิด
    2. Score = -1


Hierarchy
Scene
Game
Center
Global
miniGameNo1Scene*
Main Camera
Directional Light
PLayer1a
Floor
Canvas
EventSystem
Thing1
Thing2
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Scenes
Settings
TextMesh Pro
TutorialInfo
Packages
AI Navigation
Burst
Collections
Custom NUnit
Input System
Enemy1
PlayerActivity
PlayerProperty
Readme


Scene
Game
Center
Global
1
y
z
x
Iso


New Text



สร้างสคริปต์ Enemy2

Enemy2.cs | Enemy1.cs | UIManager.cs | PlayerActivity.cs | PlayerProperty.cs

Assembly-CSharp | Enemy2 | OnCollisionEnter(Collision collision)

```
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class Enemy2 : MonoBehaviour
{
    public float speed = 2f; // ความเร็วของศัตรู
    public int damage = -1; // คะแนนที่จะลดเมื่อชน

    private Transform playerTransform; // เก็บ Reference ของผู้เล่น
```

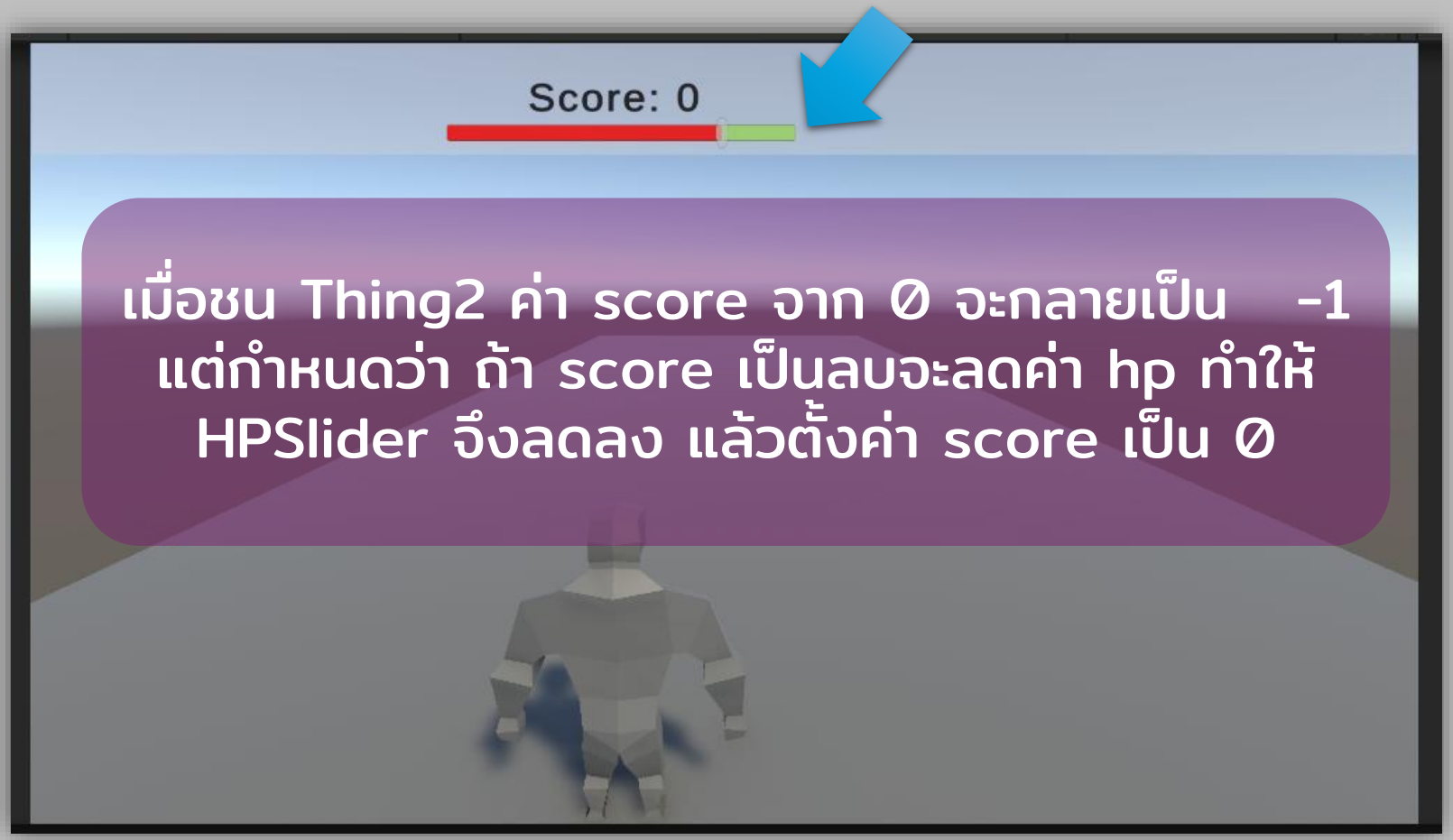
```
 9      void Start()
10      {
11          GameObject player = GameObject.FindGameObjectWithTag("Player");
12          if (player != null)
13          {
14              playerTransform = player.transform;
15          }
16          else
17          {
18              Destroy(gameObject); // ถ้าไม่เจอผู้เล่น ให้ลบศัตรูออกไปเลย
19          }
20      }
21
```

```
22      // ใช้ FixedUpdate เมื่อทำงานกับ Rigidbody
        Unity Message | 0 references
23      void FixedUpdate()
24      {
25          if (playerTransform == null)
26          {
27              return;
28          }
29          // เคลื่อนที่ศัตรูเข้าหาผู้เล่น
30          // ใช้ Vector3.forward แทน Vector3.left ถ้าศัตรูเกิดอยู่ข้างหน้า และผู้เล่นเดินไปข้างหน้า
31          // สำหรับตัวอย่างนี้ สมมติว่าศัตรูเกิดข้างหลังผู้เล่นและเคลื่อนที่ไปข้างหน้า (แกน Z เพิ่มขึ้น)
32          transform.Translate(-1.0f * Vector3.forward * speed * Time.deltaTime);
33
34          // ตรวจสอบตำแหน่ง Z ของศัตรูเทียบกับผู้เล่น
35          // ถ้าศัตรูอยู่หลังผู้เล่นแล้ว (Z น้อยกว่าหรือเท่ากับของผู้เล่น) ให้ทำลายตัวเอง
36          if (transform.position.z <= playerTransform.position.z)
37          {
38              Destroy(gameObject);
39          }
40      }
41
```

```
42      // ฟังก์ชันตรวจสอบการชนแบบ Physical Collision
            Unity Message | 0 references
43      void OnCollisionEnter(Collision collision)
44      {
45          // ตรวจสอบว่า Object ที่เราชนมี Tag ว่า Player หรือไม่
46          if (collision.gameObject.CompareTag("Player"))
47          {
48              PlayerActivity acts = collision.gameObject.GetComponent<PlayerActivity>();
49              acts.DecScore();
50
51              // ทำลายศัตรูเมื่อชน
52              Destroy(gameObject);
53          }
54      }
55  }
```

เข้า Prefab ของ Thing2
แล้วลาก Enemy2 ไปวาง
ให้กับ Thing2


Inspector
Thing2
Static
Tag Enemy2
Layer Default
Transform
Position X 0.2476 Y 3.9220 Z -10
Rotation X 0 Y 0 Z 0
Scale X 0.5 Y 0.5 Z 0.5
Sphere (Mesh Filter)
Mesh Renderer
Sphere Collider
Rigidbody
Enemy 2 (Script)
Script Enemy2
Speed 2
Damage -1
Lit (Material)
Shader Universal
Edit...


Score: 0
เมื่อชน Thing2 ค่า score จาก 0 จะกลายเป็น -1
แต่กำหนดว่า ถ้า score เป็นลบจะลดค่า hp ทำให้
HPSlider จึงลดลง แล้วตั้งค่า score เป็น 0

อธิบายการตรวจสอบการชน

ระบบ Collision แบบปกติ

- วิธีนี้จะทำให้เกิด "แรงกระแทก" หรือการ "ฟิสิกส์ชนกัน" เช่น ศัตรูจะกระเด้งออกหรือหยุดเดินเมื่อชนผู้เล่น เป็นต้น
- เพื่อให้การชนแบบฟิสิกส์ทำงานได้ถูกต้อง ต้องตั้งค่าดังนี้
  - ตัวศัตรู (Enemy Prefab)
    - Box Collider หรือแบบอื่น ๆ ที่ต้องปิดเครื่องหมาย (Uncheck) ที่ช่อง Is Trigger
    - ต้องมี Component Rigidbody นี้ และ ห้ามติ๊ก Is Kinematic (เพราะถ้าเป็น Kinematic มันจะไม่คำนวณการชนแบบฟิสิกส์ปกติ)
  - ตัวผู้เล่น (Player):
    - ต้องมี Collider เช่น Box หรือ Capsule ที่ไม่เป็น Trigger
    - ต้องมี Rigidbody เช่นกัน

สิ่งที่ต้องระวัง

- ใช้ FixedUpdate() แทน Update() เมื่อทำงานกับ Rigidbody เนื่องจาก FiexedUpdate() จะทำการคำนวณด้านฟิสิกส์ที่ละเอียด
- แต่ ถ้าไม่ต้องการผลของฟิสิกส์ที่มีความละเอียดสามารถใช้ Update() ได้ตามปกติ

ภาพโดย ChatGPT

ฟังก์ชันตรวจสอบการชน

OnCollisionEnter()

ถูกเรียก “ครั้งแรก” ที่วัตถุเริ่มชนกันโดยลักษณะการทำงาน

- ทำงาน 1 ครั้ง เมื่อ Collider ของวัตถุ 2 ชิ้น เริ่มสัมผัสกัน
- เหมาะสำหรับเหตุการณ์ที่ต้องการให้เกิด ทันทีที่ชน เช่น ลดพลังชีวิต เล่นเอฟเฟกต์เสียง / Particleนับจำนวนการชน

```
1    void OnCollisionEnter(Collision collision)
2    {
3        Debug.Log("เริ่มชนกับ: " + collision.gameObject.name);
4    }
```

ตัวอย่างการใช้งาน
1. ลูกกระสุนโดนศัตรูแล้วลด HP
2. ตัวละครตกพื้นแล้วตั้งค่า isGrounded = true

OnCollisionStay()

ถูกเรียก “ทุกเฟรม” ที่ยังคงชนกันอยู่โดยลักษณะการทำงาน

- จะถูกเรียกซ้ำ ทุก Physics Update (FixedUpdate)
- ตราบใดที่ Collider ยังสัมผัสกันอยู่
- เหมาะกับเหตุการณ์ที่ต้องตรวจสอบต่อเนื่อง เช่น ยืนบนพื้นรับดาเมจต่อเนื่องแรงเสียดทาน / แรงดัน

```
1   void OnCollisionStay(Collision collision)
2   {
3       Debug.Log("กำลังชนอยู่กับ: " + collision.gameObject.name);
4   }
5
```

ตัวอย่างการใช้งาน
1. ตัวละครยืนบนลาวาแล้วลด HP ต่อเนื่อง
2. กล่องถูกดันตลอดเวลา

OnCollisionExit()

ถูกเรียก “ครั้งเดียว” เมื่อวัตถุแยกออกจากกันโดยลักษณะการทำงาน

- ทำงานเมื่อ Collider เลิกสัมผัสกัน
- เหมาะกับเหตุการณ์หลังจากออกจากการชน เช่น ออกจากพื้นหยุดรับดาเมจรีเซ็ตสถานะ

```
1    void OnCollisionExit(Collision collision)
2    {
3        Debug.Log("เลิกชนกับ: " + collision.gameObject.name);
4    }
5
```

ตัวอย่างการใช้งาน
1. ตัวละครกระโดดออกจากพื้นแล้ว isGrounded = false
2. ออกจากพื้นที่อันตรายแล้วหยุดลด HP

เงื่อนไขสำคัญที่ต้องมี

1. อย่างน้อย หนึ่งวัตถุ ต้องมี Rigidbody
2. ทั้งสองวัตถุต้องมี Collider
3. ถ้าใช้ Is Trigger แล้วต้องใช้ OnTriggerEnter / Stay / Exit แทน

ตัวอย่างการใช้ใน miniGameNo1

```
void OnCollisionEnter(Collision collision)
{
        // ตรวจสอบว่า Object ที่เราชนมี Tag ว่า Player หรือไม่
        if (collision.gameObject.CompareTag("Player"))
        {
            // ทำลายศัตรูทิ้งทันทีหลังชน
            Destroy(gameObject);
        }
}
```

OnTriggerEnter()

- เป็นคำสั่งสำหรับตรวจจับการ “ทับซ้อน (Overlap)” ของ Collider แบบ Trigger ใน Unity นิยมใช้กับเหตุการณ์ที่ไม่ต้องการแรงฟิสิกส์ชนกันจริง แต่ต้องการรู้ว่า “มีวัตถุเข้ามาในพื้นที่หรือไม่”
- ถูกเรียก “ครั้งเดียว” เมื่อCollider ของวัตถุหนึ่ง เข้าไปใน Trigger Collider ของอีกวัตถุ โดยลักษณะการทำงาน
  - ไม่มีแรงกระแทก
  - วัตถุทะลุผ่านกันได้
  - ใช้สำหรับตรวจพื้นที่ / เขต / โซน

```
1    void OnTriggerEnter(Collider other)
2    {
3        Debug.Log("มีวัตถุเข้ามา: " + other.gameObject.name);
4    }
```

- other คือ Collider ของวัตถุที่เข้ามาใน Trigger
- ไม่ใช้ Collision เหมือน OnCollisionEnter

เขตเก็บไอเทม

```
1    void OnTriggerEnter(Collider other)
2    {
3        if (other.CompareTag("Player"))
4        {
5            Debug.Log("เก็บไอเทมได้!");
6            Destroy(gameObject);
7        }
8    }
```

เข้าพื้นที่อันตราย

```
1   void OnTriggerEnter(Collider other)
2   {
3       if (other.CompareTag("Player"))
4       {
5           playerHP -= 10;
6       }
7   }
```

ตรวจว่าผู้เล่นเข้าประตูหรือยัง

```
1    void OnTriggerEnter(Collider other)
2    {
3        if (other.CompareTag("Player"))
4        {
5            OpenDoor();
6        }
7    }
```

OnTriggerStay()

ถูกเรียก “ทุกเฟรม” เมื่อ Collider ของวัตถุหนึ่งอยู่ในเขตTrigger Collider ของอีกวัตถุ

- ถูกเรียกทุกเฟรม (ทุก Update)
- ทำงานตราบใดที่ Collider ยังทับซ้อนกับ Trigger อยู่
- ใช้สำหรับเหตุการณ์ที่ต้องตรวจสอบหรือทำงานต่อเนื่อง

```
1    void OnTriggerStay(Collider other)
2    {
3        Debug.Log("กำลังอยู่ใน Trigger กับ: " + other.gameObject.name);
4    }
```

ตัวอย่างการใช้งาน

- พื้นที่อันตรายแล้วลด HP ต่อเนื่อง
- เขตฮีลแล้วเพิ่ม HP ทีละนิด
- AI ตรวจจับผู้เล่นขณะยังอยู่ในระยะ

```
1    void OnTriggerStay(Collider other)
2    {
3        if (other.CompareTag("Player"))
4        {
5            playerHP -= Time.deltaTime * 5;
6        }
7    }
```

OnTriggerExit()

ถูกเรียก “ครั้งเดียว” เมื่อCollider ของวัตถุหนึ่งออกจากเขต Trigger Collider ของอีกวัตถุ

- ทำงานเมื่อ Collider เลิกทับซ้อน
- ใช้สำหรับรีเซ็ตสถานะ หรือหยุดการทำงานบางอย่าง

```
1    void OnTriggerExit(Collider other)
2    {
3        Debug.Log("ออกจาก Trigger กับ: " + other.gameObject.name);
4    }
```

ตัวอย่างการใช้งาน

- ออกจากเขตอันตรายแล้วหยุดลด HP
- ออกจากโซนตรวจจับแล้ว AI หยุดไล่
- ออกจาก Checkpoint แล้วบันทึกสถานะ

```
1    void OnTriggerExit(Collider other)
2    {
3        if (other.CompareTag("Player"))
4        {
5            isInDangerZone = false;
6        }
7    }
```

เงื่อนไขสำคัญที่ต้องมี

1. วัตถุอย่างน้อย 1 ชิ้นต้องมี Rigidbody
2. ต้องมี Collider
3. Trigger ต้องตั้งค่า Is Trigger = true

ภาพโดย ChatGPT

Trigger vs Collision

หัวข้อ	Trigger	Collision
เมธอด	OnTriggerEnter / Stay / Exit	OnCollisionEnter / Stay / Exit
ต้องตั้งค่า Is Trigger	ต้องเปิด	ต้องปิด
การชนทางฟิสิกส์	ไม่มีแรงชน	มีแรงชน
วัตถุทะลุผ่านกัน	ได้	ไม่ได้
ใช้ Rigidbody	อย่างน้อย 1 วัตถุ	อย่างน้อย 1 วัตถุ
ใช้ข้อมูลแรงชน	ไม่ได้	ได้ (แรง, จุดชน, normal)
ความถี่การเรียก Stay	ทุกเฟรม	ทุก FixedUpdate
เหมาะกับงาน	เขต / โซน	การชนจริง
ประสิทธิภาพ	ทำงานน้อยกว่า	ทำงานหนักกว่าเล็กน้อย
ตัวอย่างทั่วไป	เก็บไอเทม, Checkpoint	ลูกบอลชนกำแพง

Trigger

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player")
        Debug.Log("Player เข้าโอน"),
}
```

Collision

```
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("
        Debug.Log("Player ชนวัตถุ");
}
```

Trigger = ตรวจพื้นที่

Collision = ชนจริงตามเฟิสิกส์

ภาพโดย ChatGPT

เงื่อนไขสำคัญที่ต้องมี

1. ถ้าเปิด Is Trigger แล้ว Collision จะไม่ทำงาน
2. ถ้าต้องการแรงฟิสิกส์แล้วห้ามใช้ Trigger
3. Trigger เหมาะกับ Logic / Event
4. Collision เหมาะกับ Physics / Movement

การสุ่มจุดเกิด

ภาพโดย ChatGPT

การสุ่มค่า (Randomization) เป็นส่วนสำคัญในการสร้างความหลากหลายให้เกม ไม่ว่าจะเป็นการสุ่มตำแหน่งเกิดศัตรู, สุ่มค่าพลังโจมตี หรือสุ่มสี โดยเครื่องมือหลักที่เราใช้คือคลาส

UnityEngine.Random

การสุ่มตัวเลข (Numbers)

- การสุ่มเลขทศนิยม (float) ใช้ Random.Range(min, max) โดยค่าที่ได้จะรวม min และ max ด้วย

```
float randomSpeed = Random.Range(1.5f, 5.0f);
```

- การสุ่มเลขจำนวนเต็ม (int) ข้อควรระวัง: สำหรับ int ค่าสูงสุด (max) จะ ไม่ถูกรวม ในผลลัพธ์ (Exclusive)

```
// สุ่มเลข 1, 2, 3, 4 เท่านั้น (ไม่รวม 5)
int randomDamage = Random.Range(1, 5);
```

การสุ่มตำแหน่งและทิศทาง (Physics & Spawn)

ใน Unity 6.0 เรามักสุ่มตำแหน่งเพื่อสร้าง Object รอบๆ ตัวผู้เล่น

- Random.insideUnitCircle สุ่มตำแหน่งภายในวงกลมรัศมี 1 หน่วย (ใช้กับเกม 2D หรือวางบนพื้น)

```
Vector2 randomArea = Random.insideUnitCircle * 5f; // รัศมี 5 หน่วย
```

- Random.insideUnitSphere สุ่มตำแหน่งภายในทรงกลม (ใช้กับเกม 3D)

```
Vector3 random3DPoint = Random.insideUnitSphere * 10f;
```

- Random.onUnitSphere สุ่มตำแหน่ง เฉพาะที่ผิว ของทรงกลม (เหมาะสำหรับสุ่มทิศทางกระจายออก)

การสุ่มค่าความน่าจะเป็น (Probability)

หากต้องการเช็คว่า "ติดคริติคอลไหม?" หรือ "ไอเทมดรอปไหม?" ให้ใช้ Random.value ซึ่งจะคืนค่าระหว่าง 0.0 ถึง 1.0

```
float dropChance = 0.2f; // โอกาส 20%
if (Random.value <= dropChance)
{
    Debug.Log("Item Dropped!");
}
```

ระบบสุ่มแบบใหม่ใน Unity 6 Random.State

ใน Unity 6 (และเวอร์ชันก่อนหน้า) หากต้องการให้การสุ่ม "ล็อคผล" ไว้ล่วงหน้า เช่น การสร้างแผนที่แบบ Procedural ที่ใช้ Code เดิมแล้วได้แมพหน้าตาเดิม เป็นต้น เราจะใช้ Seed

```
// ตั้งค่า Seed เป็นตัวเลขอะไรก็ได้
Random.InitState(12345);
// ทุกครั้งที่รัน Code นี้ ผลลัพธ์ที่ได้จะเหมือนเดิมเสมอ
float constantRandom = Random.Range(0f, 10f);
```

การสุ่มเลือกของจาก Array หรือ List

วิธีที่ง่ายที่สุดคือการสุ่ม Index ของรายการนั้นๆ

```
public string[] enemyNames = { "Slime", "Goblin", "Orc" };
void SpawnEnemy()
{     int randomIndex = Random.Range(0, enemyNames.Length);
      string selectedEnemy = enemyNames[randomIndex];
      Debug.Log("Spawned: " + selectedEnemy);
}
```

คำแนะนำ

หากต้องใช้การสุ่มจำนวนมหาศาลในระบบ Job System ของ Unity 6 แนะนำให้ศึกษา Unity.Mathematics.Random ซึ่งจะทำงานได้เร็วกว่า UnityEngine.Random ปกติ

ใน Unity 6 เมื่อต้องการประสิทธิภาพสูงสุด เช่น การใช้ร่วมกับ ECS หรือ Job System เป็นต้น UnityEngine.Random จะมีข้อจำกัดเพราะมันทำงานบน Main Thread เท่านั้นและไม่ปลอดภัยต่อการประมวลผลแบบขนาน (Not Thread-Safe)

ภาพโดย ChatGPT

Unity จึงมี Unity.Mathematics.Random ซึ่งเป็นโครงสร้างแบบ Struct ที่ออกแบบมาให้ทำงานได้รวดเร็วมากและรองรับการประมวลผลแบบ Multi-threading

วิธีการใช้งานพื้นฐาน

เนื่องจากมันเป็น Struct จึงต้องสร้างตัวแปรขึ้นมาและกำหนด Seed (เลขสุ่มเริ่มต้น) เสมอ

```
using UnityEngine;
using Unity.Mathematics; // ต้อง Import ตัวนี้

public class MathRandomExample : MonoBehaviour
{
    void Start()
    {
        // 1. สร้าง Instance และกำหนด Seed (ห้ามเป็น 0)
        // uint.MaxValue เป็นตัวเลือกที่ดีในการสุ่ม Seed เริ่มต้น
        var random = new Unity.Mathematics.Random(1234);

        // 2. การสุ่มเลขทศนิยม (float) ระหว่าง 0 ถึง 10
        float randFloat = random.NextFloat(0f, 10f);

        // 3. การสุ่มเลขจำนวนเต็ม (int) ระหว่าง 1 ถึง 100
        int randInt = random.NextInt(1, 101); // (min, max) โดยรวม max ด้วย

        // 4. การสุ่มทิศทาง (Vector3)
        float3 randDir = random.NextFloat3Direction();

        Debug.Log($"Float: {randFloat}, Int: {randInt}");
    }
}
```

ตัวอย่างการใช้ใน Job System

จุดที่ Unity.Mathematics.Random แสดงพลังออกมาได้เต็มที่ครับ คือ การสุ่มค่าให้วัตถุจำนวนมากพร้อมกันโดยไม่ทำให้เกมกระตุก

```
using UnityEngine;
using Unity.Burst;
using Unity.Jobs;
using Unity.Mathematics;
using Unity.Collections;

public class RandomJobManager : MonoBehaviour
{
    struct MyRandomJob : IJobParallelFor
    {
        // เราต้องมี Array ของ Random สำหรับแต่ละ Thread
        public NativeArray<Unity.Mathematics.Random> randoms;
        public NativeArray<float3> results;

        public void Execute(int index)
        {
            // ดึงสถานะ Random ของ Thread นั้นๆ ออกมา
            var rng = randoms[index];
            // สุ่มตำแหน่งใหม่
            results[index] = rng.NextFloat3(new float3(-10, 0, -10), new float3(10, 0, 10));
            // สำคัญ: บันทึกสถานะ Random กลับลงไปเพื่อให้การสุ่มครั้งหน้าไม่ซ้ำเดิม
            randoms[index] = rng;
        }
    }
}
```

สรุปคำสั่งสุ่มที่ใช้บ่อยใน Mathematics

NextFloat(min, max) สุ่มเลขทศนิยม

NextInt(min, max) สุ่มเลขจำนวนเต็ม

NextBool() สุ่มค่าจริงหรือเท็จ (True/False)

NextFloat3() สุ่มค่า Vector3 (float3)

NextQuaternionRotation() สุ่มมุมหมุน

ข้อควรระวัง

- อย่าใช้ Seed เป็น 0 เนื่องจากคลาสนี้จะ Error หรือได้ผลลัพธ์ที่ไม่สุ่มถ้า Seed เป็น 0
- ถ้าสร้าง new Random(seed) ไว้ใน Update() จะได้เลขเดิมตลอดกาล เพราะมันเริ่มนับหนึ่งใหม่ทุกเฟรม ดังนั้น ควรประกาศตัวแปรไว้ระดับ Class หรือส่งต่อสถานะไปเรื่อย ๆ

miniGameNo1 กำหนดให้มีจุดเกิดของ Enemy ที่ด้านบน 4 จุด โดยจะทำการสุ่ม 2 ลำดับ คือ สุ่มตำแหน่งนี้จะเกิด และสุ่มว่าจะเกิดออกมาเป็น Thing1 หรือ Thing2

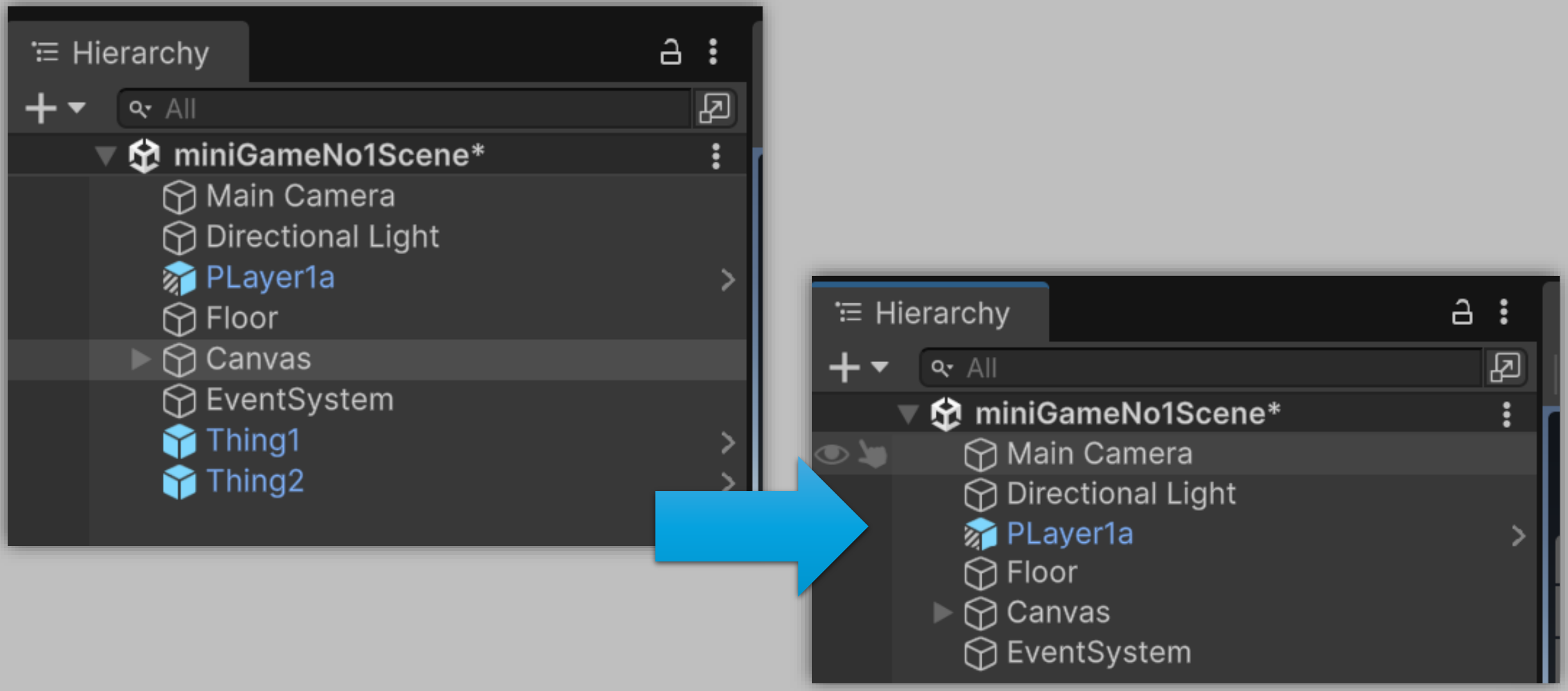
เพื่อให้ได้ประสิทธิภาพสูงสุดและโค้ดที่สะอาดใน Unity 6 เราจะใช้ Unity.Mathematics.Random ในการสุ่มเลือก Index ของตำแหน่ง (S1, S2, S3 หรือ S4) และสุ่ม Index ของประเภทวัตถุ (Thing1, Thing2)

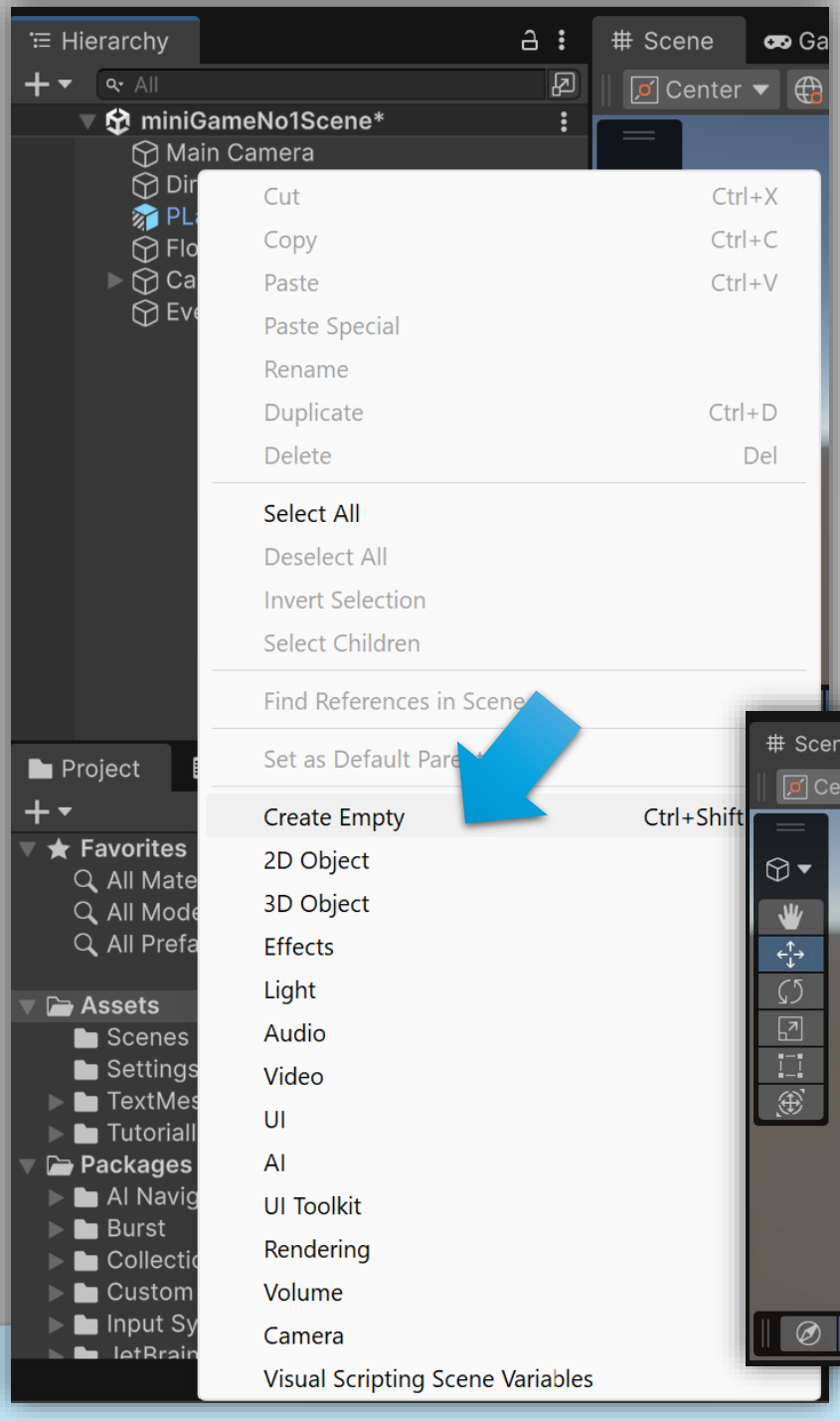
เตรียมการ

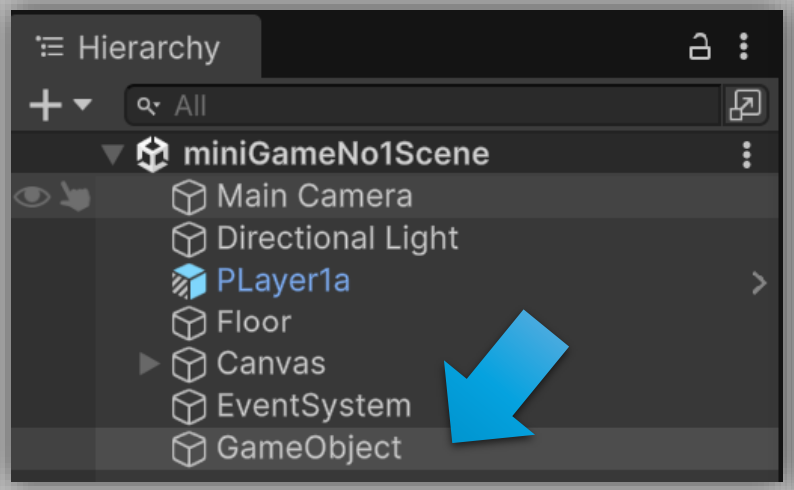
0. นำ Thing1 กับ Thing2 ออกจาก Scene

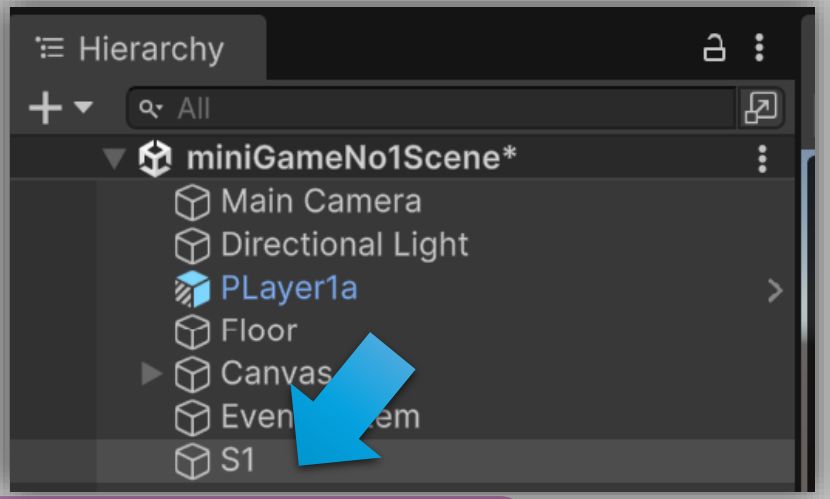
1. สร้าง Empty GameObject จำนวน 4 ตำแหน่ง วางไว้เพื่อเป็นจุดเกิดของ Enemy

2. สร้างสคริปต์ชื่อ ObjectSpawner.cs เพื่อทำการสุ่มเกิดวัตถุ Thing1 หรือ Thing2 ณ ตำแหน่งเกิดที่สุ่มได้

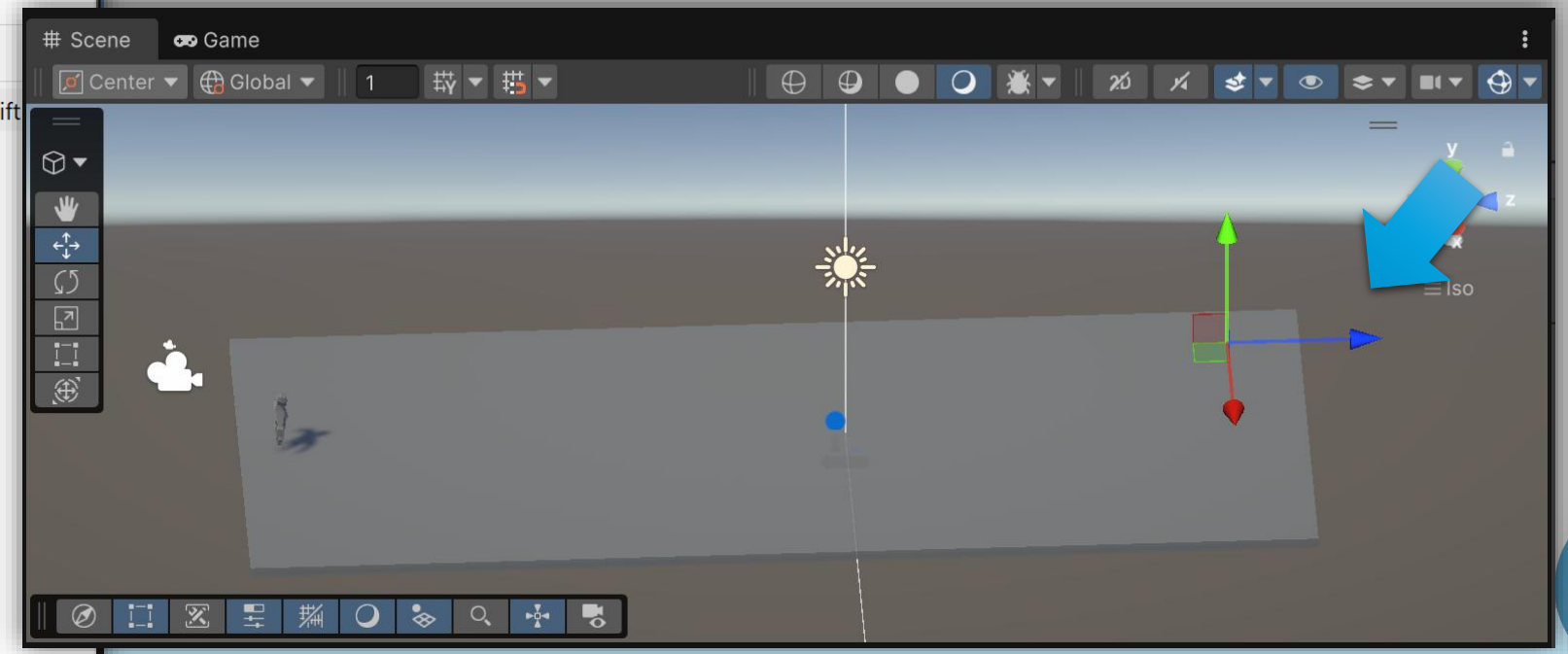

Hierarchy
All
miniGameNo1Scene*
Main Camera
Directional Light
PLayer1a
Floor
Canvas
EventSystem
Thing1
Thing2
Hierarchy
All
miniGameNo1Scene*
Main Camera
Directional Light
PLayer1a
Floor
Canvas
EventSystem

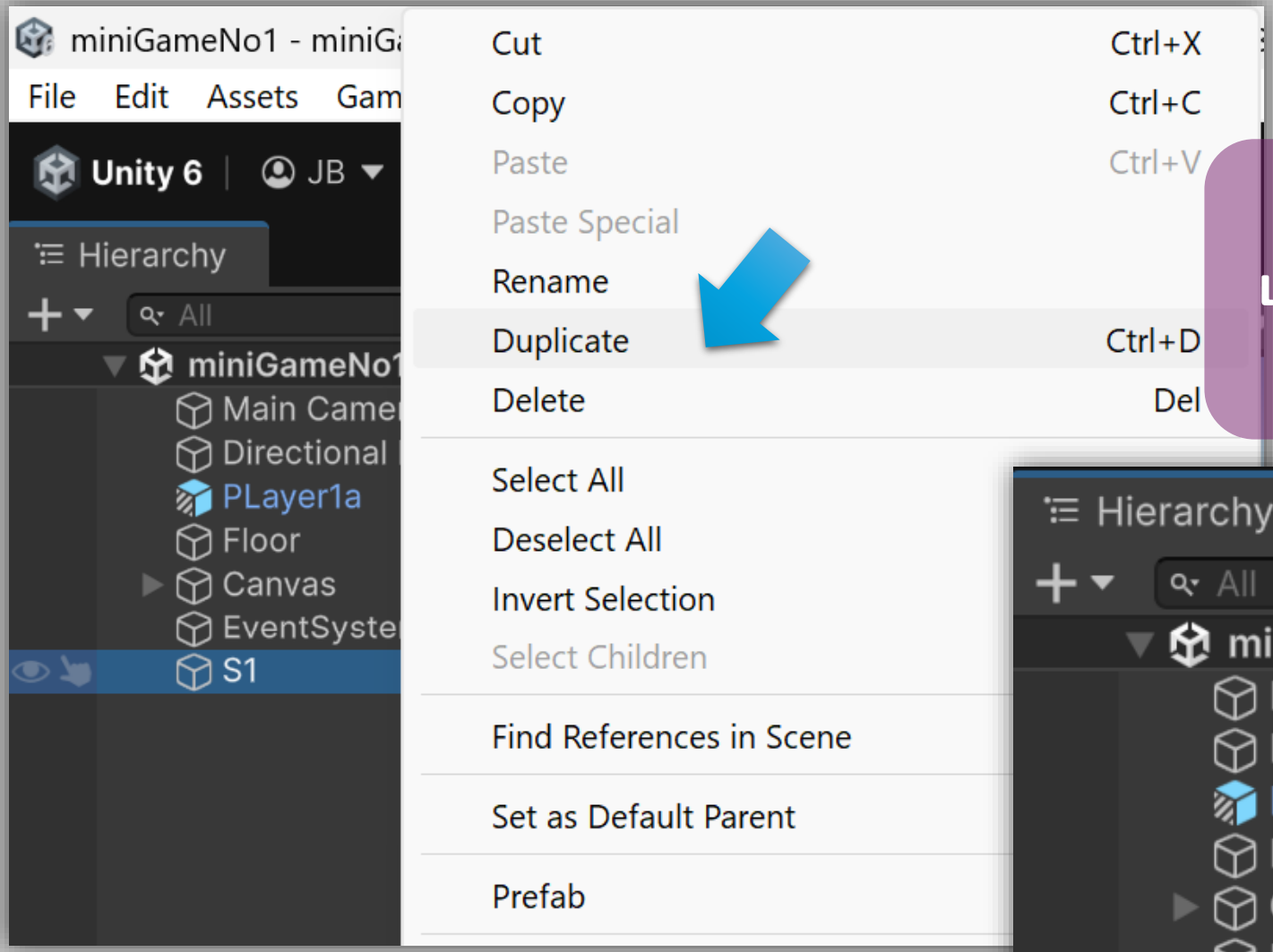




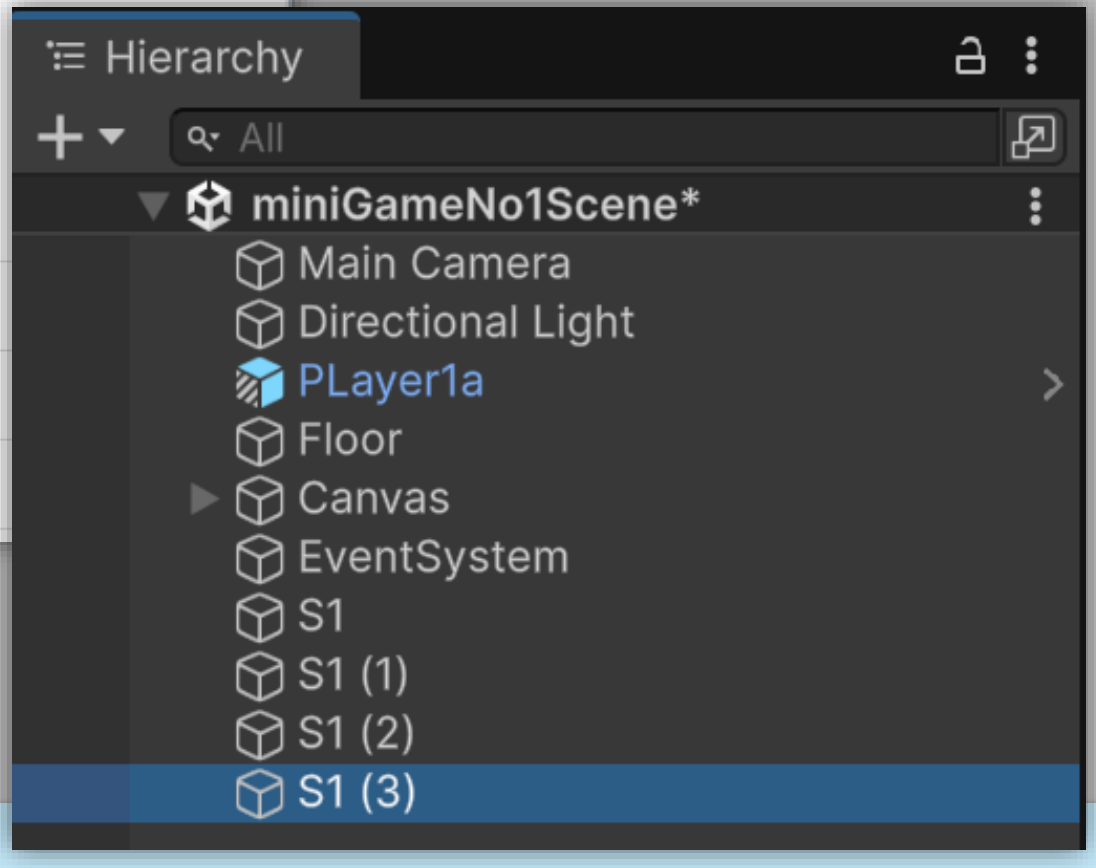


เปลี่ยนชื่อ GameObject เป็น S1

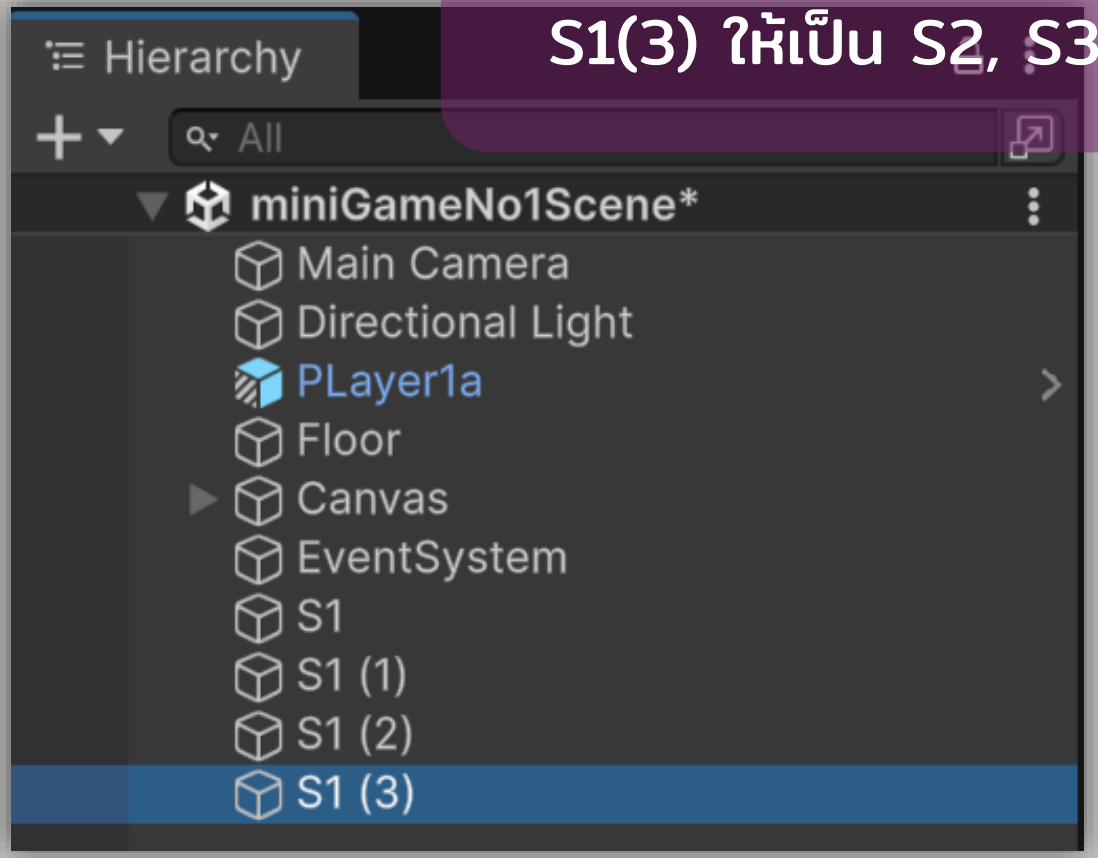


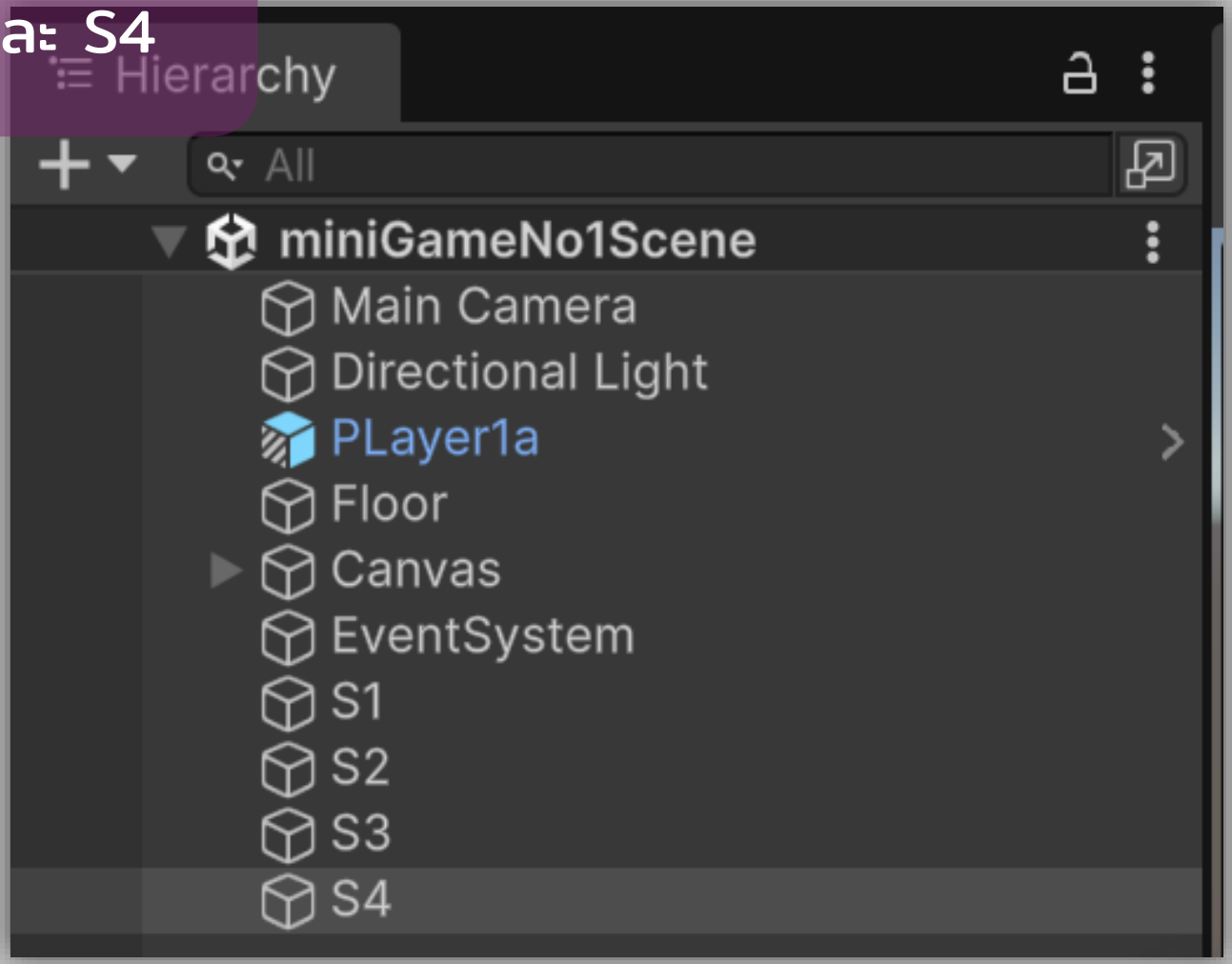


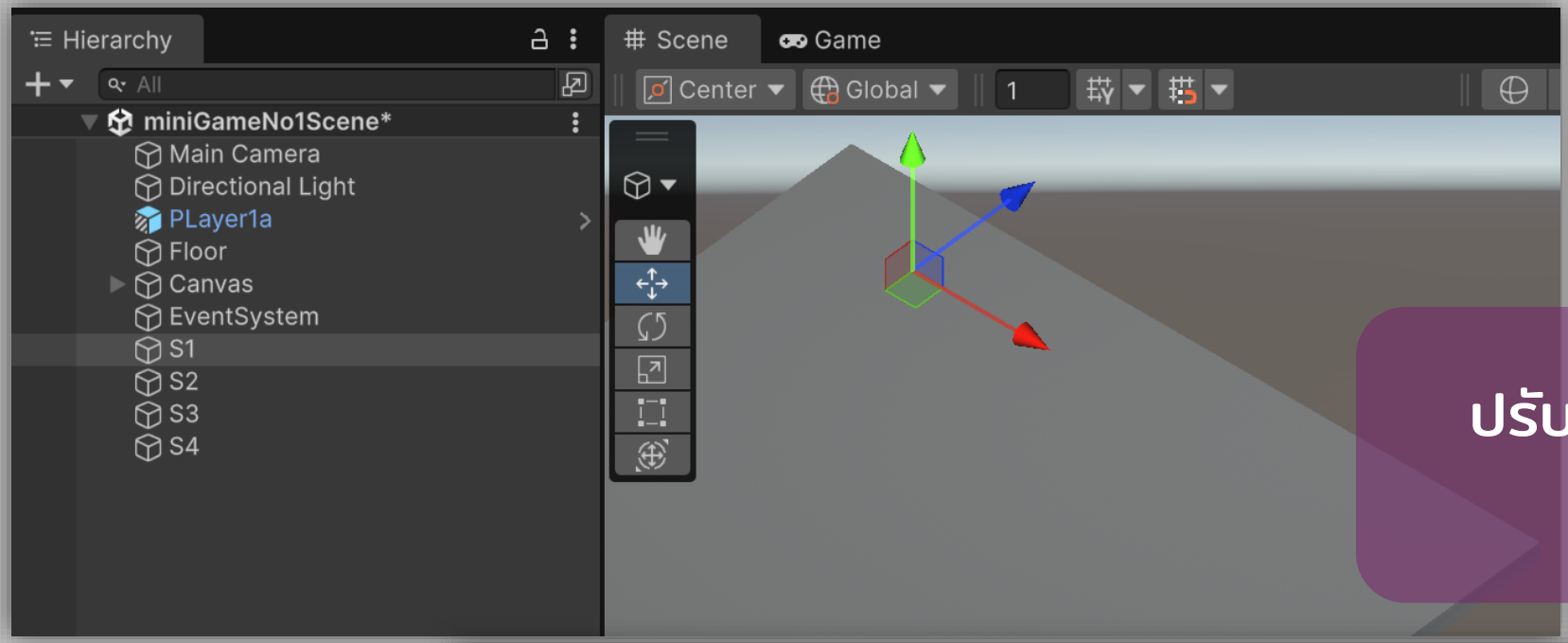
คลิกขวาที่ S1 แล้วเลือก Duplicate เพื่อคัดลอก ให้คัดลอกมาอีก 3 ชิ้น จะได้ S1 (1), S1 (2) และ S1(3)



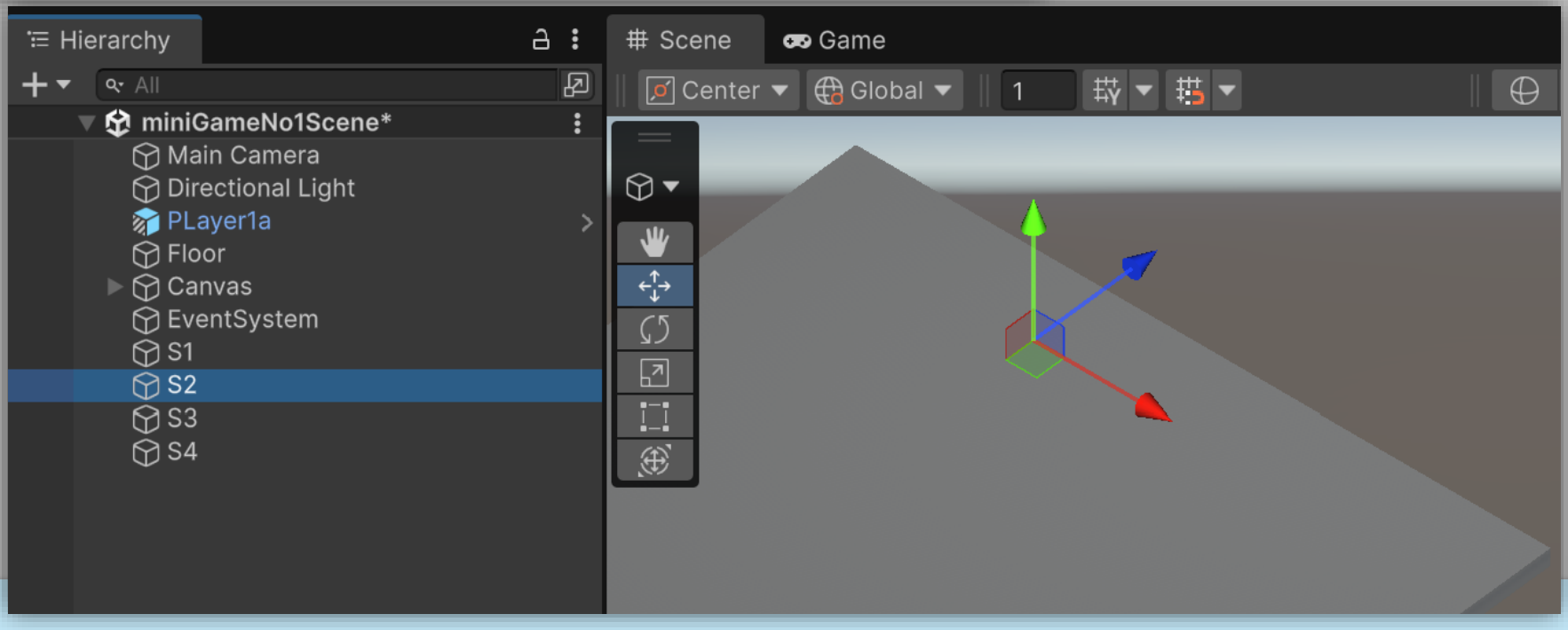
เปลี่ยนชื่อ S1 (1), S1 (2) และ S1(3) ให้เป็น S2, S3 และ S4

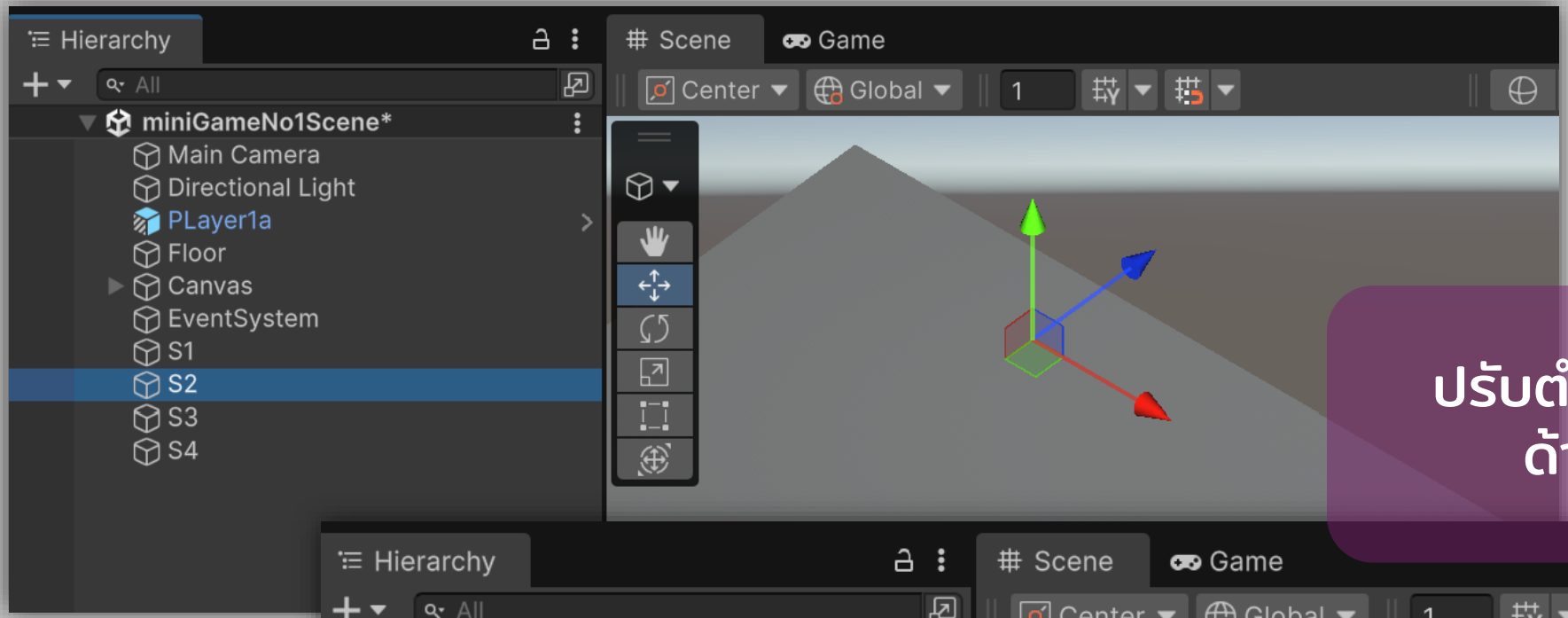




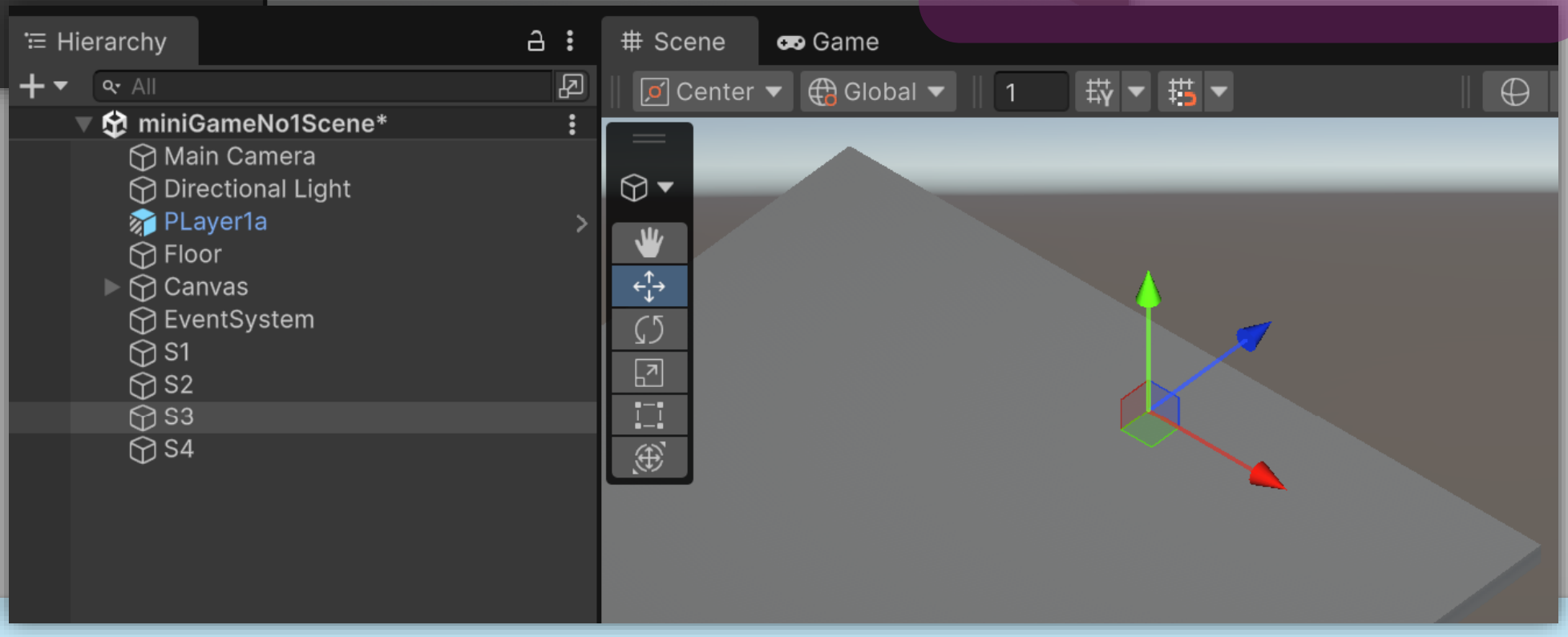


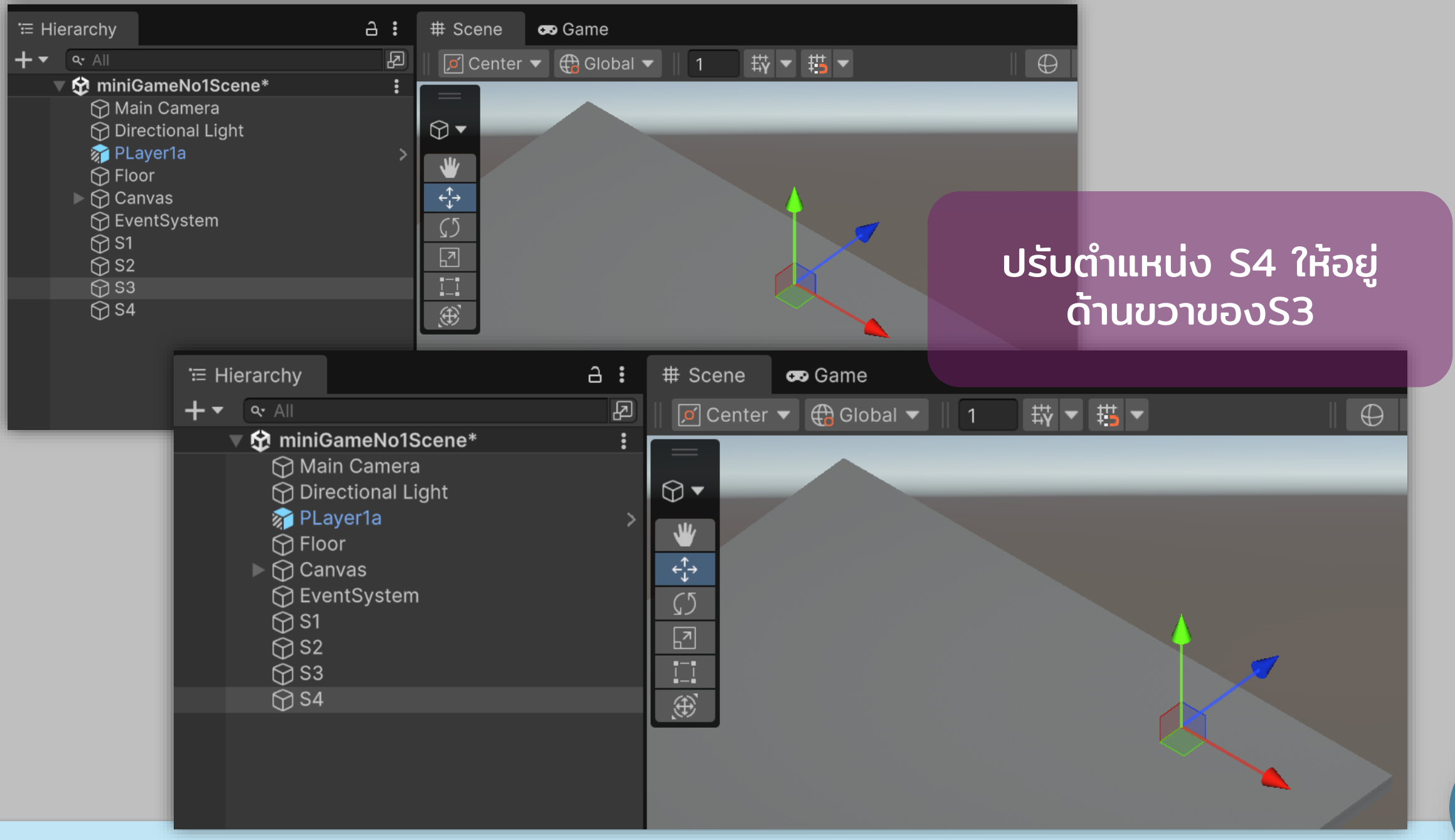
ปรับตำแหน่ง S2 ให้อยู่ด้านขวาของS1

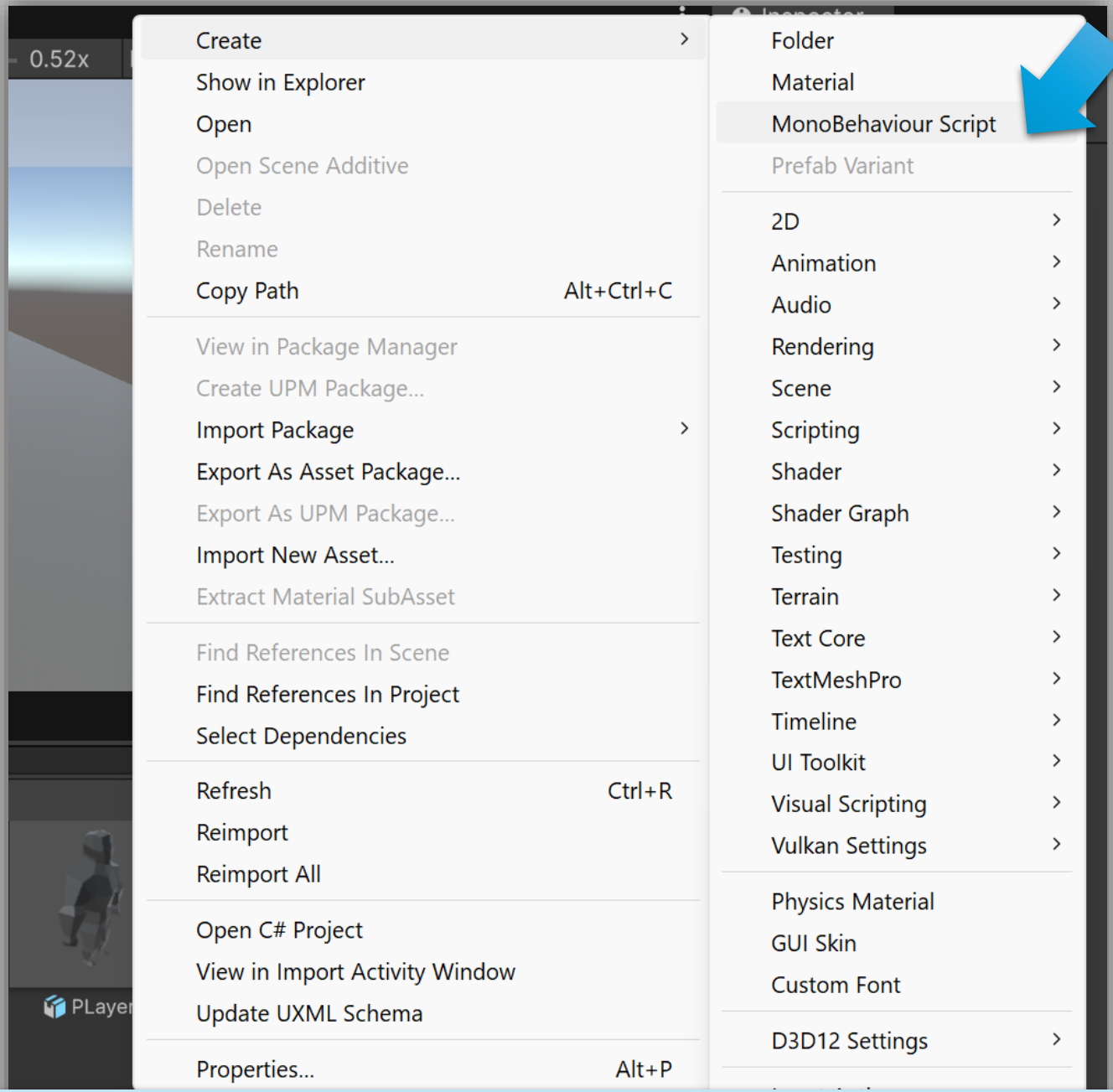




ปรับตำแหน่ง S3 ให้อยู่ด้านขวาของS2




Hierarchy
Scene
Game
All
Center
Global
1
miniGameNo1Scene*
Main Camera
Directional Light
PLayer1a
Floor
Canvas
EventSystem
S1
S2
S3
S4
ปรับตำแหน่ง S4 ให้อยู่
ด้านขวาของS3



สร้างสคริปต์ ObjectSpawner

ObjectSpawner.cs | Enemy2.cs | Enemy1.cs | UIManager.cs | PlayerActivity.cs | PlayerProperty.cs

Assembly-CSharp | ObjectSpawner | SpawnRandomObject()

```
1   using UnityEngine;
2   using Unity.Mathematics;
3   using UnityEngine.InputSystem; // สำหรับการสุ่มประสิทธิภาพสูง
4
    Unity Script (1 asset reference) | 0 references
5   public class ObjectSpawner : MonoBehaviour
6   {
7       [Header("Prefabs")]
8       public GameObject prefabA;
9       public GameObject prefabB;
10
11      [Header("Spawn Points")]
12      public Transform[] spawnPoints; // ลาก S1, S2, S3, S4 มาใส่ที่นี่
13
14      private Unity.Mathematics.Random rnd;
```

```
15      void Awake()
16      {
17          // สร้างตัวสุ่มโดยใช้ Seed จากเวลาปัจจุบัน (ห้ามใช้ 0)
18          uint seed = (uint)System.DateTime.Now.Ticks;
19          rnd = new Unity.Mathematics.Random(seed == 0 ? 1 : seed);
20      }
21
```

```
22      // ฟังก์ชันสำหรับเรียกใช้งานสุ่ม
        1 reference
23      public void SpawnRandomObject()
24      {
25          if (spawnPoints.Length == 0)
26          {
27              return;
28          }
29          // 1. สุ่มเลือกจุดเกิด (S1 - S4)
30          // NextInt(min, max) -> ใน Mathematics รวมค่า max ด้วย ดังนั้นต้อง -1
31          int pointIndex = rnd.NextInt(0, spawnPoints.Length);
32          Transform selectedPoint = spawnPoints[pointIndex];
33          // 2. สุ่มเลือกว่าจะสร้างวัตถุไหน (A หรือ B)
34          // ใช้ NextBool() เพื่อสุ่ม 50/50 หรือ NextInt(0, 2)
35          GameObject prefabToSpawn = rnd.NextBool() ? prefabA : prefabB;
36          // 3. สร้างวัตถุ (Instantiate)
37          Instantiate(prefabToSpawn, selectedPoint.position, selectedPoint.rotation);
38          // Debug.Log($"Spawned {prefabToSpawn.name} at {selectedPoint.name}");
39      }
40
```

```
41      void Update()
42      {
43          // ทดสอบด้วยการกดปุ่ม Space
44          if (Keyboard.current.spaceKey.isPressed)
45          {
46              SpawnRandomObject();
47          }
48      }
49  }
```

คำสั่ง Instantiate()

คำสั่งที่ใช้สำหรับ "การสำเนา (Clone)" วัตถุต้นแบบ (Original/Prefab) ให้กลายเป็น Object จริงขึ้นมาในฉาก (Scene) ในขณะที่เกมกำลังรันอยู่

เปรียบเทียบง่ายๆ คือ **Prefab** เป็นเหมือน "แม่พิมพ์" ส่วน **Instantiate()** คือการ "ปั๊มงาน" ออกมาจากแม่พิมพ์นั้น

แบบที่ 1 สร้างออกมาตรงตำแหน่งเดิมของ Prefab

```
Instantiate(prefab);
```

แบบที่ 2 ระบุตำแหน่ง และ มุมหมุน (นิยมใช้ที่สุด)

```
// Instantiate(วัตถุ, ตำแหน่ง, มุมหมุน);
// Quaternion.identity หมายถึง "ไม่มีการหมุน" (ค่า Rotation เป็น 0, 0, 0)
Instantiate(
        prefab,
        new Vector3(0, 5, 0),
        Quaternion.identity
);
```

แบบที่ 3 กำหนด Parent (ให้เข้าไปอยู่ใน Object อื่น)

```
// สร้างออกมาแล้วให้ไปเป็นลูกของ Object ที่ระบุ
Instantiate(prefab, parentTransform)
```

การเก็บค่าไว้ใช้งานต่อ (Reference)

เมื่อเราสั่ง Instantiate ตัวคำสั่งจะส่งค่ากลับมาเป็น Object ที่เพิ่งสร้างใหม่ เราควรสร้างตัวแปรมาเก็บค่านี้ไว้หากต้องการตั้งค่าเพิ่มทันที

```
// สร้างศัตรูออกมาแล้วเก็บไว้ในตัวแปร newEnemy
GameObject newEnemy = Instantiate(enemyPrefab,
                                  spawnPoint.position,
                                  spawnPoint.rotation);
// สั่งเปลี่ยนชื่อ หรือ ตั้งค่าพลังชีวิตทันที
newEnemy.name = "Enemy_Clone";
newEnemy.GetComponent<PlayerProperty>().hp = 50;
```

ขั้นตอนการทำงานของ Instantiate()

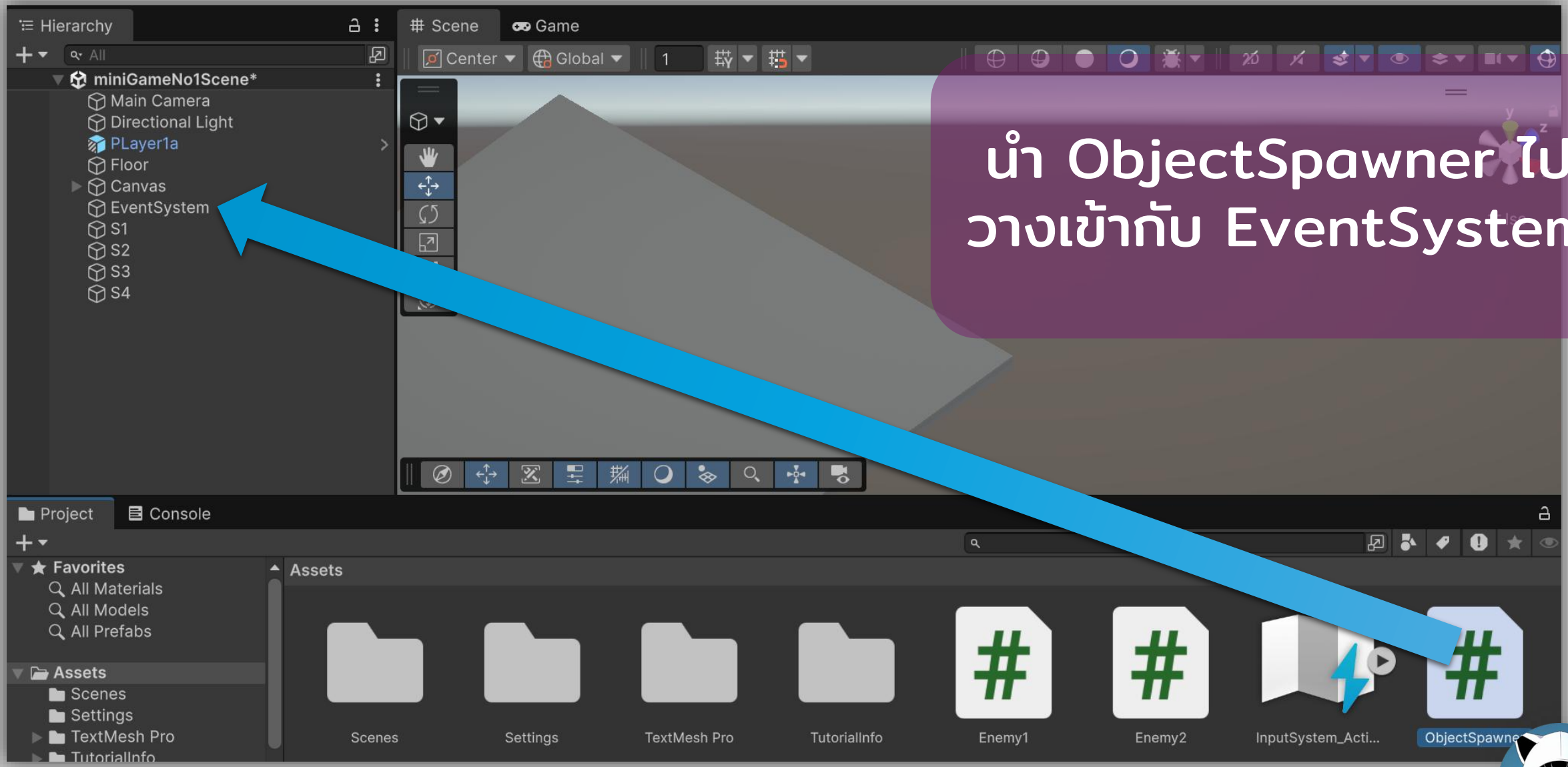
เมื่อเรียกใช้คำสั่งนี้ Unity จะทำสิ่งต่อไปนี้ตามลำดับ:

1. Allocate Memory จองพื้นที่ในหน่วยความจำสำหรับ Object ใหม่
2. Copy Data คัดลอกข้อมูลทั้งหมดจากต้นแบบ (Components, Mesh, Scripts, Values)
3. Awake & OnEnable เรียกใช้ฟังก์ชันเริ่มต้นใน Script ที่ติดมากับ Object นั้น
4. Physics & Rendering บรรจุเข้าสู่ระบบฟิสิกส์และการวาดภาพในฉาก

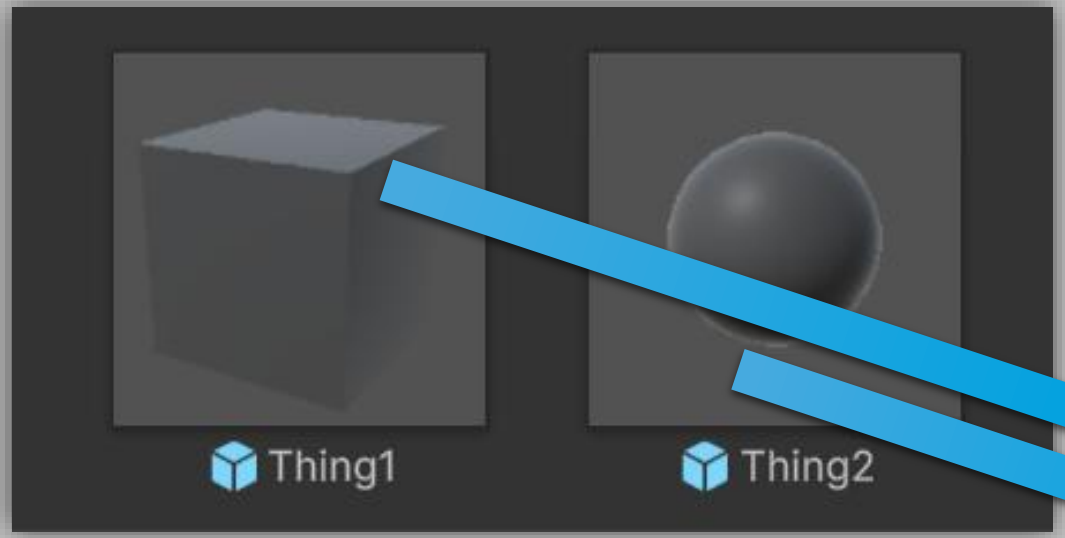
ข้อควรระวัง

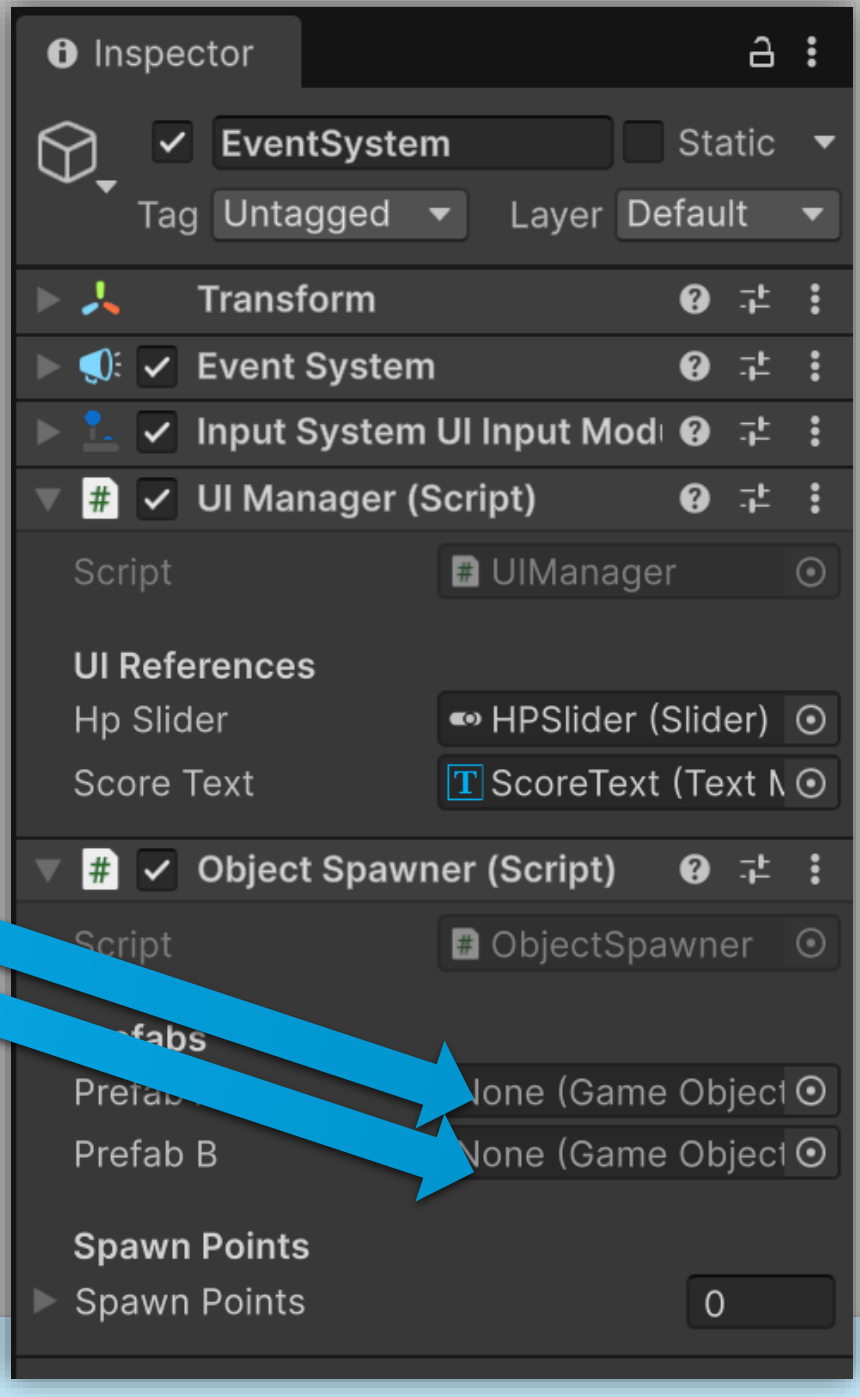
แม้ Instantiate() จะสะดวกมาก แต่ก็มี "ต้นทุน" ที่ต้องจ่าย:

- Garbage Collection
  การเรียกใช้บ่อยๆ ในเฟรมเดียวกัน เช่น สร้างกระสุน 100 นัดพร้อมกัน เป็นต้น จะทำให้เครื่องกระตุก (Lag Spikes) เพราะ Unity ต้องไปจองหน่วยความจำใหม่
- วิธีแก้
  หากต้องสร้างและทำลายวัตถุบ่อยๆ เช่น กระสุน, ศัตรูจำนวนมาก เป็นต้น ให้ศึกษาเรื่อง Object Pooling ซึ่งเป็นการนำวัตถุเก่ากลับมาใช้ใหม่แทนการทำลายทิ้ง


Hierarchy
Scene
Game
miniGameNo1Scene*
Main Camera
Directional Light
PLayer1a
Floor
Canvas
EventSystem
S1
S2
S3
S4
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Scenes
Settings
TextMesh Pro
TutorialInfo
Enemy1
Enemy2
InputSystem_Acti...
ObjectSpawne
นำ ObjectSpawner ไป
วางเข้ากับ EventSystem

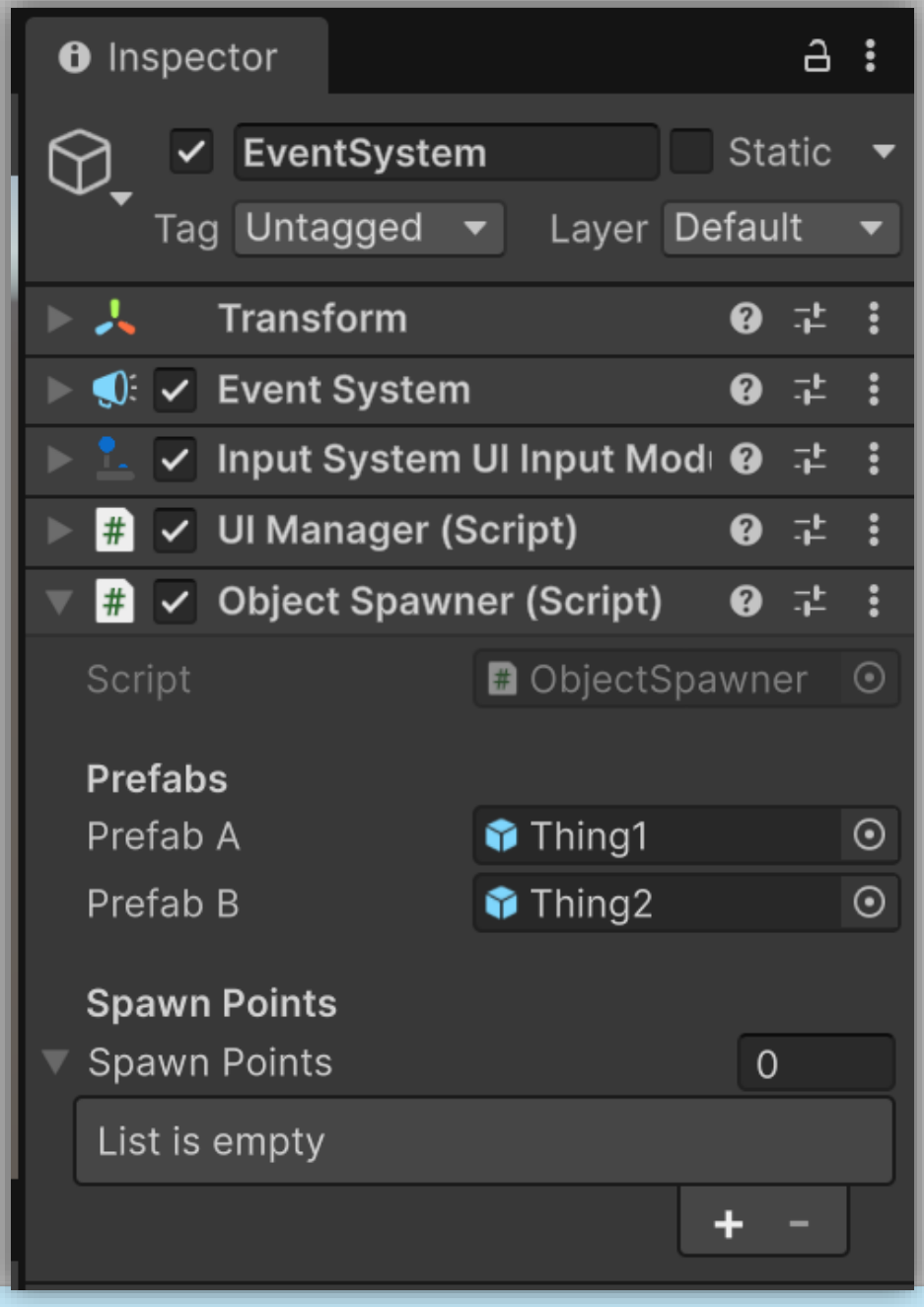
ลาก Thing1 ไปวางที่ Prefab A และลาก Thing2 ไปวางที่ Prefab B

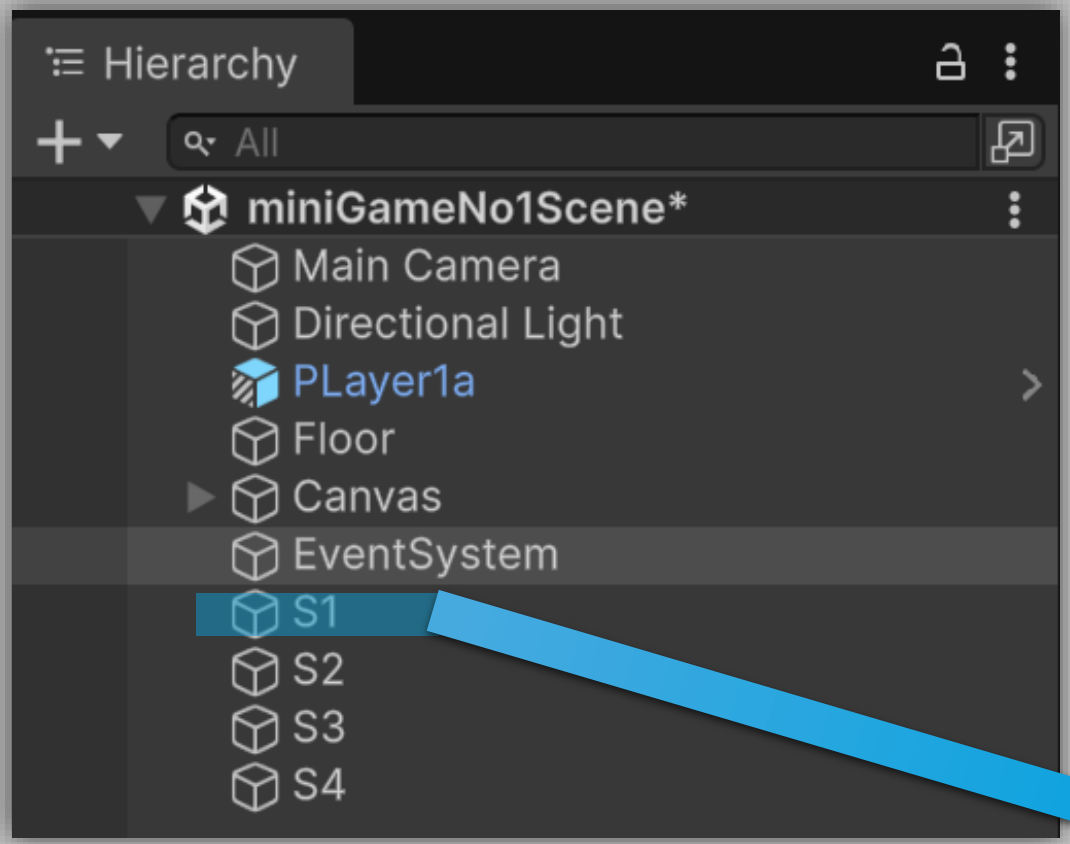


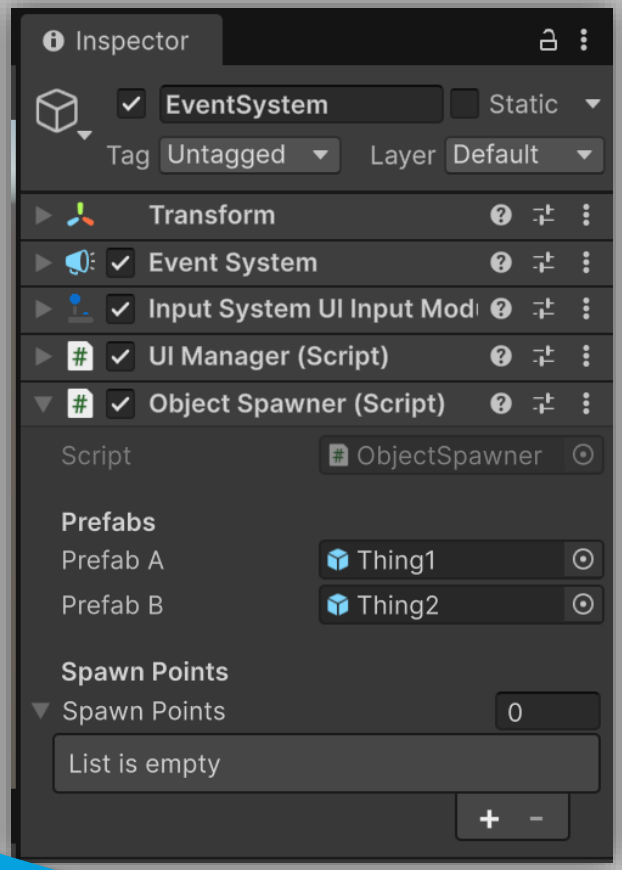


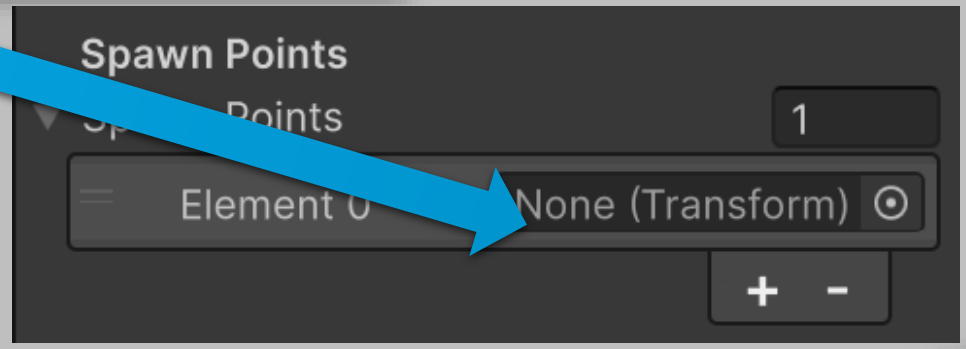
ผลลัพธ์จากการลาก Thing1 ไปวางที่ Prefab A และลาก Thing2 ไปวางที่ Prefab B

หลังจากนั้นกดลูกศรลงด้านหน้า Spawn Points เพื่อให้แสงช่อง [List is empty]



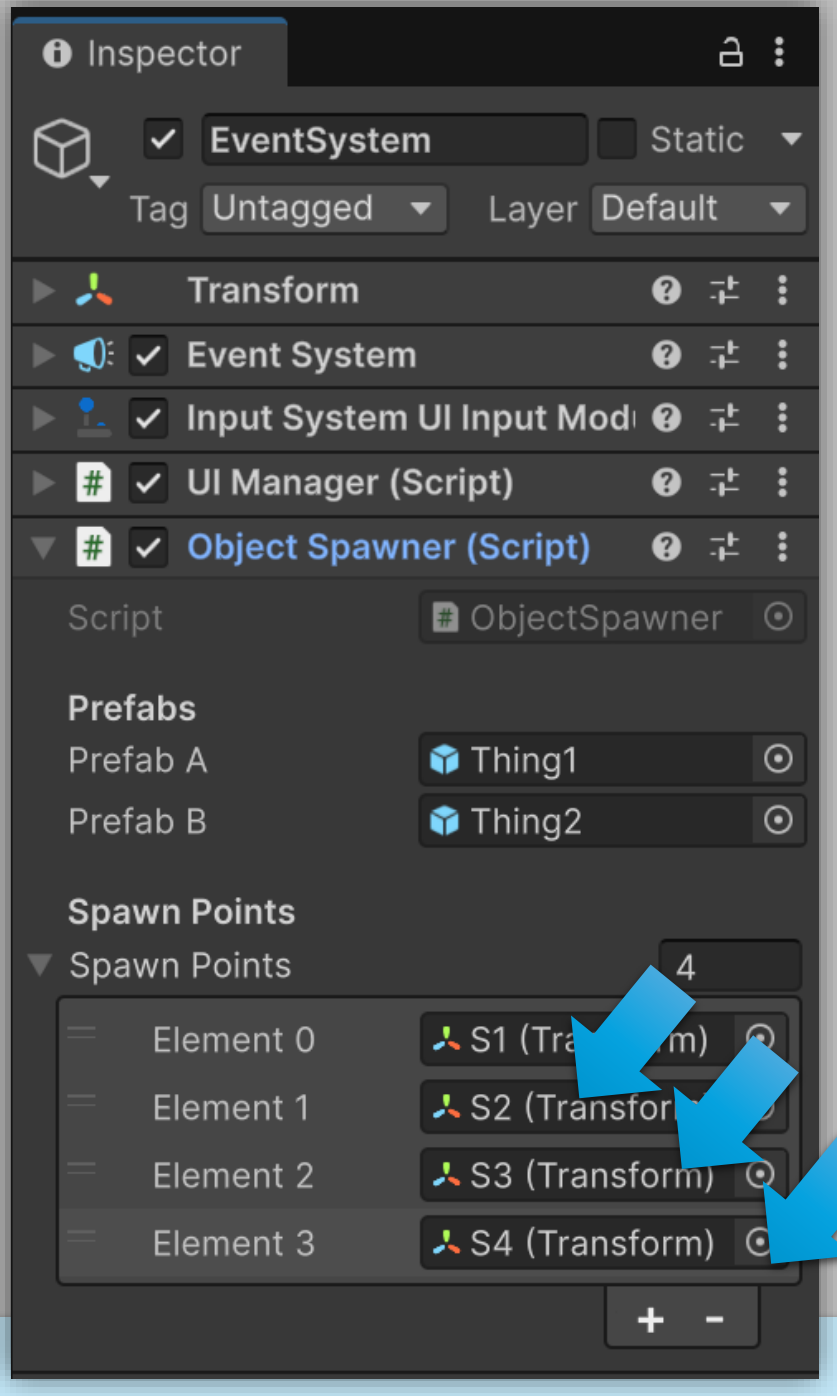


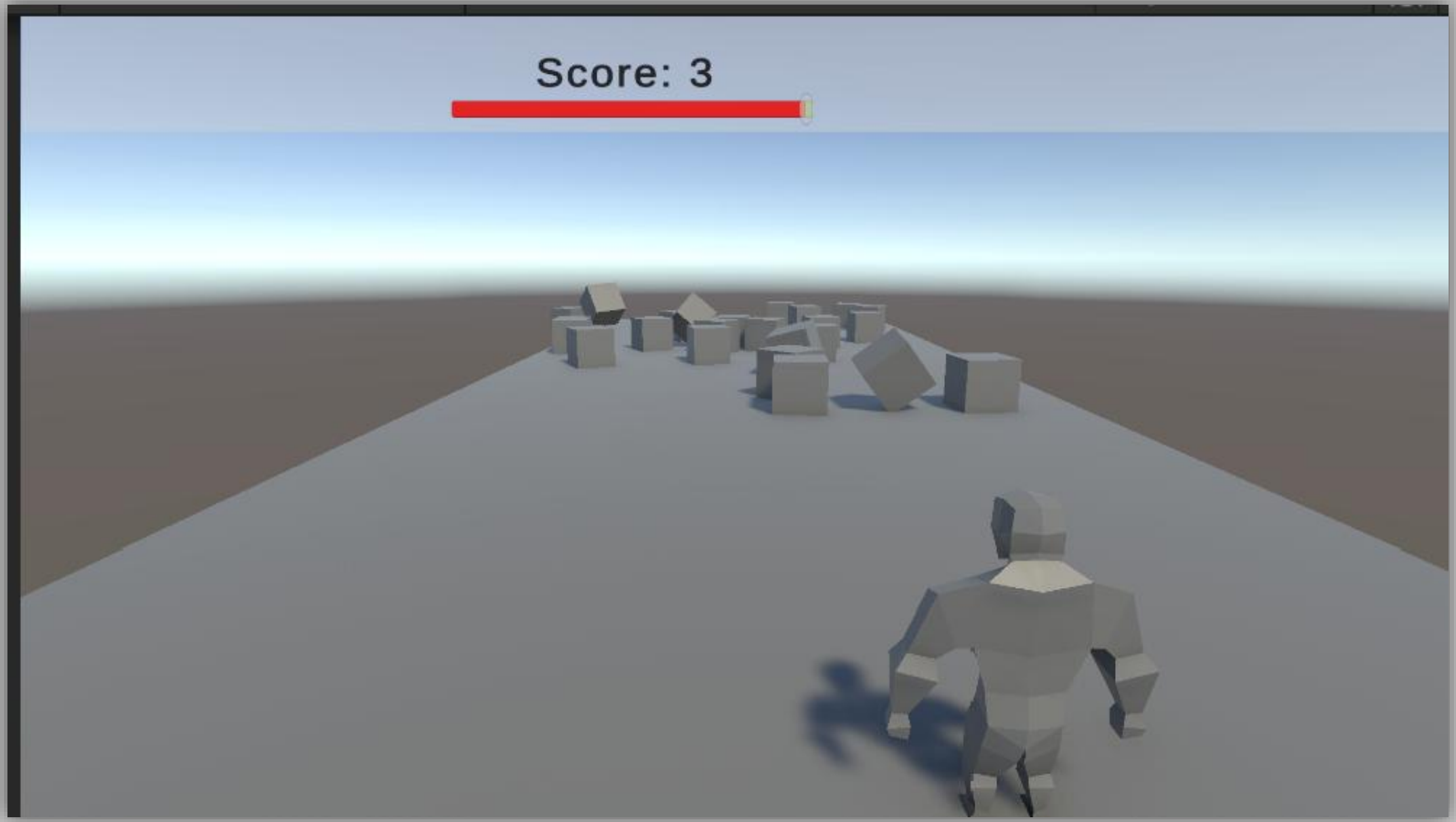




กด [+] แวจะมี Element 0
ปรากฏ ให้ลาก S1 ไปวาง

1. กด [+] แล้วจะมี Element 1 ปรากฏ ให้ลาก S2 ไปวาง
2. กด [+] แล้วจะมี Element 2 ปรากฏ ให้ลาก S3 ไปวาง
3. กด [+] แล้วจะมี Element 3 ปรากฏ ให้ลาก S4 ไปวาง




Score: 3

rnd.NextInt(0, spawnPoints.Length)

เป็นการสุ่มตัวเลขจำนวนเต็มเพื่อใช้เป็น Index ของ Array ถ้าคุณมีจุดเกิด 4 จุด (0, 1, 2, 3) คำสั่งนี้จะสุ่มเลขในช่วงนี้มาให้เรา

rnd.NextBool()

เป็นวิธีที่เร็วที่สุดในการสุ่มแบบ "สองทางเลือก" (50/50) ถ้าผลออกมาเป็น True จะเลือก Thing1 ถ้า False จะเลือก Thing2

Encapsulation

การใช้ Transform[] ทำให้คุณสามารถเพิ่มจุดเกิด S5, S6... ได้เรื่อยๆ ใน Inspector โดยไม่ต้องกลับมาแก้ Code

เทคนิคเพิ่มเติม

หากต้องการให้ Thing1 เกิดบ่อยกว่า Thing2 เช่น Thing1 มีโอกาสเกิด 80% และ Thing2 มีโอกาสเกิด 20% ให้เปลี่ยนบรรทัดที่เลือก Prefab เป็น

```
// สุ่มเลข 0.0 - 1.0 ถ้าเหลือน้อยกว่า 0.8 ให้เลือก Thing1
GameObject prefabToSpawn = (rnd.NextFloat() < 0.8f) ?
prefabA : prefabB;
```

การสุ่มเกิดทุกเวลาที่กำหนด

ObjectSpawner.cs | Enemy2.cs | Enemy1.cs | UIManager.cs | PlayerActivity.cs | PlayerProperty.cs

Assembly-CSharp | ObjectSpawner | Update()

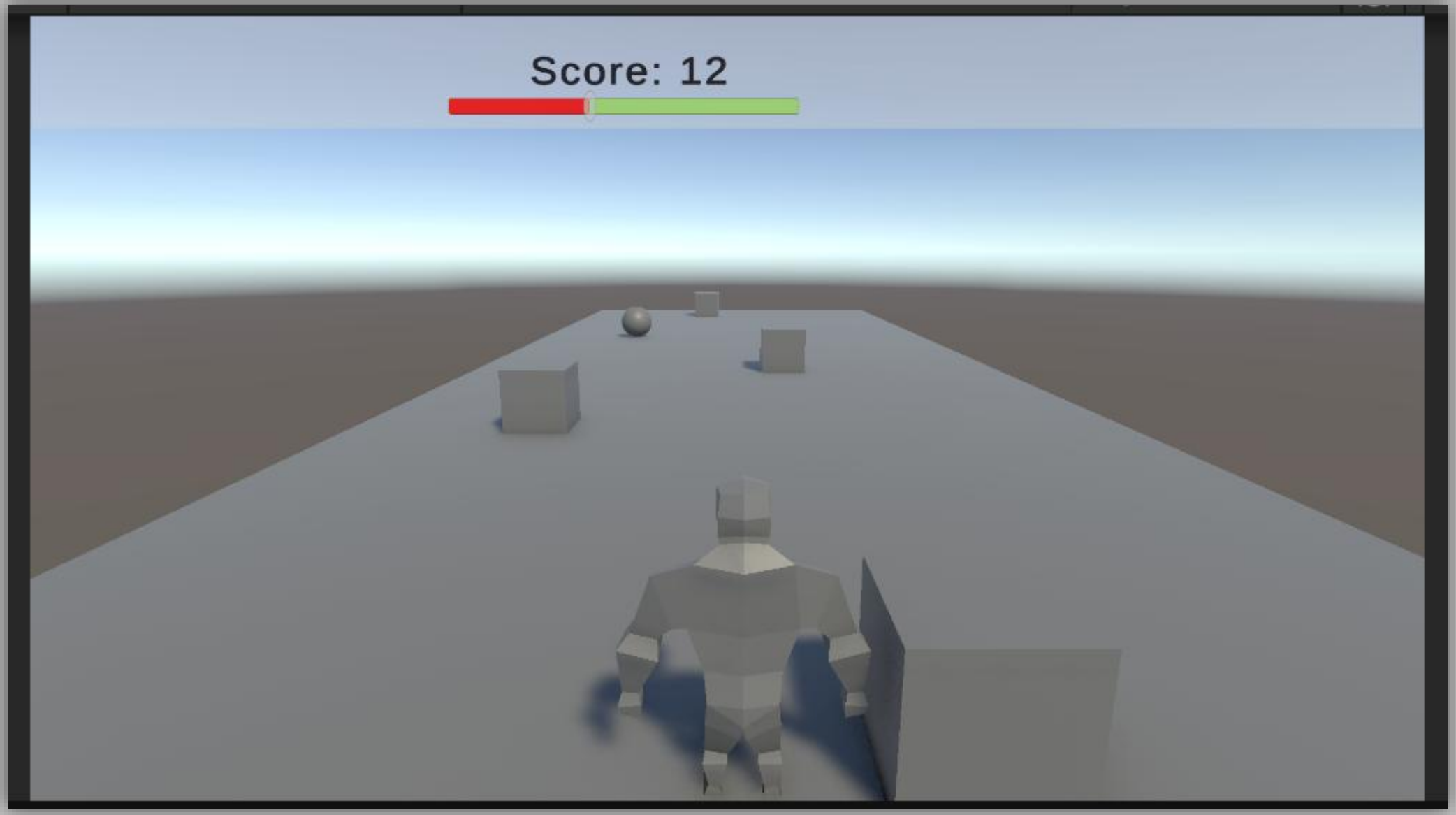
```
using UnityEngine;
using Unity.Mathematics;
using UnityEngine.InputSystem; // สำหรับการสุ่มประสิทธิภาพสูง

Unity Script (1 asset reference) | 0 references
public class ObjectSpawner : MonoBehaviour
{
    [Header("Prefabs")]
    public GameObject prefabA;
    public GameObject prefabB;

    [Header("Spawn Points")]
    public Transform[] spawnPoints; // ลาก S1, S2, S3, S4 มาใส่ที่นี่

    [Header("Timer Settings")]
    public float minTime = 0.2f;
    public float maxTime = 2.0f;

    private float timer;        // ตัวนับเวลาถอยหลัง
    private float nextInterval; // เวลาที่จะใช้สุ่มในรอบนั้นๆ

    private Unity.Mathematics.Random rnd;
```

```
22      void Awake()
23      {
24          // สร้างตัวสุ่มโดยใช้ Seed จากเวลาปัจจุบัน (ห้ามใช้ 0)
25          uint seed = (uint)System.DateTime.Now.Ticks;
26          rnd = new Unity.Mathematics.Random(seed == 0 ? 1 : seed);
27
28          // สุ่มเวลาสำหรับครั้งแรกสุด
29          SetNextInterval();
30      }
31
```

```
32      void SetNextInterval()
33      {
34          // สุ่มค่า n ระหว่าง 0.2 - 2.0
35          nextInterval = rnd.NextFloat(minTime, maxTime);
36      }
37
```

```
39      public void SpawnRandomObject()
40      {
41          if (spawnPoints.Length == 0)
42          {
43              return;
44          }
45          // 1. สุ่มเลือกจุดเกิด (S1 - S4)
46          // NextInt(min, max) -> ใน Mathematics รวมค่า max ด้วย ดังนั้นต้อง -1
47          int pointIndex = rnd.NextInt(0, spawnPoints.Length);
48          Transform selectedPoint = spawnPoints[pointIndex];
49          // 2. สุ่มเลือกว่าจะสร้างวัตถุไหน (A หรือ B)
50          // ใช้ NextBool() เพื่อสุ่ม 50/50 หรือ NextInt(0, 2)
51          GameObject prefabToSpawn = rnd.NextBool() ? prefabA : prefabB;
52          // 3. สร้างวัตถุ (Instantiate)
53          Instantiate(prefabToSpawn, selectedPoint.position, selectedPoint.rotation);
54          // Debug.Log($"Spawned {prefabToSpawn.name} at {selectedPoint.name}");
55      }
56
```

```
57      void Update()
58      {
59          // 1. สะสมเวลาที่ผ่านไปในแต่ละเฟรม
60          timer += Time.deltaTime;
61
62          // 2. เช็คว่าเวลาสะสมถึงกำหนดหรือยัง
63          if (timer >= nextInterval)
64          {
65              SpawnRandomObject();        // ทำการสร้างวัตถุ
66              SetNextInterval();  // สุ่มเวลาสำหรับรอบถัดไป
67              timer = 0;                  // รีเซ็ตตัวนับเวลา
68          }
69      }
70  }
```


Score: 12

Time.deltaTime

คือเวลาที่ใช้ในการวาดเฟรมล่าสุด การนำมาบวกสะสมใน timer จะทำให้เราได้เวลาที่เป็นวินาทีจริงๆ ไม่ว่าจะเล่นที่กี่ FPS ก็ตาม

SetNextInterval()

ฟังก์ชันนี้คือหัวใจสำคัญ มันจะถูกเรียกทุกครั้งหลังจากมีการสร้างวัตถุเสร็จ เพื่อกำหนด "เป้าหมายเวลา" (Target) ของรอบถัดไปให้ไม่ซ้ำกัน

ช่วงเวลา ท

ค่าจะถูกสุ่มใหม่เสมอในทุกๆ รอบการเกิด ทำให้ศัตรูหรือวัตถุไม่ได้ออกมาเป็นจังหวะที่คาดเดาได้ เช่น รอบแรกอาจจะรอ 0.5 วินาที รอบที่สองอาจจะรอ 1.8 วินาที เป็นต้น

ข้อแนะนำ

หากต้องการดูว่าเวลาเหลือเท่าไหร่ในหน้า Inspector คุณสามารถเปลี่ยน private float timer เป็น [SerializeField] private float timer เพื่อใช้ตรวจสอบการทำงานขณะรัน (Debugging) ได้

ข้อแนะนำ

หากสุ่มสร้างวัตถุถี่มาก เช่น 0.2 วินาทีต่อตัว เป็นต้น การใช้ Instantiate และ Destroy บ่อยๆ จะทำให้เครื่องกระตุกได้ แนะนำให้ใช้ Unity Engine.Pool (Object Pooling) ที่มีมาให้ใน Unity 6 เพื่อประหยัดทรัพยากร

อธิบาย Engine.Pool (Object Pooling

ใน Unity 6 การสร้างและทำลายวัตถุซ้ำๆ ด้วย Instantiate และ Destroy เช่น กระสุน หรือศัตรูที่เกิดบ่อยๆ เป็นต้น จะสร้างภาระให้หน่วยความจำและทำให้เกิด Lag Spikes จากการที่ Garbage Collector (GC) เข้ามาล้างข้อมูล

Object Pooling คือเทคนิค "ยืมและคืน" แทนที่จะทิ้งถังขยะ เราเลือกที่จะเก็บวัตถุที่ใช้เสร็จแล้วไว้ใน "บ่อ (Pool)" และนำกลับมาใช้ใหม่เมื่อต้องกา

ส่วนประกอบของ Unity Engine.Pool

Unity ให้คลาส ObjectPool<T> มาเพื่อจัดการเรื่องนี้โดยเฉพาะ โดยเราต้องกำหนด 4 ฟังก์ชันหลัก:

1. Create: สร้างวัตถุใหม่เมื่อในบ่อไม่มีของเหลือให้ยืม
2. OnGet: ตั้งค่าเมื่อเอาวัตถุออกจากบ่อ เช่น เปิดใช้งาน SetActive(true)
3. OnRelease: ตั้งค่าเมื่อเอาวัตถุกลับเข้าบ่อ เช่น ปิดการใช้งาน SetActive(false)
4. OnDestroy: ทำลายวัตถุทิ้งจริงๆ(เมื่อบ่อเต็มเกินไป

ตัวอย่างการสร้าง Pool สำหรับกระสุนหรือศัตรู

ตัวอย่างการนำ ObjectPool มาประยุกต์ใช้

```
1   using UnityEngine;
2   using UnityEngine.Pool; // ต้องใช้ Namespace นี้
3
4   public class EnemyPooler : MonoBehaviour
5   {
6       public GameObject enemyPrefab;
7       private IObjectPool<GameObject> pool;
8
9       [SerializeField] private bool usePool = true;
10      [SerializeField] private int defaultCapacity = 10;
11      [SerializeField] private int maxSize = 20;
12
```

```
13      void Awake()
14      {
15          // สร้าง Pool พร้อมกำหนดฟังก์ชันต่างๆ
16          pool = new ObjectPool<GameObject>(
17              CreateEnemy,        // ฟังก์ชันสร้าง
18              OnGetFromPool,      // ฟังก์ชันยืม
19              OnReleaseToPool,    // ฟังก์ชันคืน
20              OnDestroyObject,    // ฟังก์ชันทำลาย (ถ้าบ่อเต็ม)
21              collectionCheck: true,
22              defaultCapacity,
23              maxSize
24          );
25      }
26
```

```
27      // 1. สร้างวัตถุใหม่
28      private GameObject CreateEnemy()
29      {
30          GameObject go = Instantiate(enemyPrefab);
31          // เก็บ Reference ของ Pool ไว้ที่ตัว Object เพื่อให้มันส่งตัวเองกลับคืนบ่อได้
32          go.AddComponent<PooledObject>().myPool = pool;
33          return go;
34      }
35
```

```
36      // 2. เมื่อถูกดึงไปใช้
37      private void OnGetFromPool(GameObject go)
38      {
39          go.SetActive(true);
40      }
41
```

```
42      // 3. เมื่อถูกส่งคืนบ่อ
43      private void OnReleaseToPool(GameObject go)
44      {
45          go.SetActive(false);
46      }
47
```

```
48      // 4. เมื่อต้องทำลายจริงๆ
49      private void OnDestroyObject(GameObject go)
50      {
51          Destroy(go);
52      }
53
```

```
54      // วิธีเรียกใช้งานแทน Instantiate
55      public void Spawn()
56      {
57          GameObject enemy = pool.Get();
58          enemy.transform.position = transform.position;
59      }
60  }
```

วิธีทำให้ Object ส่งตัวเองกลับเข้า Pool

เราต้องสร้าง Script เล็กๆ แปะไว้ที่ตัว Prefab (เช่น ศัตรู) เพื่อให้มันรู้ว่าต้องกลับเข้าบ่อไหน แทนการสั่ง Destroy

```
1   public class PooledObject : MonoBehaviour
2   {
3       public IObjectPool<GameObject> myPool;
4
5       // แทนที่จะใช้ Destroy(gameObject) ให้เรียกฟังก์ชันนี้แทน
6       public void Deactivate()
7       {
8           myPool.Release(gameObject);
9       }
10
11      // ตัวอย่าง: ถ้าตกแมพ หรือชนผู้เล่น ให้คืนบ่อ
12      void OnCollisionEnter(Collision collision)
13      {
14          if (collision.gameObject.CompareTag("Player"))
15          {
16              Deactivate();
17          }
18      }
19  }
```

หมายเหตุ

ไม่ได้นำมาปรับเกมของเรา แต่เป็นเรื่องที่เอาไว้ให้ศึกษาเพิ่มเติมเพื่อเพิ่มประสิทธิภาพของเกมในอนาคต

Q&A