Lab 5 (5 tiết): Enemy (đối thủ)

MỤC ĐÍCH

Bài lab này giúp sinh viên tìm hiểu và xây dựng các Enemy (đối thủ).

YÊU CẦU

- Sinh viên phải làm xong Lab 4 trước khi làm lab này, dự án trong Lab 4 sẽ được dùng lại cho Lab 5 (dự án của Lab 4 đã để sẵn trong thư mục Lesson 5).

- Sinh viên đọc toàn bộ phần nội dung và thực hiện theo hướng dẫn.

- Thư mục đi kèm bài lab: Lesson 5

- Sau đó thực hiện các bài tập tương ứng.

NỘI DUNG

1. Giới thiệu

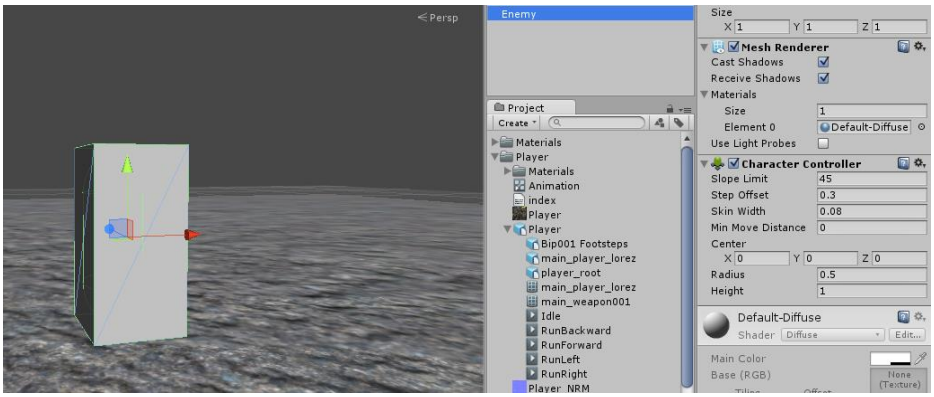
Trong bài lab này, tương tự như Lab 4, sinh viên sẽ tìm hiểu phương pháp tạo ra một đối thủ (Enemy) có khả năng tự di chuyển tới nhân vật.

Sử dụng lại dự án The Game đã tạo ở Lab 4 (trong trường hợp đã mất, sinh viên có thể sử dụng dự án trong thư mục The Game đi kèm trong thư mục Lesson 5).

Trong bài lab này, thuật ngữ Enemy được sử dụng (tương tự thuật ngữ Player), không dịch ra tiếng Việt vì ngữ nghĩa không đầy đủ.

2. Tạo đối tượng Enemy

Tạo một khối hộp (Cube), đặt tên là Enemy tại vị trí trên mặt đất, thiết lập Scale sao cho Enemy cao bằng Player, thêm vào khối hộp thành phần Character Controller.



Tạo một Script có tên là EnemyMovement và gắn vào Enemy, có nội dung như sau:

```
public class EnemyMovement : MonoBehaviour {

    CharacterController _controller; // Nhân vật
    Transform _player; // Tìm vị trí của Player

    void Start () {
     //Tìm nhân vật có Tag là Player, không tìm theo tên vì có 2 Player
     GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
     _player = playerGameObject.transform; // Lấy transform của nhân vật

      // Controller của Enemy
     _controller = GetComponent<CharacterController>();
    }

    void Update () {
     // hướng = vị trí người chơi - vị trí đối thủ
     Vector3 direction = _player.position - transform.position;
         // Đối thủ tự di chuyển tới nhân vật
         _controller.Move(direction * Time.deltaTime);
    }
}
```

Giải thích:

```
Vector3 direction = _player.position - transform.position;
 _controller.Move(direction * Time.deltaTime);
```

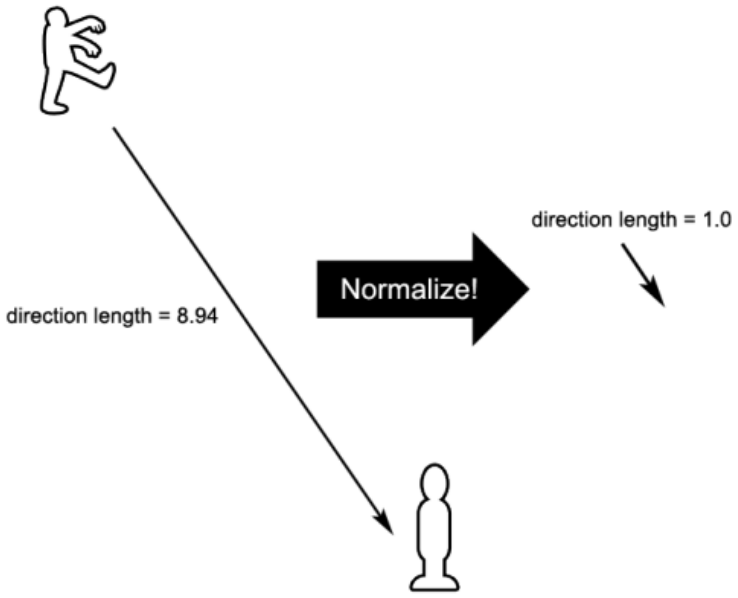
Vector `direction` lưu giữ khoảng cách giữa 2 đối tượng, mỗi khi 1 frame chạy thì Enemy sẽ cố gắng tiến lại gần Player theo khoảng cách đó. Cho tới khi lại gần nhau thì khoảng cách bằng 0 và Enemy sẽ không chuyển động.

Chọn Player cha trên Hierachy, gán tag là Player, nhấn Play để xem kết quả.

3. Normalize vector

Trong việc thực hiện Game, thay vì chuyển động từ A tới B, người ta sẽ dùng 1 vector AB để minh họa chuyển động, Vector này có độ dài bằng |AB| và có hướng từ A tới B. Khi đó thay vì chuyển động với độ dài AB thì người ta lấy 1 vector pháp tuyến (có độ dài bằng 1) nhân với tốc độ di chuyển.

Trong Unity, khái niệm chuyển động như trên, được gọi là **Normalize.** Nói cách khác, *Normalize là quá trình chuyển một vector về vector có hướng như ban đầu nhưng độ dài bằng 1*.



Trong Unity, công việc đó do một phương thức thực hiện, đó là phương thức Normalize():

Vector3 direction = _player.position - transform.position;

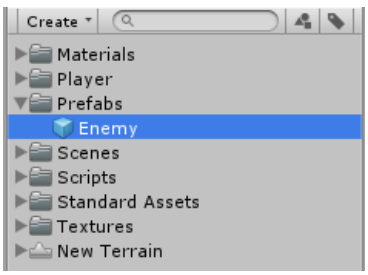
direction.Normalize();

Áp dụng phương pháp Normalize, ta có thể dễ dàng thêm vào thuộc tính tốc độ chuyển động để dễ điều khiển như sau:

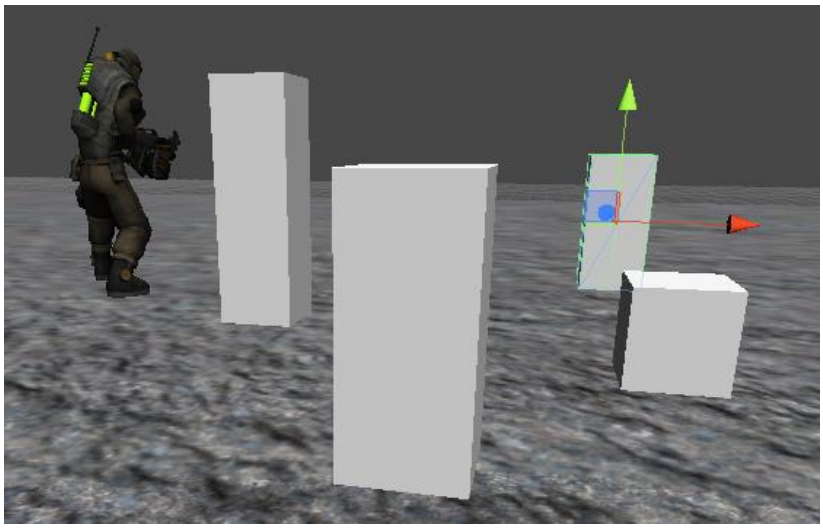
```
public class EnemyMovement : MonoBehaviour {

    CharacterController _controller; // Nhân vật
    Transform _player; // Tìm vị trí của Player
    // Biến tốc độ di chuyển
    public float _moveSpeed = 5.0f;


    void Start () {
     //Tìm nhân vật có Tag là Player, không tìm theo tên vì có 2 Player
     GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
     _player = playerGameObject.transform; // Lấy transform của nhân vật

      // Controller của Enemy
     _controller = GetComponent<CharacterController>();
    }

    void Update () {
     // hướng = vị trí người chơi - vị trí đối thủ
     Vector3 direction = _player.position - transform.position;
     direction.Normalize(); // Chuyển về vector cùng phương, có độ lớn = 1
     // Tăng tốc
     Vector3 velocity = direction * _moveSpeed;
     // Di chuyển theo tốc độ
     _controller.Move(velocity * Time.deltaTime);
    }
}
```

4. Prefab cho Enemy

Tạo một thư mục có tên Prefabs, kéo đối tượng Enemy vào thư mục vừa tạo, ta được một Enemy dự trữ cho các đối tượng địch thủ sau này.



Từ một Enemy trong Prefab, kéo thêm khoảng 03 Enemy ra màn hình game, thay đổi kích thước, tốc độ di chuyển của các Enemy này.



Nhấn Play để xem kết quả.

5. Thêm giá trị Health (năng lượng) cho Enemy

Để Enemy có năng lượng và bị giảm năng lượng khi bị bắn, cần khai báo 2 biến kiểu int: một miêu tả năng lượng tối đa, một miêu tả năng lượng hiện hành.

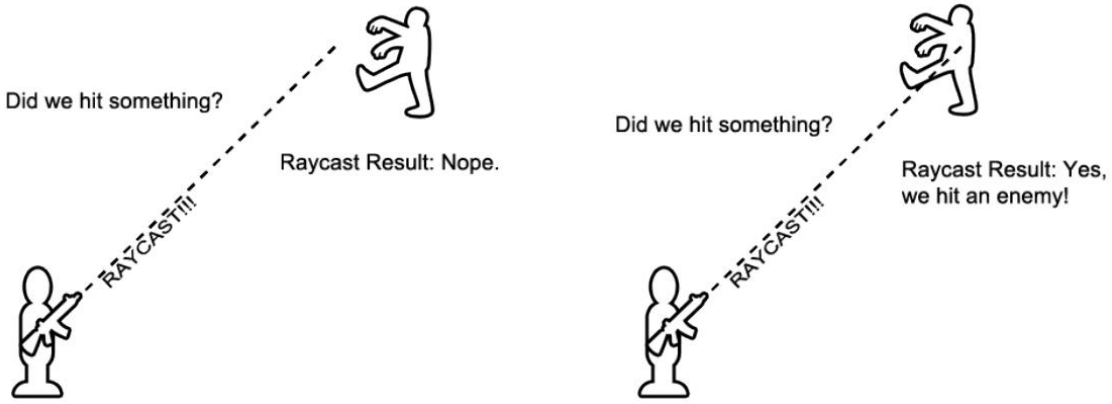
Tạo Script có tên Health có nội dung như sau:

```
public class Health : MonoBehaviour {

    public int  _maximumHealth = 100; // Năng lượng cực đại
    int _currentHealth = 0; // Năng lượng hiện hành
    void Start () {
        // Gán năng lượng hiện hành bằng cực đại
        _currentHealth = _maximumHealth;
    }
    // Phương thức hủy đối tượng game khi bị bắn
    public void Damage(int damageValue)
    {
        // mỗi lần bị bắn, _currentHealth bị sụt giảm
        _currentHealth -= damageValue;
        if (_currentHealth <= 0) // Nếu _currentHealth <=0
        {
            Destroy(gameObject); // thì hủy đối tượng
        }
    }
}
```

Chọn Enemy trong Prefab, gán script Health cho Enemy đó, lúc này các Enemy trên Scene cũng đều bị điều khiển bởi Script này.

6. Bắn đạn sử dụng Raycasts

Raycasting là kỹ thuật vẽ một tia vô hình, từ một điểm vào không gian, dùng để xác định sự va chạm (có thể) với bất kỳ đối tượng khác. Như vậy, ta có thể vẽ một đường thẳng từ họng súng người chơi và hướng về trước, nếu va chạm với Enemy, ta gọi phương thức Damage của Enemy.



Tạo Script có tên RifleWeapon, có nội dung như sau:

```
public class RifleWeapon : MonoBehaviour {

    public int _damageDealt = 50;
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        // Người dùng nhấn chuột trái (phím Fire1 với máy chơi game)
        if (Input.GetButtonDown("Fire1"))
        {
            // Vẽ một tia vô hình bắt đầu từ giữa Camera
            // Vị trí (0.5, 0.5, 0) thể hiện giữa màn hình camera với trục X, Y.
            Ray mouseRay = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
            RaycastHit hitInfo; // Lưu giữ thông tin va chạm
            if (Physics.Raycast(mouseRay, out hitInfo)) // Nếu có giao cắt
            {
                // Lấy class Health từ thành phần của đối tượng khi có giao điểm
                Health enemyHealth = hitInfo.transform.GetComponent<Health>();
                if (enemyHealth != null) // Nếu có thành phần đó
                {
                    // Gọi phương thức hủy
                    enemyHealth.Damage(_damageDealt);
                }
            }
        }
    }
}
```

Gán Script này vào đối tượng Player (cha), chạy game: mỗi khi có Enemy gần tới thì đưa ống ngắm vào nhà nhấn trái chuột 2 lần để làm biến mất Enemy.

7. Khóa và mở khóa chuột

Khi chạy game, muốn cho chuột bị khóa (không hiển thị), ta thực hiện code như sau trong lớp RifleWeapon, tại phương thức Start:

```
void Start () {
    Screen.lockCursor = true; // Khóa chuột
}
```

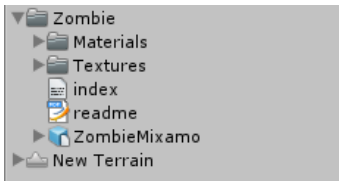
Khi nhấn phím Esc thì mở chuột lại như cũ, thêm code trong phương thức Update:

```
// Update is called once per frame
void Update () {
    if (Input.GetKey(KeyCode.Escape))
    {
        Screen.lockCursor = false;
    }

    // Người dùng nhấn chuột trái (phím Fire1 với máy chơi game)
    if (Input.GetButtonDown("Fire1"))
    {
        Screen.lockCursor = true;
```

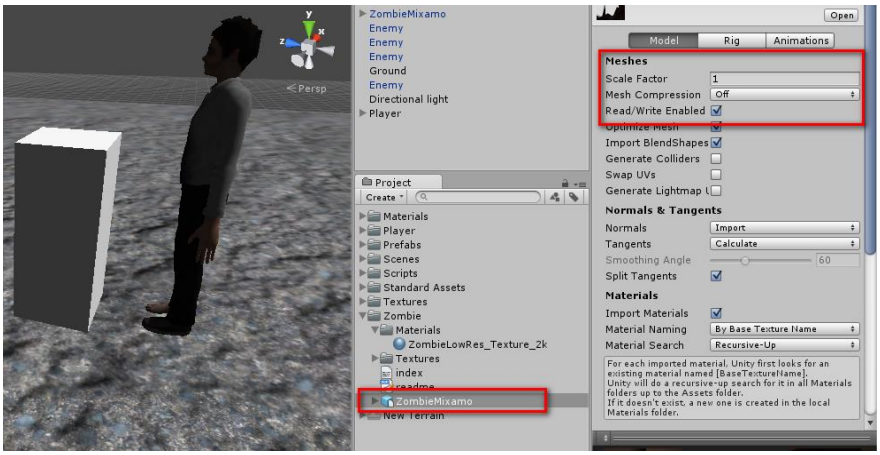
Chạy và xem kết quả.

8. Thêm mô hình Zombie

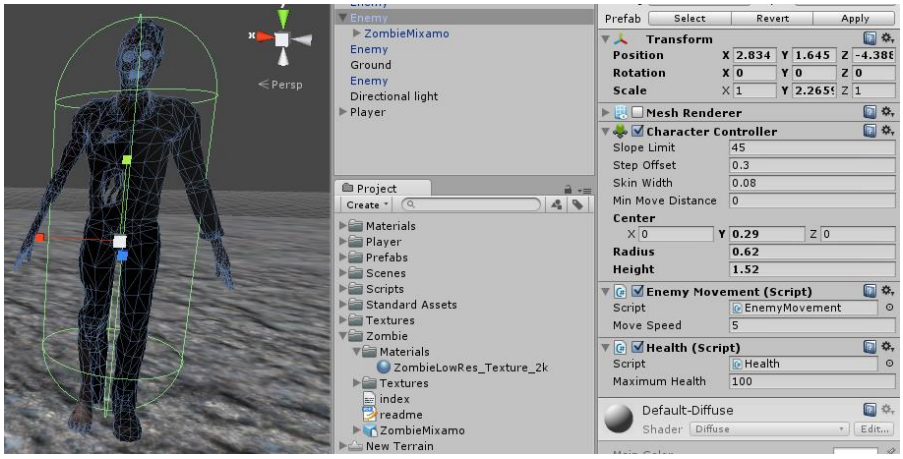
Trong thư mục Lesson 5, tìm tới thư mục Zombie, kéo vào dự án.



Click chọn mô hình ZombieMixamo, trên Inspector chọn Model, thay đổi tham số Scale Factor từ 0.1 lên thành 1 (mục đích của việc này là giúp cho độ lớn của Zombie bằng độ lớn của Player), kéo Zombie này ra Scense gần một Enemy nào đó.



Trên Hierachy, chọn 1 Enemy, trên Inspector bỏ đi (Remove Component) các thành phần như Box Collider, Mesh Fillter, Mesh Renderer (uncheck). Chuyển ZombieMixamo thành con của Enemy này, thay đổi tham số Position để Zombie đứng trên mặt đất. Chọn tiếp Enemy, thay đổi các tham số Center, Radius và Height cho phù hợp (chi tiết xem lại mục 10 của Lab 4, *đã hướng dẫn đưa Player thay Capsule*).



Nhấn Apply trên Inspector để áp dụng cho toàn bộ các Enemy khác. Nhấn Play để xem các Zombie này di chuyển theo nhân vật Player.

9. Gán trọng lực cho nhân vật và hướng mặt nhân vật vào Player

Khi chạy game, ta thấy rằng các nhân Zombie này chưa đứng trên mặt đất và mặt chưa quay về nhân vật. Ta có thể thực hiện điều này bằng cách thay đổi code trong script Enemy Movement. Mở Script này ra, thay đổi code như sau:

Trong lớp EnemyMovement, thêm các tham số về trọng lực và gia tốc chiều y:

```
// Khai báo thêm biến trọng lực và biến phụ y_Velocity
public float _gravity = 2.0f;
 float _yVelocity = 0.0f;
```

Trong phương thức Update, thêm code như sau:

```
void Update () {
 // hướng = vị trí người chơi - vị trí đối thủ
 Vector3 direction = _player.position - transform.position;
 direction.Normalize(); // Chuyển về vector cùng phương, có độ lớn = 1
 // Tăng tốc
 Vector3 velocity = direction * _moveSpeed;
    // Nếu khác mặt đất
 if (!_controller.isGrounded)
    {
      // chiều y sẽ giảm xuống
         _yVelocity -= _gravity;
    }
      velocity.y = _yVelocity; // gán lại vị trí chiều y
 // Di chuyển theo tốc độ
 _controller.Move(velocity * Time.deltaTime);
}
```

Để Enemy luôn quay mặt vào Player, thêm đoạn code sau vào trong phương thức Update:

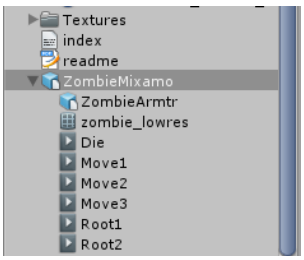
```
velocity.y = _yVelocity; // gán lại vị trí chiều y

direction.y = 0; // gán hướng chiều y = 0 (không dùng hướng y)
// gán rotation của Enemy quay theo hướng Player
transform.rotation = Quaternion.LookRotation(direction);

// Di chuyển theo tốc độ
_controller.Move(velocity * Time.deltaTime);
```

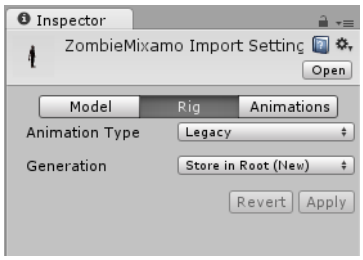
Trong phần này có phương thức LookRotation cho phép quay đối tượng theo một hướng nào đó, ở đây ta gán hướng y (lên trên) bằng không nên Enemy sẽ không quay theo hướng này.

Nhấn Play để xem kết quả:

10. Làm cho nhân vật chuyển động

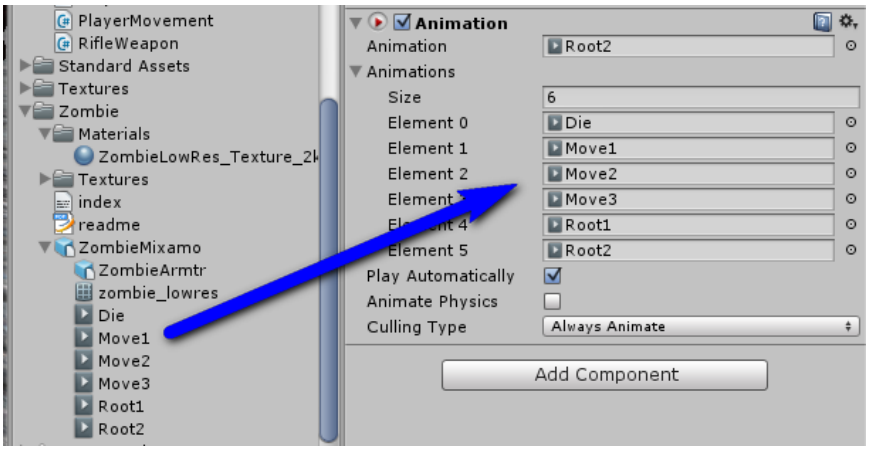
Chọn ZombieMixamo prefab, trên Inspector chọn thẻ Rig, chọn về chế độ Legacy, nhấn Apply.





Chọn thẻ Animation, ta thấy Zombie đã có sẵn một số hình thức chuyển động như Move, Die, Root…

Chọn một Zombie bất kỳ trên Hierachy, thêm một thành phần Animation, cấu hình Animations có size = 6, mở rộng ZombieMixamo prefab, kéo các Clip sẵn có vào Animation như hình sau:



Nhấn Apply để áp dụng cho toàn bộ các Zombie khác.

Tạo mới một tập tin Script có tên là EnemyAnimation, gắn vào một Enemy bất kỳ, nội dung Script như sau:

```
public class EnemyAnimation : MonoBehaviour {

    // Tạo một Animation
    Animation _animation;
    void Start () {
        // Lấy thành phần Animation từ Zombie, do script gắn vào Enemy
      _animation = GetComponentInChildren<Animation>();
        // Chế độ lặp lại
         _animation["Move1"].wrapMode = WrapMode.Loop;
        // Chạy clip này
         _animation.Play("Move1");
        // Xác định thời điểm Move1 bắt đầu thực hiện
         _animation["Move1"].normalizedTime = Random.value;
    }

    // Update is called once per frame
    void Update () {

    }
}
```

Trong script này phương thức Random.value sẽ phát sinh một số ngẫu nhiên trong khoảng từ 0 đến 1. Khi đó normalizedTime sẽ điều khiển điểm thực thi của Animation (Ví dụ: điểm thực thi là bắt đầu khi random ra 0.0, điểm thực thi là ở giữa khi random ra số 0.5 và điểm thực thi là cuối khi random ra số 0.9…)

Nhấn Play để xem kết quả: các Enemy di chuyển "bám" theo Player.

Để giúp sinh động cho các Enemy, ta có thể làm cho các Enemy di chuyển theo các cách khác nhau dựa trên các loại chuyển động như "Move1", "Move2", "Move3". Thay đổi code như sau trong lớp EnemyAnimation:

```
void Start () {
    // Lấy thành phần Animation từ Zombie, do script gắn vào Enemy
  _animation = GetComponentInChildren<Animation>();

  string animationToPlay = ""; // biến điều khiển animation
  switch (Random.Range(0, 3)) // Lấy ngẫu nhiên 1 giá trị từ 1 đến 3
  {
      default:
         case 0:
         animationToPlay = "Move1";
         break;
     case 1:
         animationToPlay = "Move2";
         break;
     case 2:
         animationToPlay = "Move3";
         break;
  }
    // Chế độ lặp lại
  _animation[animationToPlay].wrapMode = WrapMode.Loop;
    // Chạy clip này
  _animation.Play(animationToPlay);
    // Xác định thời điểm Move1 bắt đầu thực hiện
  _animation[animationToPlay].normalizedTime = Random.value;
}
```

11. Enemy bị chết khi bắn

Để Enemy bị chết khi bắn, trong Script Heath, phương thức Damage, thay đổi lại như sau:

```
// Phương thức hủy đối tượng game khi bị bắn
public void Damage(int damageValue)
{
    // mỗi lần bị bắn, _currentHealth bị sụt giảm
    _currentHealth -= damageValue;
    if (_currentHealth <= 0) // Nếu _currentHealth <=0
    {
        // Chế độ lặp lại
        _animation["Die"].wrapMode = WrapMode.Once;
        // Chạy clip này
        _animation.Play("Die");
        // Xác định thời điểm Move1 bắt đầu thực hiện
        _animation["Die"].normalizedTime = Random.value;
       Destroy(gameObject,2.6f); // thì hủy đối tượng

    }
```

Lưu ý: _thuộc tính *_animation* được thực hiện như trong lớp *EnemyAnimation*. Phương thức Destroy(gameObject, 2.6) sẽ hủy Enemy sau 2.6s, bằng thời gian chạy animation "Die".

BÀI TẬP

Bài 1.

Thiết lập vị trí của các Enemy ở các điểm khác nhau, có tốc độ di chuyển khác nhau và cùng tiến tới Player.

Bài 2.

Tạo phương pháp phát sinh nhiều Enemy:

- Tạo một đối tượng empty tên là EnemySpawnManager.
- Tạo một script có tên EnemySpawnManager có nội dung như sau:

```
public class EnemySpawnManager : MonoBehaviour
{
    public GameObject _enemyToSpawn;

    public float _spawnDelay = 1.0f;

    float _nextSpawnTime = -1.0f;

    void Update()
    {
        if (Time.time >= _nextSpawnTime)
        {
            Instantiate(_enemyToSpawn);
            _nextSpawnTime = Time.time + _spawnDelay;
        }
    }
}
```

- Gán Script vừa tạo vào đối tượng EnemySpawnManager, trên Inspector, kéo Enemy vào phần Enemy To Spawn.
- Nhấn Play để xem kết quả. Sinh viên tự tìm hiểu và giải thích code

Bài 3. Điều chỉnh code như sau trong hàm Update để các Enemy nhảy xuống từ rìa màn hình:

```
void Update()
{
    if (Time.time >= _nextSpawnTime)
    {
        Vector3 edgeOfScreen = new Vector3(1.0f, 0.5f, 8.0f);
        Vector3 placeToSpawn = Camera.main.ViewportToWorldPoint(edgeOfScreen);
        Quaternion directionToFace = Quaternion.identity;
        Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
        _nextSpawnTime = Time.time + _spawnDelay;
    }
}
```