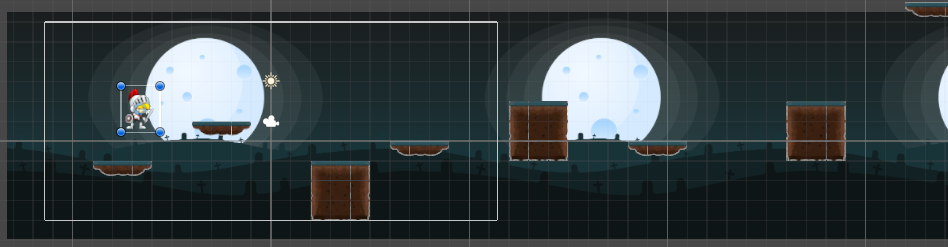
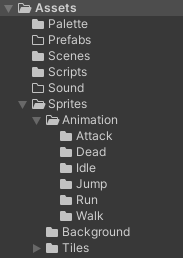
1. Animation của sprite (dự án jumping)

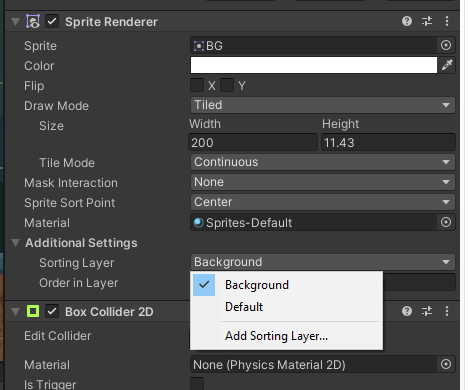
* Tạo project 2D kiểu như thế này



* Asset free lấy ở đây: <https://www.gameart2d.com/freebies.html> chọn một package bất kỳ để thiết kế
* Tạo cấu trúc thư mục trong project như sau:



* Level design sử dụng tilemap, background nhiều lớp có hiệu ứng parallax, Tạo mới order layer có tên là background rồi kéo lên phía trên layer Default, sau đó gán layer đó cho những hình ảnh background để bảo đảm chúng luôn nằm dưới khi render



**Tạo animation cho player**

* Player lấy sprite idle đưa vào Sprite Renderer để làm hình ảnh ban đầu cho player, add collider, rigidbody 2d cho player
* Add script PlayerController để điều khiển player di chuyển và nhảy như sau

public class PlayerController : MonoBehaviour

{

public float moveSpeed;

public float jumpForce;

public float radius;

Rigidbody2D rb;

private bool m\_FacingRight = true;

float horizontalInput;

float verticalInput;

[SerializeField] private Transform groundCheck;

[SerializeField] private LayerMask layerMaskGround;

private bool isGrounded =false; // kiem tra xem player co dang dung tren ground hay khong

private void OnDrawGizmos() // vẽ vòng tròn từ tâm của vị trí groundcheck trên player, tương ứng với Physics2D.OverlapCircle

{

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(groundCheck.transform.position, radius);

}

// Start is called before the first frame update

void Start()

{

rb = GetComponent<Rigidbody2D>();

}

// Update is called once per frame

void Update()

{

horizontalInput = Input.GetAxis("Horizontal");

verticalInput = Input.GetAxis("Vertical");

}

bool GroundCheck()

{

Collider2D hit = Physics2D.OverlapCircle(groundCheck.transform.position, radius, layerMaskGround);

if(hit != null) return true;

return false;

}

private void FixedUpdate()

{

rb.velocity = new Vector2(horizontalInput \* moveSpeed, rb.velocity.y);

isGrounded = GroundCheck();

// Kiểm tra xem nếu player đang đứng dưới ground và người chơi nhấn phím W

if (isGrounded && Input.GetKey(KeyCode.W))

{

// Add a vertical force to the player.

isGrounded = false;

rb.AddForce(new Vector2(0f, jumpForce), ForceMode2D.Impulse);

}

// điểu khiển player quay sang trái hoặc sang phải theo chiều nhấn

if (horizontalInput > 0 && !m\_FacingRight) Flip();

else if (horizontalInput < 0 && m\_FacingRight) Flip();

}

private void Flip()

{

// Switch the way the player is labelled as facing.

m\_FacingRight = !m\_FacingRight;

// Multiply the player's x local scale by -1.

Vector3 theScale = transform.localScale;

theScale.x \*= -1;

transform.localScale = theScale;

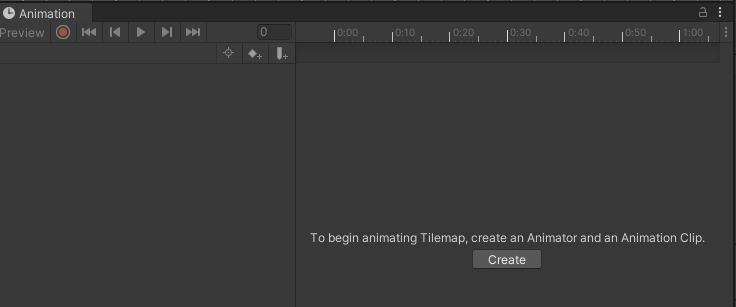
}

}

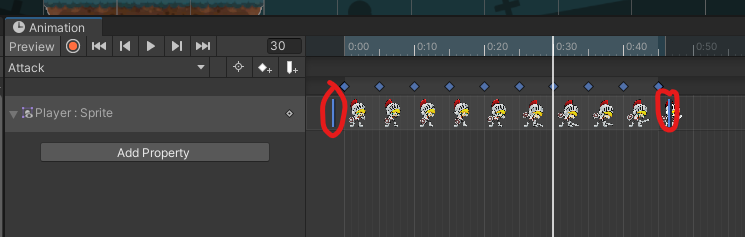
* Tạo layermask để gán cho những đối tượng cần kiểm tra va chạm bằng Physic2d, có thể là raycast, có thể là OverlapCircle..

**Tạo các animation cho player**

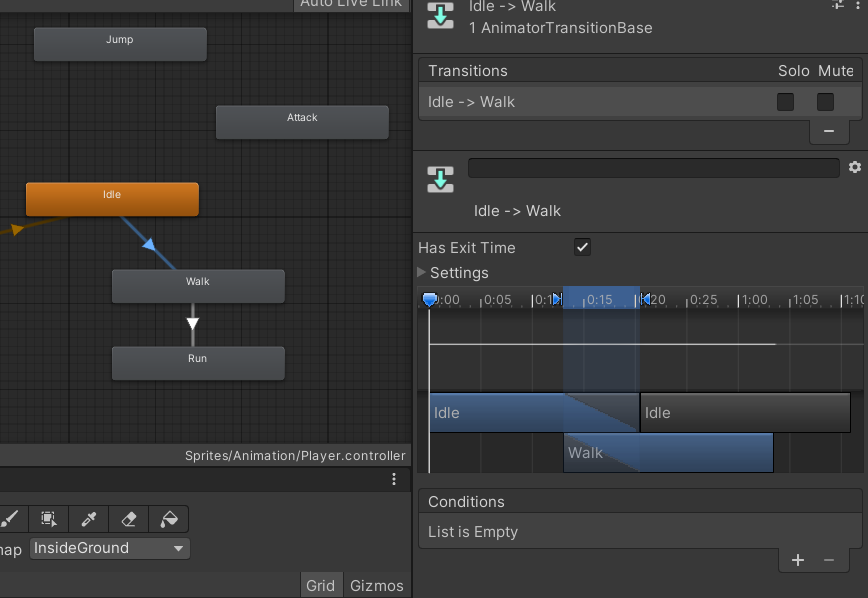
* Click vào player, mở cửa sổ animation, click vào button create để tạo animation đầu tiên, đồng thời cũng tạo luôn animator và tự động gán vào component cho player



* Tạo animation bằng các kéo các sprite để tạo thành chuyển động vào trong cửa sổ frame của animation. Muốn tăng giảm tốc độ chuyển động của nhân vật thì ta thay đổi thời gian giữa mỗi frame(càng cao thì animation càng chậm). Chọn toàn bộ các key và click vào thanh dọc như hình dưới và kéo để thay đổi thời gian

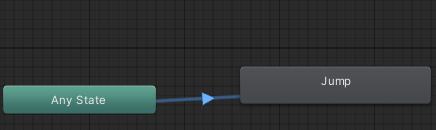


* Mở animaor tạo liên kết giữa các các trạng thái tương ứng với các animation vừa tạo, mặc định từ node entry có liên kết tới animation đầu tiên được tạo là Idle, nếu muốn chuyển sang animation khác thì từ anim được chọn, click phải 🡪 make transition đến anim mới. Mỗi transition sẽ có một số thuộc tính sau:



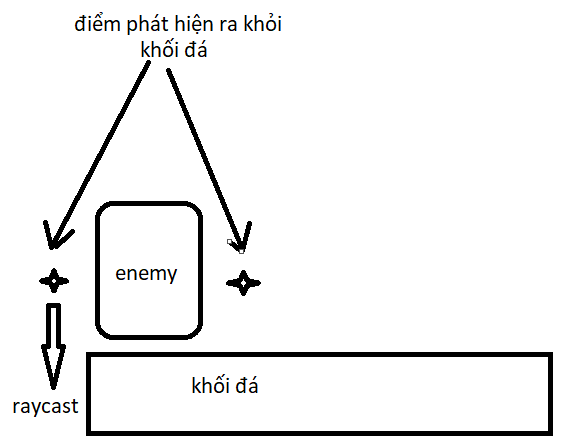
1. **From State (Trạng thái ban đầu):** Trạng thái mà transition bắt đầu từ đó.
2. **To State (Trạng thái kết thúc):** Trạng thái mà transition kết thúc tại đó.
3. **Conditions (Điều kiện):** Điều kiện để transition được kích hoạt và diễn ra. Các điều kiện này có thể là các tham số Animator hoặc triggger.
4. **Has Exit Time (Có thời gian kết thúc):** Xác định liệu transition có chờ đợi thời gian kết thúc của trạng thái hiện tại hay không.
5. **Exit Time (Thời gian kết thúc):** Thời gian mà transition sẽ kích hoạt sau khi trạng thái hiện tại bắt đầu.
6. **Transition Duration (Thời gian chuyển đổi):** Thời gian mà transition sẽ mất để chuyển từ trạng thái ban đầu đến trạng thái kết thúc.
7. **Offset (Điều chỉnh):** Điều chỉnh thời gian kích hoạt của transition.
8. **Motion (Chuyển động):** Các chuyển động được áp dụng trong quá trình transition.
9. **Settings (Thiết lập):** Các thiết lập khác nhau cho transition, bao gồm cả thiết lập về transition vật lý.

* Để việc chuyển giữa các anim diễn ra tức thì thì ta bỏ check của has exit time và trong setting 🡪 set transition duration = 0
* Để điều khiển lúc nào có transtion thì ta tạo parameter để đưa vào kiểm tra trong condition của transition đó
* Với chuyển động nhảy, là một chuyển động có thể xuất phát từ nhiều trạng thái khác nhau, đang idle hay walk, run thì đều có thể nhảy, do đó ta không tạo transtition từ trong mỗi state đó mà ta có node any state để làm việc đó



Tạo Enemy và điều khiển enemy di chuyển trong game

* Tạo enemy tương tự như player, tạo prefab cho enemy
* Tạo các điểm có thể sinh enemy ở trong game, các điểm đó sẽ được thiết kế sẵn nằm trên các khối đá, tất cả điểm đó sẽ nằm trong 1 parent spawnEnemyGroup ở tọa độ 0,0,0. Viết script lấy tọa độ các điểm spawn là các child trong group để sinh enemy
* Điều khiển enemy di chuyển trên các khối đá, mỗi enemy sẽ có 2 điểm ở bên trái và phải để nhận biết là mình có di chuyển ra khỏi vị trí khối đá hay không, nếu raycast từ 2 điểm đó chiếu xuống không chạm vào collider của khối đá thì tức là nó đã ra tới mép của khối đá, cần di chuyển theo hướng ngược lại



* Để kiểm tra player đã tới gần enemy hay chưa, thì mỗi enemy sẽ có 1 raycast hướng ra phía ngoài hướng di chuyển để phát hiện
* Độ dài raycast phát hiện player, tốc độ di chuyển của mỗi enemy sử dụng các biến nhận giá trị random để tạo sự đa dạng trong game

Hiệu ứng paralax (game jumping, game endless run)

**Animation của third person (làm demo một project third person)**

* Tạo mới scene 3d
* Import 3d model character vào, kéo file fbx vào asset, thực hiện extract texture và extract material nếu cần
* Scale nhân vật đúng tỉ lệ, đặt model vào trong gameobject Player
* Thiết kế môi trường đơn giản, có thể dùng terrain
* Viết script điều khiển player di chuyển như sau:

public class Player3dController : MonoBehaviour

{

[SerializeField] float moveSpeed = 2f;

[SerializeField] float runSpeed = 5f;

[SerializeField] float rotationSpeed = 5f;

public Animator animator;

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

float horizontalInput = Input.GetAxis("Horizontal");

float verticalInput = Input.GetAxis("Vertical");

var moveDirection = (new Vector3(horizontalInput, 0, verticalInput)).normalized;

if (Input.GetKey(KeyCode.LeftShift))

{

transform.position += moveDirection \* runSpeed \* Time.deltaTime;

}

else transform.position += moveDirection \* moveSpeed \* Time.deltaTime;

if (moveDirection != Vector3.zero)

{

Quaternion toRotation = Quaternion.LookRotation(moveDirection, Vector3.up);

transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed\*Time.deltaTime);

}

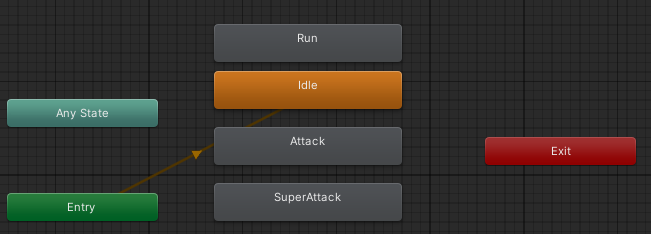
}

}

* Tạo animation cho Player: đầu tiên lấy animation đã được thiết kế sẵn trong model 3D, click vào từng asset model sau đó click vào animtion trong model đó rồi nhấn Ctrl + D để đưa animation đó ra ngoài, sau đó đặt tên lại các animation và được chúng vào hết trong một thư mục animation tạo trong thư mục asset

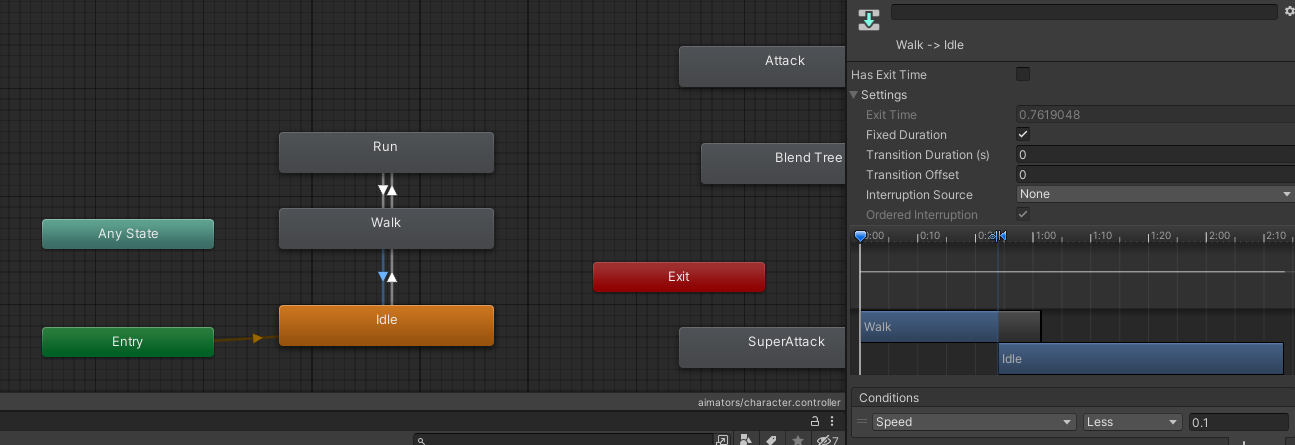


* Tạo animator controller sau đó gắn cho Model nằm trong gameobject Player
* Mở animator lên mà kéo lần lượt các animation vào cửa sổ thiết kế



* Tạo transition giữa các state để điều khiển chuyển động, tạo parameter Speed để làm điều kiện thực thi transition, idle qua wlk thì speed < 0.1, walk qua run thì speed < 1.1.

Với speed là giá trị nhận từ input khi nhấn phím.



* Ta edit hàm update như sau

void Update()

{

float horizontalInput = Input.GetAxis("Horizontal");

float verticalInput = Input.GetAxis("Vertical");

**float moveAmount = Mathf.Clamp01(Mathf.Abs(horizontalInput) + Mathf.Abs(verticalInput));**

var moveDirection = (new Vector3(horizontalInput, 0, verticalInput)).normalized;

if (Input.GetKey(KeyCode.LeftShift) && moveDirection != Vector3.zero)

{

**moveAmount += 1;**

transform.position += moveDirection \* runSpeed \* Time.deltaTime;

}

else transform.position += moveDirection \* moveSpeed \* Time.deltaTime;

**animator.SetFloat("Speed", moveAmount);**

if (moveDirection != Vector3.zero)

{

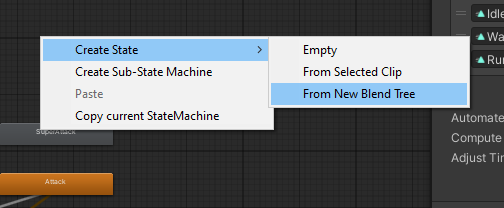
Quaternion toRotation = Quaternion.LookRotation(moveDirection, Vector3.up);

transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed\*Time.deltaTime);

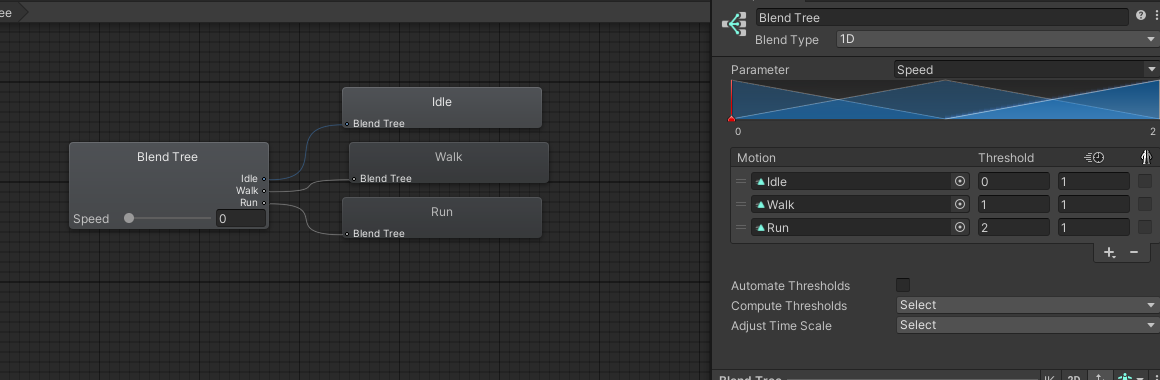
}

}

* Tuy nhiên hiệu ứng chưa được mượt mà vì chưa có sự chuyển nhanh dần và chậm dần giữa các trạng thái, do đó trong animtor controller ta dùng animation blend để tạo sự pha trộn giữa các animation. Tạo mới blend tree như sau



* Đổi tên condition thành Speed, lần lượt click + để add các motion,kéo các animation vào 3 motion lần lượt là idle, walk và run, bỏ chọn automate thresholds và nhập các giá trị vào như hình dưới



* Xóa các state cidle, run, walk trong animtor và thay thế bằng blend tree mới tạo, chúng ta sẽ thấy chuyển động của nhân vật trở nên mượt mà hơn

**Xữ lý camera follow cho third person character**

Animation của UI (zoom in out, fade in out) Demo ứng dụng Dental

Chuyển động của các đối tượng khác trong game (rơi nhờ gravity), change position = script