PEC 3 – Animación de robots

Joaquín López Soriano

Media para videojuegos

Universidad Oberta de Cataluña

Contenido

Resumen de las entregas 3
Visión de futuro 7
Explicación sobre los Animator controllers 8
Resumen de la creación de las animaciones 10
Enlace del proyecto 10
Bibliografía 11
Software 11
Exportar animaciones 11
Realizar animaciones con Maya 11
Asset 11
Movimiento y rotación 11
Terminología 11

Resumen de las entregas

El desarrollo incremental de las diferentes prácticas para entender como funciona el proceso de creación y animación de personajes 3D ha sido adecuado.

La evolución desde una escena vacía en Maya hasta ver tus personajes animados interactuando resulta reconfortante, más todavía tras todos los problemas superados durante cada práctica.

Respecto a esta práctica, para animar a cada robot se ha continuado usando la aplicación Maya.
Al principio, la realización de la animación ha resultado complicado hasta entender como se realiza; ubicar el marcador de frame en un frame, modificar las propiedades de las articulaciones del esqueleto y grabar el nuevo estado con la tecla *'S'*.

El animar los robots no ha sido una tarea fácil porque al mover las articulaciones dejaban huecos entre los elementos que forman el objeto 3D u obtenido un objeto 3D no muy verosímil, restringiendo y delimitando la animación de los mismos.

Para realizar el proyecto, se ha recreado el escenario de una ciudad donde dos mecha luchan.
Para ello, se ha usado el asset de Unity CITY package, que ofrece gran cantidad de objetos para construir una cuidad.

La cámara se ha posicionado de forma frontal, y situando a cada robot enfrentado el uno contra el otro como en los juegos de lucha.

Se ha agregado una música de fondo y un sonido para cada acción.

Tras ello, se ha implementado el animator controller, explicando en un punto posterior.

Por último, se ha definido el movimiento de cada robot. Para ello, se ha usado el componente *Character controller*, para definir el movimiento y rotación de cada robot. Para la captura de teclado, se ha usado el *asset* de *Unity Input System*. Se ha creado un manejador para controlar a cada robot.

Los controles de cada robot son:

Tecla	Robot1	Robot2
Movimiento	WASD	Fecha arriba, flecha izquierda, flecha abajo y flecha derecha
Ataque	F	Teclado numérico 2
Defensa	G	Teclado numérico 3

Para su implementación, es decir, para programar el movimiento y la animación, se ha programado el *script PlayerController*. A destacar que la rotación del robot se realiza en la dirección del movimiento, por lo que siempre está mirando de frente al movimiento que realiza.

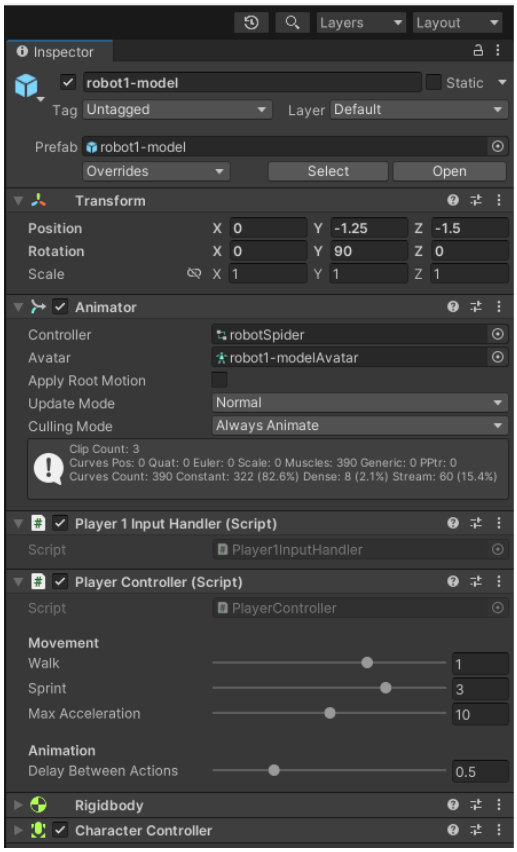
El código del *script* se muestra a continuación:

```
public class PlayerController : MonoBehaviour
{
    // Fields
    [Header("Movement")]
    [Tooltip("The speed at which the npc walks")] [SerializeField]
[Range(0, 1.5f)]
    protected float walk = 1f;

    [Tooltip("The speed at which the npc runs")] [SerializeField]
[Range(0, 4)]
    protected float sprint = 3f;

    [Tooltip("Max speed of movement transitions")] [Range(0, 20)]
    [SerializeField] private float maxAcceleration = 10f;


    [Header("Animation")]
    [Tooltip("Minimum duration between two actions")][Range(0, 2)]
    [SerializeField] private float delayBetweenActions = 0.5f;

    // script classes
    private Animator _animator;
    private CharacterController _controller;
    private PlayerInputHandler _inputHandler;

    //variables
    private float _speed;
    private float _movementSpeed;       // max speed
    private float _accelerationSpeed;   // max acceleration
    private float _animationBlend;      // speed of the animation
    private Vector3 _destinationPoint;  // point to go to
    private const float RotationVelocity = 360;
    private float _lastTimeAction;

    private void Awake()
    {
        // set components on the gameObject
        _animator = GetComponent<Animator>();
        _controller = GetComponent<CharacterController>();
        _inputHandler = GetComponent<PlayerInputHandler>();
        // max speed between the walk and the sprint value
        _movementSpeed = Random.Range(walk, sprint);
        // max acceleration between half the acceleration and the
acceleration value
        _accelerationSpeed = Random.Range(maxAcceleration / 2,
maxAcceleration);
    }


    // Update is called once per frame
    void Update()
    {
        Movement();
        if ((_lastTimeAction + delayBetweenActions) < Time.time)
        {
            if (_inputHandler.GetAttackInputPressed())
            {
                Action(Constant.Animation.ATTACK);
            }
```

```
            else if (_inputHandler.GetDefensenputPressed())
            {
                Action(Constant.Animation.DEFENSE);
            }
        }
    }

    //movement
    private void Movement()
    {
        // converts move input to a world space vector based on our
character's transform orientation
        Vector2 move = _inputHandler.GetMoveInput();

        if (move.Equals(Vector2.zero))
        {
            _animationBlend = 0;
            _speed = 0;
        }
        else
        {
            Vector3 inputDirection = new Vector3(move.x, 0.0f,
move.y).normalized;
            _speed = Mathf.Lerp(_speed, _movementSpeed,
                _accelerationSpeed * Time.fixedTime);
            // move the player
            Quaternion toRotation =
Quaternion.LookRotation(inputDirection, Vector3.up);
            transform.rotation =
Quaternion.RotateTowards(transform.rotation, toRotation,
RotationVelocity * Time.deltaTime);
            _controller.Move(inputDirection * (_speed *
Time.deltaTime));

        }

        //animation
        _animationBlend = Mathf.Lerp(_animationBlend,
_speed,Time.fixedTime * _accelerationSpeed);
        _animator.SetFloat(Constant.Animation.SPEED, _animationBlend);

    }

    private void Action(string action)
    {
        _animator.SetTrigger(action);
        _lastTimeAction = Time.time;
    }

}
```

Por último, el *prefab* de cada objeto robot, los componentes asignados a cada uno de ellos serían los siguientes:

- *Animator*: asignar un controlador de animación que ejecute las animaciones asociadas.
- *Player1InputHandler*: gestionar las acciones de teclado. En este caso del robot1.
- *PlayerController*: gestionar su movimiento, rotación y animación.
- *RigidBody*: permite actuar bajo el control de la física.
- *CharacterController*: mover el robot y define un *capsule collider*.



Visión de futuro

Realizar estas tareas a comprender mejor la visión sobre el funcionamiento de la animación tanto en videojuegos como en películas y entender el trabajo que existe detrás de trabajos como el de diseñadores gráficos, riggers o animadores gráficos.

Respecto al proyecto, es un punto de partida para continuar desarrollando la aplicación y convertirlo en un juego del género de lucha o en uno Beat 'em up individual o cooperativo.

También habría que mejorar el *collider*, y ajustarlo a la forma de cada robot.

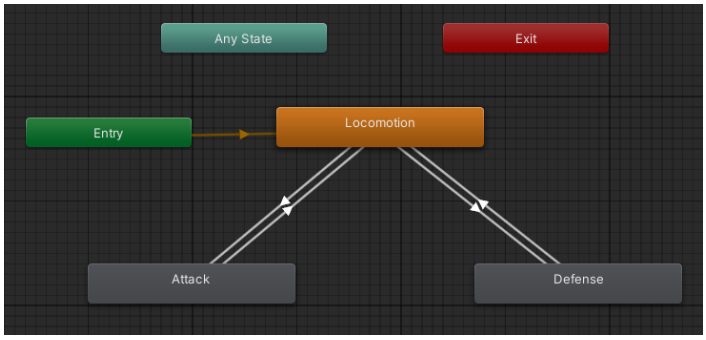
Mejorar el modelo del robot para mejorar las animaciones, ya que al animar el robot, las transformaciones sufridas durante el proceso deforman el modelo.

Explicación sobre los Animator controllers

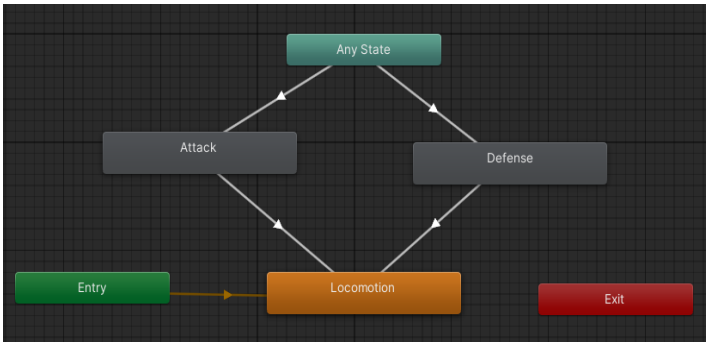
Para realizar la animación de los robots, se han definido 3 estados en la máquina de estados finita:

- *Locomotion*: árbol de mezclas (blend tree), donde según el valor de la velocidad, (speed), se ejecuta una animación.
- *Attack*: animación de ataque
- *Defense*: animación de defensa.

La estructura de la máquina de estados del controlador de animación es la siguiente:



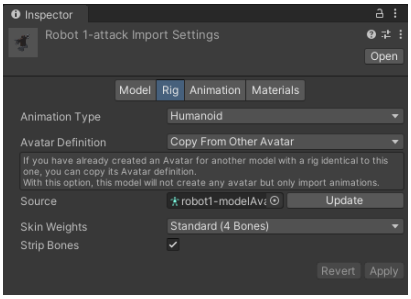
Controlador de animación Robot1



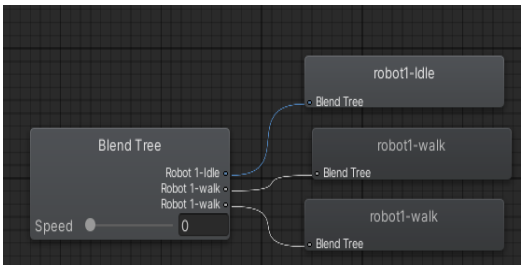
Controlador de animación Robot1

Se ha usado un controlador con una estructura diferente para cada uno para comprobar que de las dos formas se consigue el mismo resultado. La diferencia estriba en el hecho de que en el primero controlador, para pasar a la animación de ataque o defensa, se ha de estar antes en la animación de locomoción, pero en la segunda, da igual en que animación se éste para pasar a la animación de ataque o defensa.

Para asociar una animación a un modelo, se ha de definir la forma del esqueleto y la fuente, es decir, el modelo asociado a la animación como se muestra en la siguiente imagen.

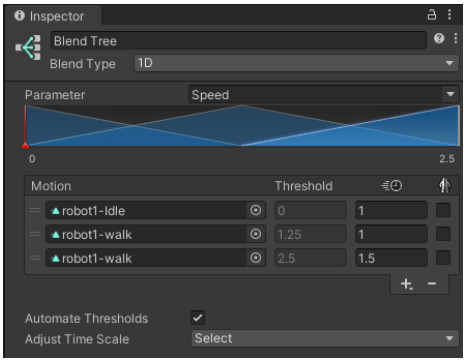


Para definir el árbol de mezclas de locomoción, se ha definido el parámetro velocidad *(speed)* para determinar la animación a ejecutar. Este árbol tiene 3 estados:



- Idle: quieto
- Walk: caminar
- Run: correr

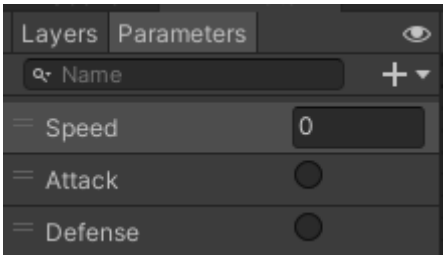
A continuación, se muestra la animación asignada para cada estado. La correspondencia es la siguiente:



- Idle: robot1-idle
- Walk: robot1-walk
- Run: robot1-walk

Para caminar y correr se ha usado la misma animación. Para diferenciar caminar de correr, se ha incrementado la velocidad de animación de correr un 50%.

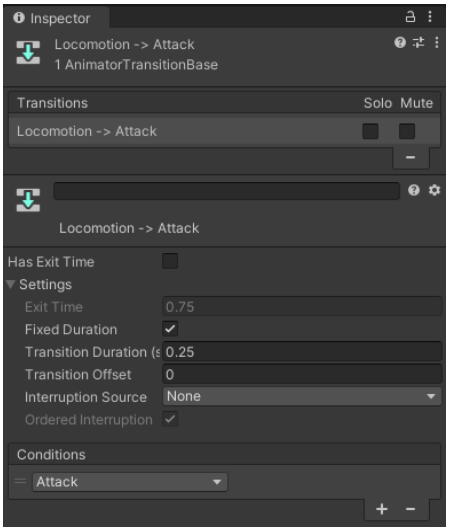
La forma de pasar a la animación de ataque o defensa se realiza mediante un disparador o trigger. Se ha definido un disparador para cada animación:



- Attack: para la animación de ataque.
- Defense: para la animación de defensa.

La condición de cada transición que termina en la animación *locomotion* es la siguiente, tanto para la animación de ataque o defensa, con la salvedad de cambiar el valor del disparador.

Se ha de deseleccionar la opción de *has exit time* para que se ejecute en el momento en que se activa el disparador y en *conditions*, definir que se ha de realizar la transición con el disparador adecuado, en este caso el asociado con la animación de ataque.



Resumen de la creación de las animaciones

Las animaciones de los robots se han intentando hacer que se parezcan a movimientos similares a los humanos, además de armónicos y natural, pero como se ha explicado en el punto Resumen de las entregas, las limitaciones técnicas han impedido realizar unas mejores animaciones.

Se han creado 4 animaciones para cada robot, una para cada acción:

- *Parado*: cuando el robot no realiza ninguna acción
- *Movimiento*: cuando el robot se mueve.
- *Ataque*: cuando el robot realiza un movimiento de ataque.
- *Defensa*: cuando el robot realiza un movimiento de defensa.

En el primer robot, el que tiene orugas por piernas, solo se ha animado el tronco superior, brazos y cabeza.

El segundo robot, el arácnido, se ha buscado que el movimiento de las patas sea similar al de una araña. Este robot ha constado más animarlo debido a la complejidad de sincronizar el movimiento de cada par de patas y que de forma global el movimiento resultase fluido.

Enlace del proyecto

El proyecto se ha subido a la plataforma Github, en él se puede encontrar el ejecutable en la carpeta *executable* con nombre *PEC3.exe*. El enlace al mismo es el siguiente: https://github.com/JoaquinLopezSoriano/Media-PEC3.git

Bibliografía

Software

- Maya: https://www.autodesk.es/products/maya
- Unity: https://unity.com/es

Exportar animaciones

- https://www.youtube.com/watch?v=KiSZ6J0VqS8
- https://www.youtube.com/watch?v=i8cmDK_O-1g

Realizar animaciones con Maya

- https://www.youtube.com/watch?v=1wvdQy2Fdhw
- https://www.youtube.com/watch?v=-KcwtJSAz4g
- https://www.youtube.com/watch?v=3UR9r-tb6KQ

Asset

- City: https://assetstore.unity.com/packages/3d/environments/urban/city-package-107224
- Audio: https://pixabay.com/music/

Movimiento y rotación

- https://docs.unity3d.com/ScriptReference/CharacterController.html
- https://unity.com/es/features/input-system
- https://www.youtube.com/watch?v=BJzYGsMcy8Q
- https://www.youtube.com/watch?v=UJsaEVPntMg

Terminología

- Género beat em up: https://es.wikipedia.org/wiki/Videojuego_de_beat_%27em_up
- Género de lucha: https://es.wikipedia.org/wiki/Videojuego_de_lucha
- Definición de rigger: https://www.notodoanimacion.es/que-es-y-como-hacer-un-buen-rigging/
- Definición de diseñador gráfico: https://www.unir.net/ingenieria/revista/disenador-3d/
- Definición de animador gráfico: https://www.unrealengine.com/es-ES/explainers/animation/what-is-a-3d-animator
- Definición mecha: https://es.wikipedia.org/wiki/Mecha_(ciencia_ficci%C3%B3n)