Animación por Ordenador:

Esqueleto Vulkan

Cristian Delgado Cruz

30 de septiembre del 2023

Índice

Introducción

La práctica final de vulkan, consiste en generar un esqueleto, que vaya cambiando entre unos keyframes para realizar una animación, para ello, se utiliza la clase "*CABalljoin*" que es a su vez la unión de dos clases, *"CASphere*" y "*CACylinder*".

Para ello creamos dos clases nuevas, "*CASkeleton*" y "*CAAnimation*" que van a realizar la generación del esqueleto y la animación, respectivamente.

Mi ejemplo consiste en un chute de un penalti de fútbol, donde el jugador simula el golpeo de una pelota después de coger carrerilla y se queda esperando a saber si marca, una vez sabiendo que ha marcado, festeja en la misma posición.

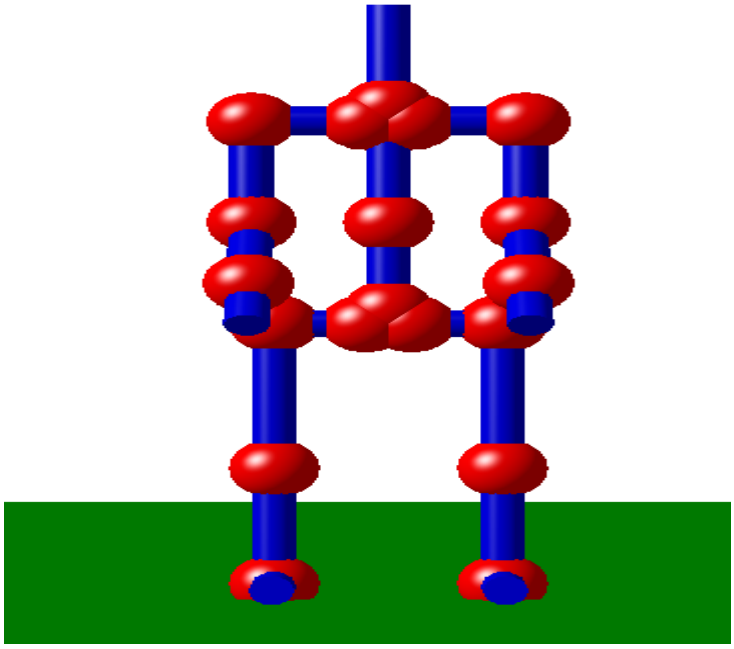

Figura 1

Clases modificadas

CAScene

Dentro del .h, tenemos la declaración del Esqueleto y la Animación, que se usarán para crear el esqueleto y crear la Animación, además tenemos contador y mov, que se utilizarán en el update.

```
// AÑADIDO PARA COMPLETAR LA PRÁCTICA //

CASkeleton* Esqueleto;
CAAnimation* Animacion;

int contador;
float mov;
```

Figura 2

Dentro del .cpp, tenemos por una parte la instanciación de los objetos y los atributos en el constructor, tenemos dos formas de orientar el esqueleto definidas, pero una está comentada.

```
// AÑADIDO PARA COMPLETAR LA PRÁCTICA //

Esqueleto = new CASkeleton();
Esqueleto->createBuffers(vulkan);
Esqueleto->setLocation(glm::vec3(0.0f, 1.0f, 0.0f));
//Esqueleto->setOrientation(glm::vec3(0.5f, 0.0f, 0.5f), glm::vec3(0.0f, 1.0f, 0.0f)); //De lado
Esqueleto->setOrientation(glm::vec3(0.0f, 0.0f, 1.0f), glm::vec3(0.0f, 1.0f, 0.0f)); //De frente
Esqueleto->setLight(light);

Animacion = new CAAnimation(Esqueleto);
contador = 0;
mov = -1.0f;
```

Figura 3

Los demás métodos actúan igual para el esqueleto que para el ground, eliminando, añadiendo los comandos, etc.

```
//
// FUNCIÓN: CAScene::~CAScene()
//
// PROPÓSITO: Destruye el objeto que representa la escena
//
CAScene::~CAScene()
{
    delete ground;


    // AÑADIDO PARA COMPLETAR LA PRÁCTICA //

    delete Esqueleto;

}
```

```
//
// FUNCIÓN: CAScene::destroyBuffers(CAVulkanState* vulkan)
//
// PROPÓSITO: Destruyelos buffers de las figuras que forman la escena
//
void CAScene::destroyBuffers(CAVulkanState* vulkan)
{
    ground->destroyBuffers(vulkan);


    // AÑADIDO PARA COMPLETAR LA PRÁCTICA //

    Esqueleto->destroyBuffers(vulkan);

}
```

Figura 4 y Figura 5

Por otra parte el método update usa el contador y el movimiento para poner las poses del esqueleto y la animación, a cada KeyFrames se le asigna un número del contador, y el paso entre poses, se le asigna un rango entre esos números asignados a las KeyFrames, cuando se llega al final, es decir, al último número equivalente a un KeyFrame, se reinicia el contador.
Aparte, también tenemos un movimiento que efectúa el esqueleto, que también tiene asignado un rango del contador, que es el mismo rango sobre el que se efectúan las KeyFrames y movimientos intermedios de la carrera del esqueleto antes de chutar.

```
// REPETICION //

else {
    Animacion->realizarAnimacion(0); //Reiniciar todas las poses
    contador = -1; //Reiniciar contador
    mov = -1.0f; //Colocar atras el esqueleto
}


// MOVIMIENTO DEL ESQUELTO //

if (contador < 800) {
    mov = mov + 0.0025f;//de -1.0f a 1.0f
    Esqueleto->setLocation(glm::vec3(0.0f, 1.0f, mov));
}
contador++;
```

```
// MOVIMIENTO DE PIERNAS Y BRAZOS INICIAL//

if (contador == 0) {
    Animacion->posicionInicial(1);
}
else if (contador > 0 && contador < 100){
    Animacion->realizarAnimacion(1);
}
else if (contador == 100) {
    Animacion->posicionFinal(1);
    Animacion->posicionInicial(2);
}
```

Figura 6 y Figura 7

CABalljoin

Dentro del .h, tenemos la declaración de un vector de hijos que son también Balljoint, así podemos realizar asociaciones de hijos y padres, para ello también tenemos declarado la matriz Padre, y los métodos para asociar tanto los hijos como recoger y asignar la matriz Padre.

```
// AÑADIDO PARA COMPLETAR LA PRÁCTICA //

std::vector<CABalljoint*> Hijos;
glm::mat4 Padre;
```

```
// AÑADIDO PARA COMPLETAR LA PRÁCTICA //

void setSon(CABalljoint* s);
glm::mat4 getFatherMatrix();
void setFatherMatrix(glm::mat4 f);
```

Figura 8 y Figura 9

Dentro del .cpp, en el constructor definimos la matriz padre como una matriz unidad, con eso nos ahorramos un problema con aquellas BallJoint que no tengan padre. Por otra parte los métodos que realizaban por ejemplo, el delete, los comandos etc, que llaman la escena, también llamará a sus respectivos hijos para que hagan lo mismo.

```
// AÑADIDO PARA COMPLETAR LA PRÁCTICA //

Padre = glm::mat4(1.0f);
```

```
// AÑADIDO PARA COMPLETAR LA PRÁCTICA //

if (Hijos.size() > 0)
{
    for (int i = 0; i < Hijos.size(); i++)
    {
        delete Hijos[i];
    }
}
```

Figura 10 y Figura 11

Dentro del constructor, además, tenemos el siguiente código el cual asignará a los hijos la matriz "*mms*" la cual se colocará al final del hueso (cilindro) gracias a trasladar una vez más a la mitad del length total del hueso.

```
// AÑADIDO PARA COMPLETAR LA PRÁCTICA //

// Sumamos el padre que en caso de ser este el padre sera matriz identidad

glm::mat4 matrix = Padre * jointm * posem;


joint->setLocation(matrix);
glm::mat4 mm = glm::translate(matrix, glm::vec3(0.0f, 0.0f, length / 2));
bone->setLocation(mm);

glm::mat4 mms = glm::translate(mm, glm::vec3(0.0f, 0.0f, length/2));

if (Hijos.size() > 0) {
    for (int i = 0; i < Hijos.size(); i++)
    {
        Hijos[i]->setFatherMatrix(mms);
    }
}
```

Figura 12

Clases creadas

CASkeleton

Dentro del .h, tenemos los métodos públicos que son los mismos que el BallJoint, y los atributos, que también son los mismos, faltando length y matriz padre, pues el esqueleto no se dibuja, como tal, sino, que crea todo lo demás y sirve para mover todas las BallJoint a la vez.

```
#include "CAFigure.h"
#include "CABalljoint.h"

class CASkeleton : public CAFigure {


public:

    glm::vec3 location;
    glm::vec3 dir;
    glm::vec3 up;
    glm::vec3 right;
    GLfloat angles[3];
    std::vector<CABalljoint*> Hijos;

    CASkeleton();
    ~CASkeleton();

    void createBuffers(CAVulkanState* vulkan);
    void destroyBuffers(CAVulkanState* vulkan);
    void addCommands(CAVulkanState* vulkan, VkCommandBuffer commandBuffer, int index);
    void updateDescriptorSets(CAVulkanState* vulkan, uint32_t imageIndex, glm::mat4 view, glm::mat4 projection);
    void ComputeMatrix();
    void setLight(CALight l);
    void setLocation(glm::vec3 loc);
    void setOrientation(glm::vec3 nDir, glm::vec3 nUp);
    void setPose(float xrot, float yrot, float zrot);
    void setPoseArticulacion(int pose, float a, float b, float c, float d);
```

Figura 13

Por otra parte aquí están todos los hijos y la declaración del método para generar el esqueleto, el cual será llamado en el constructor.

```
private:

    // AÑADIDO PARA COMPLETAR LA PRÁCTICA //

    CABalljoint* pelvis; //Nodo Hijo 1
    CABalljoint* caderaI; //Nodo Hijo 2
    CABalljoint* caderaD; //Nodo Hijo 3
    CABalljoint* columna; //Nodo Hijo 1-1
    CABalljoint* piernaI; //Nodo Hijo 2-1
    CABalljoint* piernaD; //Nodo Hijo 3-1
    CABalljoint* cuello; //Nodo Hijo 1-1-1
    CABalljoint* claviculaI; //Nodo Hijo 1-1-2
    CABalljoint* claviculaD; //Nodo Hijo 1-1-3
    CABalljoint* rodillaI; //Nodo Hijo 2-1-1
    CABalljoint* rodillaD; //Nodo Hijo 3-1-1
    CABalljoint* hombroI; //Nodo Hijo 1-1-2-1
    CABalljoint* hombroD; //Nodo Hijo 1-1-3-1
    CABalljoint* tobilloI; //Nodo Hijo 2-1-1-1
    CABalljoint* tobilloD; //Nodo Hijo 3-1-1-1
    CABalljoint* codoI; //Nodo Hijo 1-1-2-1-1
    CABalljoint* codoD; //Nodo Hijo 1-1-3-1-1
    CABalljoint* munecaI; //Nodo Hijo 1-1-2-1-1-1
    CABalljoint* munecaD; //Nodo Hijo 1-1-3-1-1-1

    void generate(CAVulkanState* vulkan);
```

Figura 14

En el .cpp, usamos los mismos métodos que en el BallJoint , pero sin generar nada del propio padre (esqueleto), dentro del método createbuffer llamamos al generate, que creará el esquema de hijos.

```
//
// FUNCIÓN: CASkeleton::createBuffers(CAVulkanState* vulkan)
//
// PROPÓSITO: Inicializa las piezas de la figura
//
void CASkeleton::createBuffers(CAVulkanState* vulkan)
{
    CAMaterial jointMat = {};
    jointMat.Ka = glm::vec3(1.0f, 0.0f, 0.0f);
    jointMat.Kd = glm::vec3(1.0f, 0.0f, 0.0f);
    jointMat.Ks = glm::vec3(0.8f, 0.8f, 0.8f);
    jointMat.Shininess = 16.0f;

    generate(vulkan);

}
```

```
glm::mat4 matrix = jointm * posem;

if (Hijos.size() > 0) {
    for (int i = 0; i < Hijos.size(); i++)
    {
        Hijos[i]->setFatherMatrix(matrix);
    }
}
```

Figura 15 y 16

```
//
// FUNCIÓN: CASkeleton::generate(CAVulkanState* vulkan)
//
// PROPÓSITO: Genera el esqueleto.
//
void CASkeleton::generate(CAVulkanState* vulkan)
{

    /*********************************/
    /*******HIJOS DE ARRIBA***********/
    /*********************************/

    pelvis = new CABalljoint(0.3f);
    pelvis->createBuffers(vulkan);
    columna = new CABalljoint(0.4f);
    columna->createBuffers(vulkan);
    cuello = new CABalljoint(0.35f);
    cuello->createBuffers(vulkan);

    pelvis->setSon(columna);
    columna->setSon(cuello);


    // IZQUIERDA //

    claviculaI = new CABalljoint(0.25f);
    claviculaI->createBuffers(vulkan);
    hombroI = new CABalljoint(0.35f);
    hombroI->createBuffers(vulkan);
    codoI = new CABalljoint(0.30f);
    codoI->createBuffers(vulkan);
    munecaI = new CABalljoint(0.20f);
    munecaI->createBuffers(vulkan);
```

```
// CONFIGURACION DEL PADRE //

pelvis->setLocation(glm::vec3(0.0f, 0.0f, 0.0f));
pelvis->setOrientation(glm::vec3(0.0f, 1.0f, 0.0f), glm::vec3(0.0f, 0.0f, 1.0f));

// CONFIGURACION DE LOS HIJOS //

claviculaI->setLocation(glm::vec3(-0.05f, 0.0f, -0.05f));
claviculaD->setLocation(glm::vec3(0.05f, 0.0f, -0.05f));
claviculaI->setOrientation(glm::vec3(-1.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
claviculaD->setOrientation(glm::vec3(1.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));

// CONFIGURACION DE LOS BRAZOS PARA QUE ESTEN HACIA ABAJO //
// SOLO SE NOTA SI NO REALIZAMOS LA ANIMACION //

hombroD->setPose(0.0f, 90.0f, 0.0f);
hombroI->setPose(0.0f, -90.0f, 0.0f);
codoD->setPose(-45.0f, 0.0f, 0.0f);
codoI->setPose(-45.0f, 0.0f, 0.0f);

/*********************************/
/*********HIJOS DE ABAJO********/
/*********************************/


// IZQUIERDA //

caderaI = new CABalljoint(0.2f);
caderaI->createBuffers(vulkan);
piernaI = new CABalljoint(0.5f);
piernaI->createBuffers(vulkan);
rodillaI = new CABalljoint(0.4f);
rodillaI->createBuffers(vulkan);
tobilloI = new CABalljoint(0.25f);
```

Figura 17 y Figura 18

Además el método “*setPoseArticulacion*” asigna poses a sus hijos respecto a cual KeyFrame queremos usar y los ángulos para cada movimiento, siendo el caso 0 el que reinicia entero.

```
// FUNCIÓN: CASkeleton::generate(int pose, float a, float b, float c, float d)
//
// PROPÓSITO: Asigna las posiciones de los angulos calculadas a las articulaciones pertinentes.
//
void CASkeleton::setPoseArticulacion(int pose, float a, float b, float c, float d) {

    switch (pose)
    {
    case 0:
        piernaD->setPose(0.0f, 0.0f, 0.0f);
        rodillaD->setPose(0.0f, 0.0f, 0.0f);
        piernaI->setPose(0.0f, 0.0f, 0.0f);
        rodillaI->setPose(0.0f, 0.0f, 0.0f);
        hombroI->setPose(0.0f, -90.0f, 0.0f);
        codoI->setPose(-45.0f, 0.0f, 0.0f);
        hombroD->setPose(0.0f, 90.0f, 0.0f);
        codoD->setPose(-45.0f, 0.0f, 0.0f);
        claviculaD->setPose(0.0f, 0.0f, 0.0f);
        caderaD->setPose(0.0f, 0.0f, 0.0f);
        break;
    case 1:
        piernaD->setPose(a, 0.0f, 0.0f);
        rodillaD->setPose(b, 0.0f, 0.0f);
        piernaI->setPose(-a, 0.0f, 0.0f);
        rodillaI->setPose(-b, 0.0f, 0.0f);
```

Figura 19

CAAnimation

Dentro del .h, tenemos los métodos para generar la animación, además de colocar la pose (KeyFrame) inicial y final, y los atributos para cambiar los ángulos y recoger el esqueleto.

Figura 20

Dentro del .cpp, el constructor asigna el esqueleto y coloca los ángulos a 0, y los métodos que asignan las posiciones iniciales y finales son los “Keyframes”

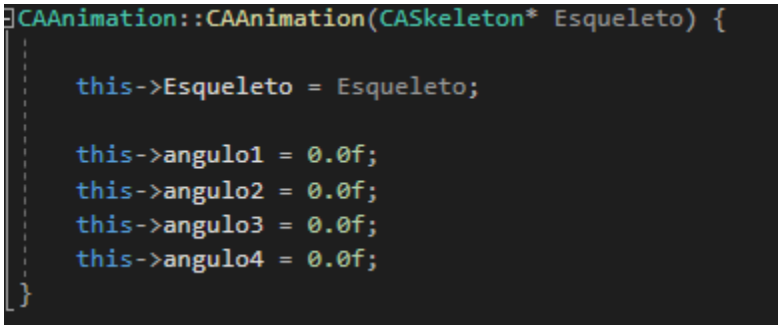

Figura 21

```
//
// FUNCIÓN: CAAnimation::posicionFinal(int keyFrame)
//
// PROPÓSITO: Coloca el esqueleto en la posicion inicial(pose) del keyframe que le pasan
//
void CAAnimation::posicionInicial(int keyFrame) {

    switch (keyFrame)
    {
    case 0:
        angulo1 = 0.0f;
        angulo2 = 0.0f;
        angulo3 = 0.0f;
        angulo4 = 0.0f;
        break;
    case 1:
        angulo1 = 0.0f;
        angulo2 = 0.0f;
        angulo3 = 0.0f;
        angulo4 = 0.0f;
        break;
```

Figura 22

```
//
// FUNCIÓN: CAAnimation::posicionFinal(int keyFrame)
//
// PROPÓSITO: Coloca el esqueleto en la posicion final(pose) del keyframe que le pasan
//
void CAAnimation::posicionFinal(int keyFrame) {

    switch (keyFrame)
    {
    case 1:
        angulo1 = 45.0f;
        angulo2 = -30.0f;
        angulo3 = 45.0f;
        angulo4 = 30.0f;
        break;
    case 2:
        angulo1 = 0.0f;
        angulo2 = 0.0f;
        angulo3 = 0.0f;
        angulo4 = 0.0f;
        break;
    case 3:
```

Figura 23

Dentro de este método, se realiza toda la animación, para ello se usa la fórmula siguiente:

Contador(Rango) * Velocidad(Suma del ángulo por unidad del contador) = Angulo

Con esto y el Keyframe podemos realizar toda la animación.

```
//
// FUNCIÓN: CAAnimation::realizarAnimacion(int keyFrame)
//
// PROPÓSITO: Realiza la posicion que le pasan
//
void CAAnimation::realizarAnimacion(int keyFrame) {

    switch (keyFrame) //Formula usada => Contador(Cantidad que le corresponde) * Velocidad(Suma del angulo) = Angulo; Velocidad = Angulo/Contador;
    {
    case 0:
        angulo1 = 0.0f;
        angulo2 = 0.0f;
        angulo3 = 0.0f;
        angulo4 = 0.0f;
        break;
    case 1: //Contador = 100
        angulo1 = angulo1 + 0.45f; //Hasta el angulo 45
        angulo2 = angulo2 - 0.25f; //Hasta el angulo -30
        angulo3 = angulo3 + 0.45f; //Hasta el angulo 45
        angulo4 = angulo4 + 0.25f; //Hasta el angulo 30
        break;
    case 2: //Contador = 100
        angulo1 = angulo1 - 0.45f; //Hasta el angulo 0
        angulo2 = angulo2 + 0.30f; //Hasta el angulo 0
        angulo3 = angulo3 - 0.45f; //Hasta el angulo 0
        angulo4 = angulo4 - 0.25f; //Hasta el angulo 0
        break;
```

Figura 24

Poses (KeyFrames)

Tenemos 8 poses, con un máximo de un movimiento distinto de 4 articulaciones:

La pose 1 y la pose 3, son la misma pose pero de forma diferentes, la pose 1 es el movimiento de la pierna derecha y brazo izquierdo hacia delante, y los inversos hacia atrás, y la pose 3 es lo contrario, mientras que la pose 2 y 4, es volver al punto 0 con todo el esqueleto.

Las piernas y hombros llegan hasta el ángulo 45, comenzando desde el 0, y los brazos y tobillos hasta el 30, comenzando también desde el 0, (si van hacia atrás, -45 y -30), cuando se realiza la pose 4, se vuelve a la pose 1, y se realiza la secuencia 1-2-3-4, una vez más, hasta un total de 2.

También se mueve el esqueleto a z = -1.0f, desde z= 1.0f;

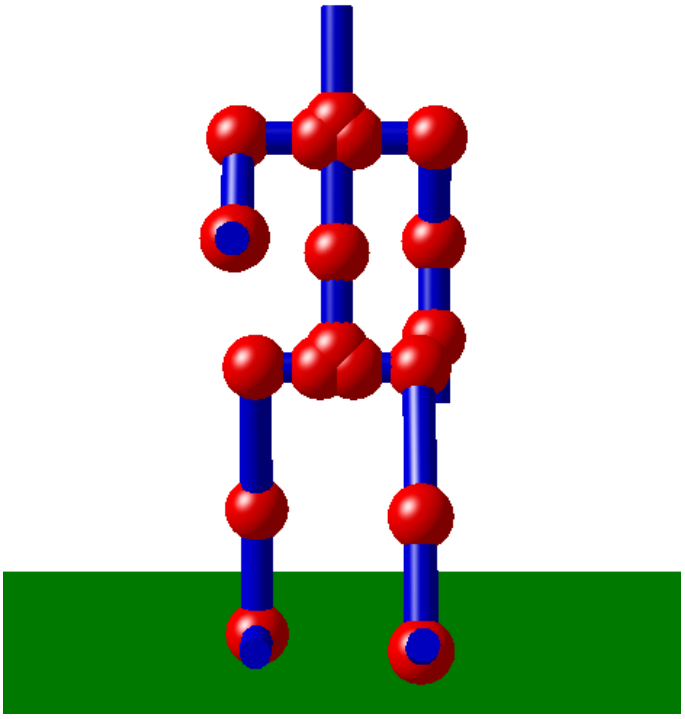

Figura 25

La pose 5 mueve la pierna hacia atrás hasta los -50 grados, la rodilla también hacia atrás hasta los -80 grados, la clavícula hacia atrás igualmente, hasta los 40 grados, y la cadera un pelín hacia arriba, para hacer un chute cruzado, hasta los 15 grados.

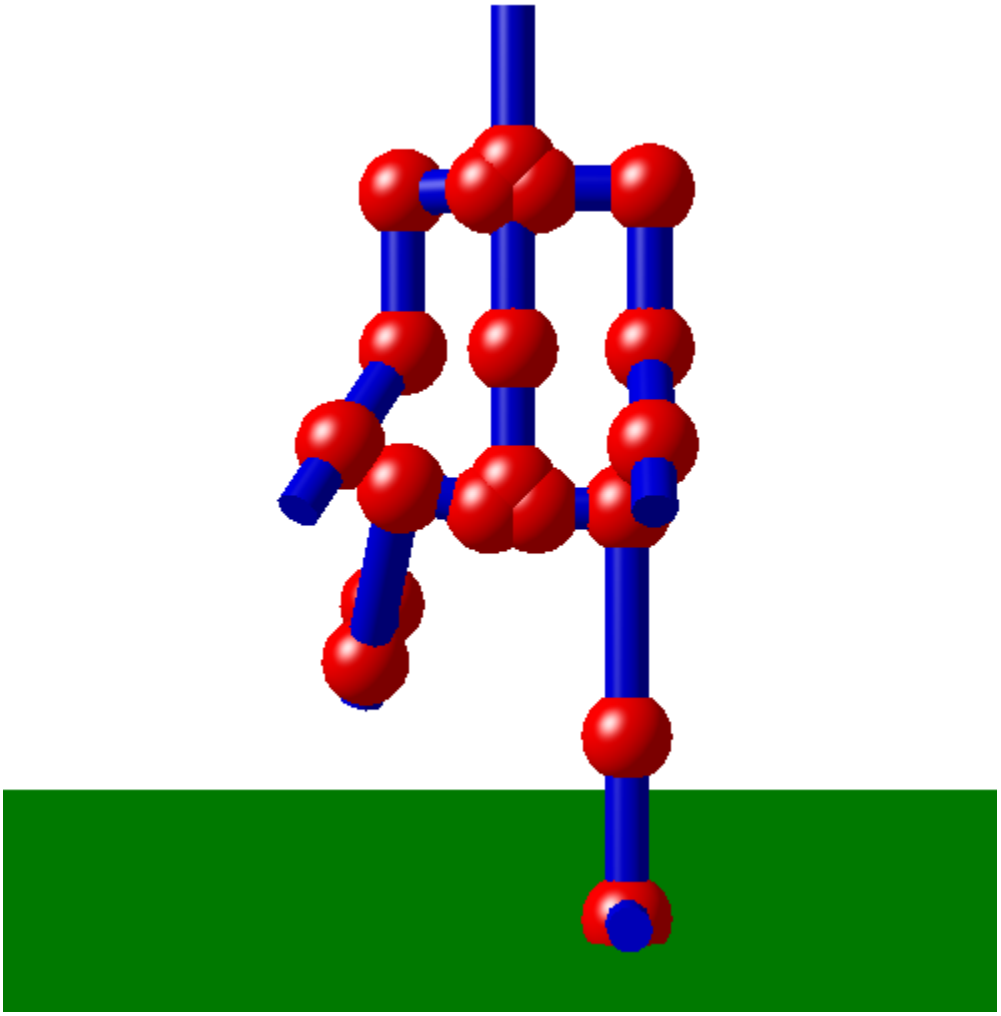

Figura 26

La pose 6 mueve la pierna hacia delante 60 grados, la rodilla también hacia delante hasta los 0 grados, la clavícula hacia delante igualmente, hasta los -15 grados, y la cadera un pelín hacia abajo , para hacer un chute cruzado, hasta los 5 grados.

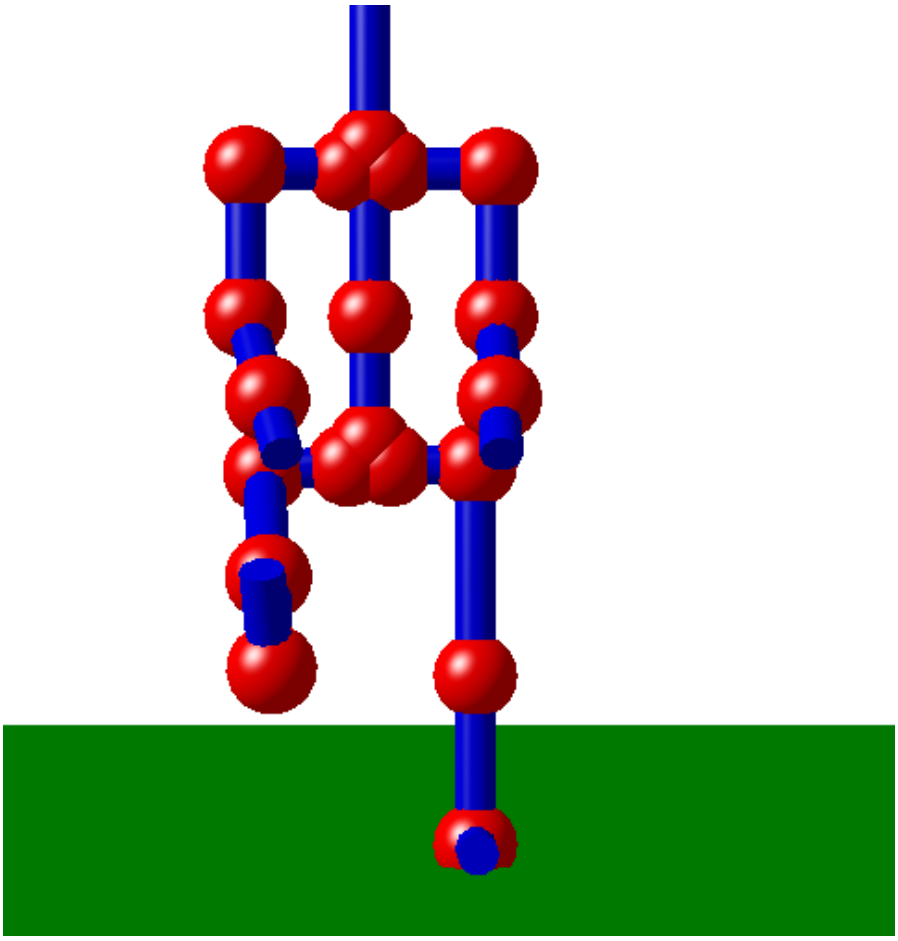

Figura 27

La pose 7 mueve la pierna hacia atrás hasta los 0 grados, la rodilla también hacia atrás hasta los 0 grados, la clavícula hacia atrás igualmente, hasta los 0 grados, y la cadera un pelín hacia arriba, hasta los 0 grados, además, levanta los codos hasta -135 para colocarse en posición como de duda, mientras que mira si el portero para el penalti o marca gol.

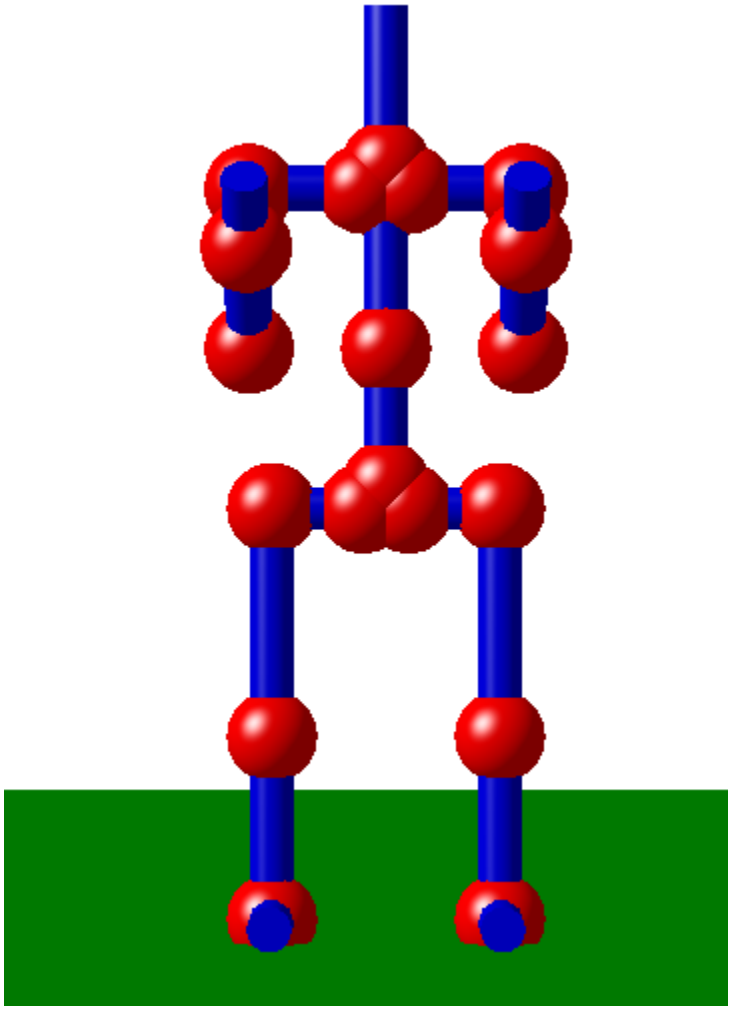

Figura 28

La pose 8 mueve el codo hasta los 0 grados para ponerlo recto, y el brazo hasta los 180 grados en su coordenada z, para colocarlos hacia arriba, a modo de celebración estática.

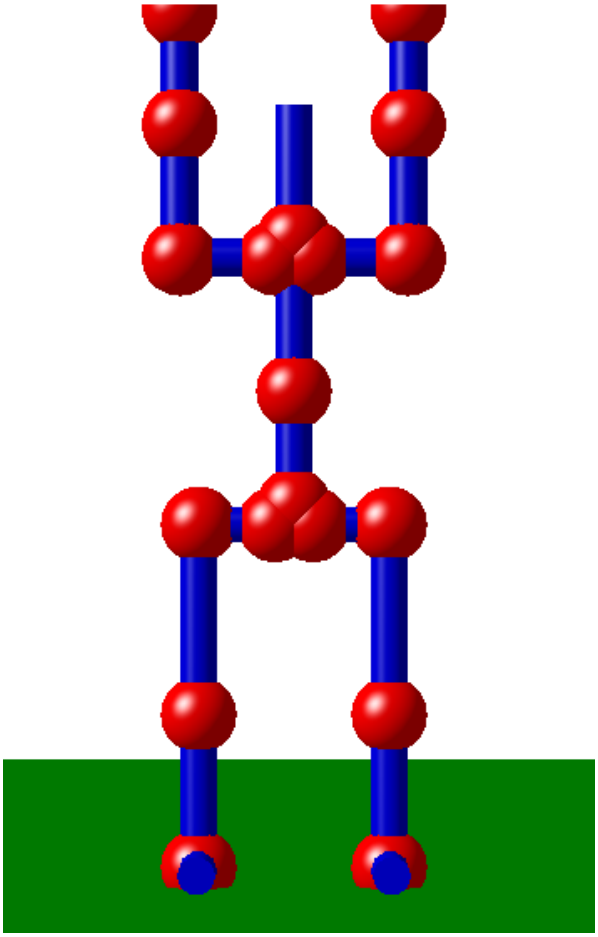

Figura 29

Pruebas

Como el método update es de la clase escena, y se llama cada segundo, tuve que crear si o si un contador para poder realizar dos animaciones diferentes con if-else, por lo que cree contador.

Al principio actualizaba la localización del esqueleto cuando se llamaba a la posición inicial del movimiento de las piernas, pero, eso hacía que el esqueleto se teletransportase, con cada iteración, por lo que necesitaba que aumentase con el contador, por lo que lo separe, y cree mov para que fuera una variable auxiliar para poder reiniciarla al comenzar.

El tiempo (rango de contador) que tiene cada pose es diferente, porque algunas poses necesitan moverse más ángulo que otras y para que la velocidad fuera más o menos parecida, se necesita que sean diferentes por la fórmula usada.

Desde contador igual a 1225 hasta 1325, se queda parado esperando para saber si marca, y al final también colocamos de 1475 hasta 1600 para saber que la animación ha terminado.

En realidad, se podrían agregar bastantes más poses para generar una animación aún más fluida, o para animar por ejemplo la celebración de forma más dinámica, pero tampoco es lo que se pide en la práctica, así que he preferido dejarlo ya de esta forma.