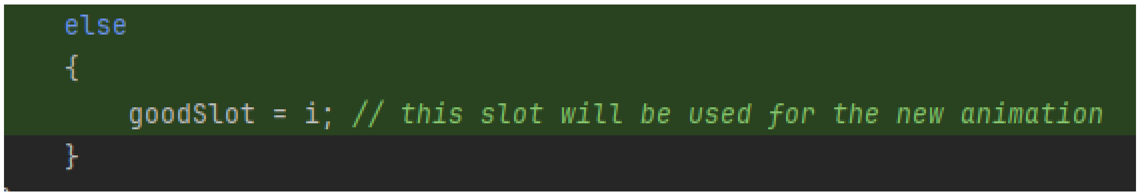
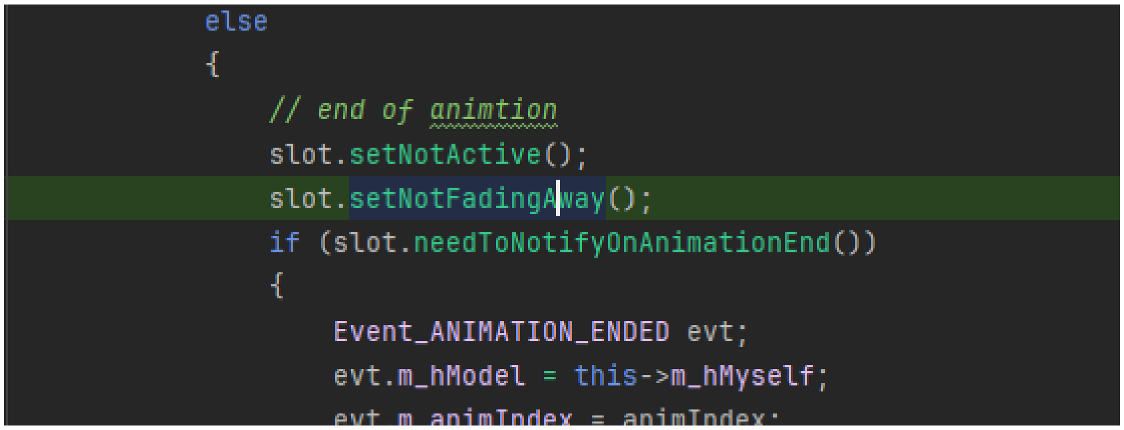
Video Link:
https://drive.google.com/file/d/1ifHOKcIPQUIFxiuI7EHNNeiJHN8_xk1t/view?usp=drive_link

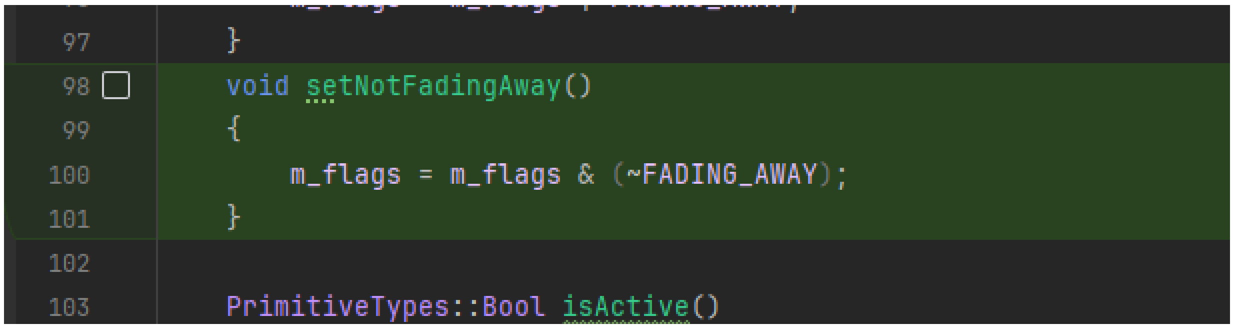
Task #1: Blend between two animations
Inside DefaultAnimationSM::setAnimation, goodslot is not set to a new value when one animation is not active, simple set goodslot index to the current iterated animation slot even when this animation slot is not active, also we need to change the FADING_AWAY bit when an animation ends.



```
    else
    {
        goodSlot = i; // this slot will be used for the new animation
    }
```



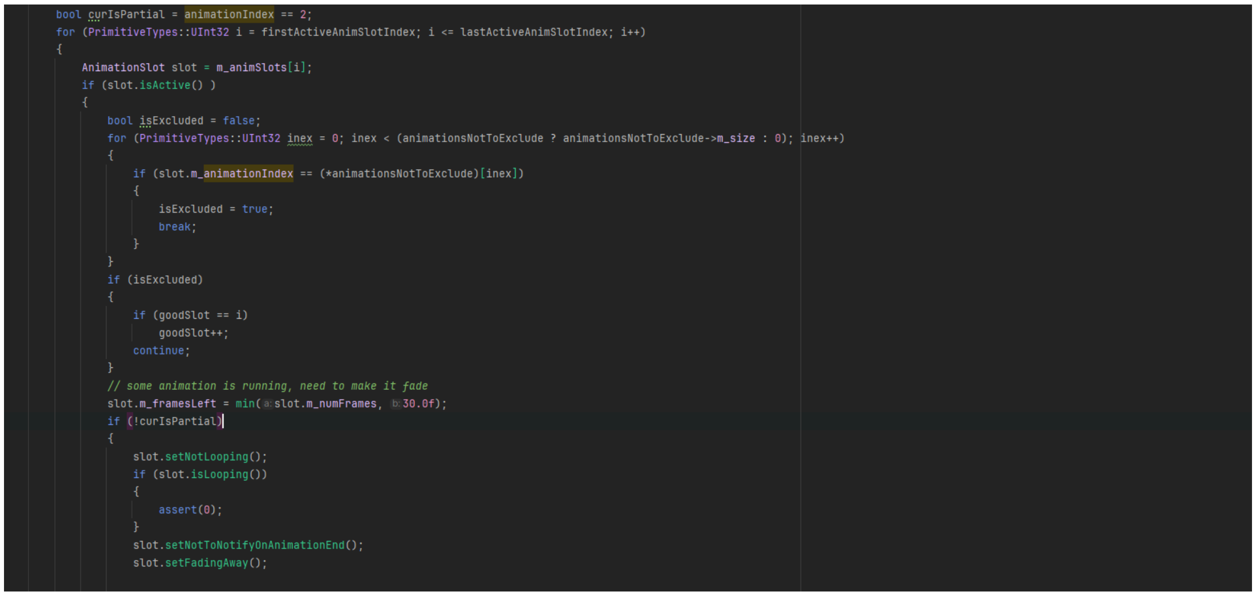
```
            else
            {
                // end of animtion
                slot.setNotActive();
                slot.setNotFadingAway();
                if (slot.needToNotifyOnAnimationEnd())
                {
                    Event_ANIMATION_ENDED evt;
                    evt.m_hModel = this->m_hMyself;
                    evt.m_animIndex = animIndex;
```



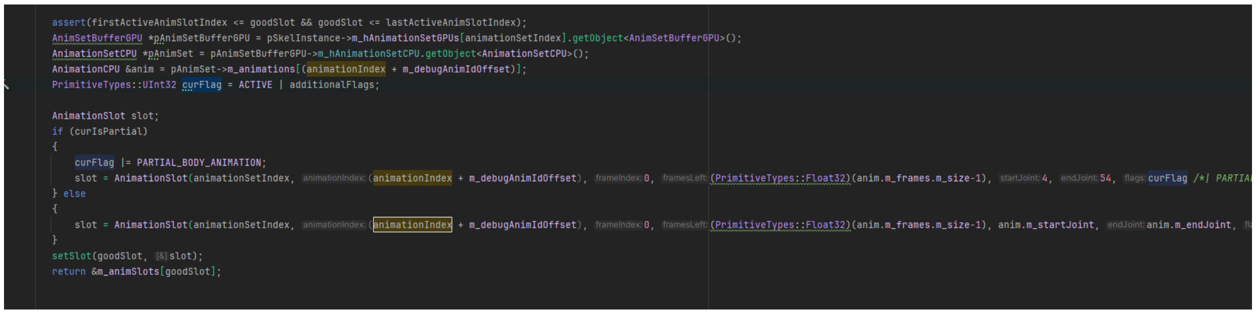
```
    }
    void setNotFadingAway()
    {
        m_flags = m_flags & (~FADING_AWAY);
    }

    PrimitiveTypes::Bool isActive()
```

Task #2: partial blending
Because all of our animations data includes all joints, in order to do partial animation,I set a fixed animation to be partial, which is the a02_aim animation in vampire-sample-action-pack_Hips.animseta. I picked the upper body joints indexed from 4- 54, marked if as PARTIAL_BODY_ANIMATION. Do not deactivate or set the current animation to

fading when calling setAnimation if the new animation is a partial body animation.



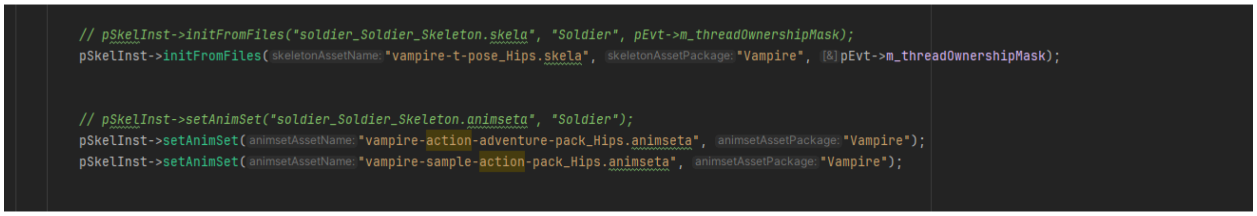
```
bool curIsPartial = animationIndex == 2;
for (PrimitiveTypes::UInt32 i = firstActiveAnimSlotIndex; i <= lastActiveAnimSlotIndex; i++)
{
    AnimationSlot slot = m_animSlots[i];
    if (slot.isActive() )
    {
        bool isExcluded = false;
        for (PrimitiveTypes::UInt32 inex = 0; inex < (animationsNotToExclude ? animationsNotToExclude->m_size : 0); inex++)
        {
            if (slot.m_animationIndex == (*animationsNotToExclude)[inex])
            {
                isExcluded = true;
                break;
            }
        }
        if (isExcluded)
        {
            if (goodSlot == i)
                goodSlot++;
            continue;
        }
        // some animation is running, need to make it fade
        slot.m_framesLeft = min( a: slot.m_numFrames,  b: 30.0f);
        if (!curIsPartial)
        {
            slot.setNotLooping();
            if (slot.isLooping())
            {
                assert(0);
            }
            slot.setNotToNotifyOnAnimationEnd();
            slot.setFadingAway();
```



```
assert(firstActiveAnimSlotIndex <= goodSlot && goodSlot <= lastActiveAnimSlotIndex);
AnimSetBufferGPU *pAnimSetBufferGPU = pSkelInstance->m_hAnimationSetGPUs[animationSetIndex].getObject<AnimSetBufferGPU>();
AnimationSetCPU *pAnimSet = pAnimSetBufferGPU->m_hAnimationSetCPU.getObject<AnimationSetCPU>();
AnimationCPU &anim = pAnimSet->m_animations[(animationIndex + m_debugAnimIdOffset)];
PrimitiveTypes::UInt32 curFlag = ACTIVE | additionalFlags;

AnimationSlot slot;
if (curIsPartial)
{
    curFlag |= PARTIAL_BODY_ANIMATION;
    slot = AnimationSlot(animationSetIndex,  animationIndex: (animationIndex + m_debugAnimIdOffset),  frameIndex: 0,  framesLeft: (PrimitiveTypes::Float32)(anim.m_frames.m_size-1),  startJoint: 4,  endJoint: 54,  flags: curFlag /*| PARTIA
} else
{
    slot = AnimationSlot(animationSetIndex,  animationIndex: (animationIndex + m_debugAnimIdOffset),  frameIndex: 0,  framesLeft: (PrimitiveTypes::Float32)(anim.m_frames.m_size-1), anim.m_startJoint,  endJoint: anim.m_endJoint,
}
setSlot(goodSlot,  [&] slot);
return &m_animSlots[goodSlot];
```

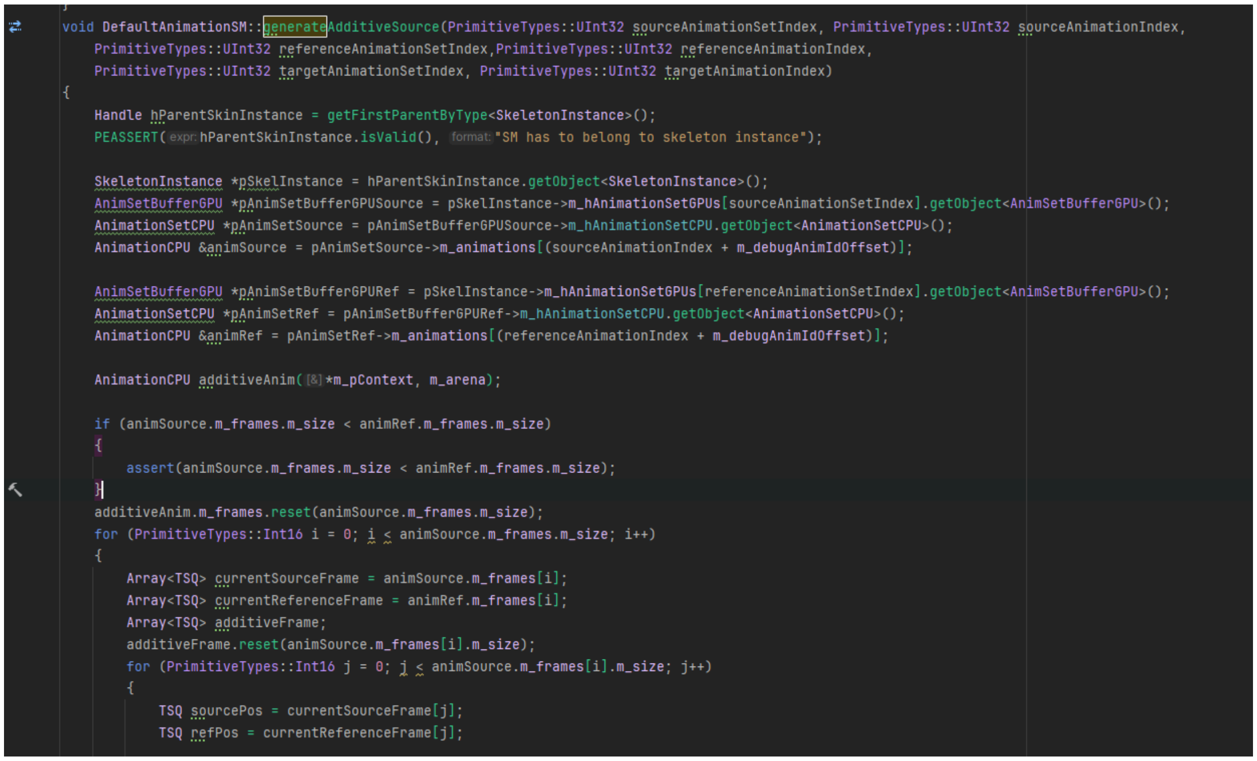
#Task3: additive blending

Load the 2 given animation sources inside SoldierNPC::SoldierNPC

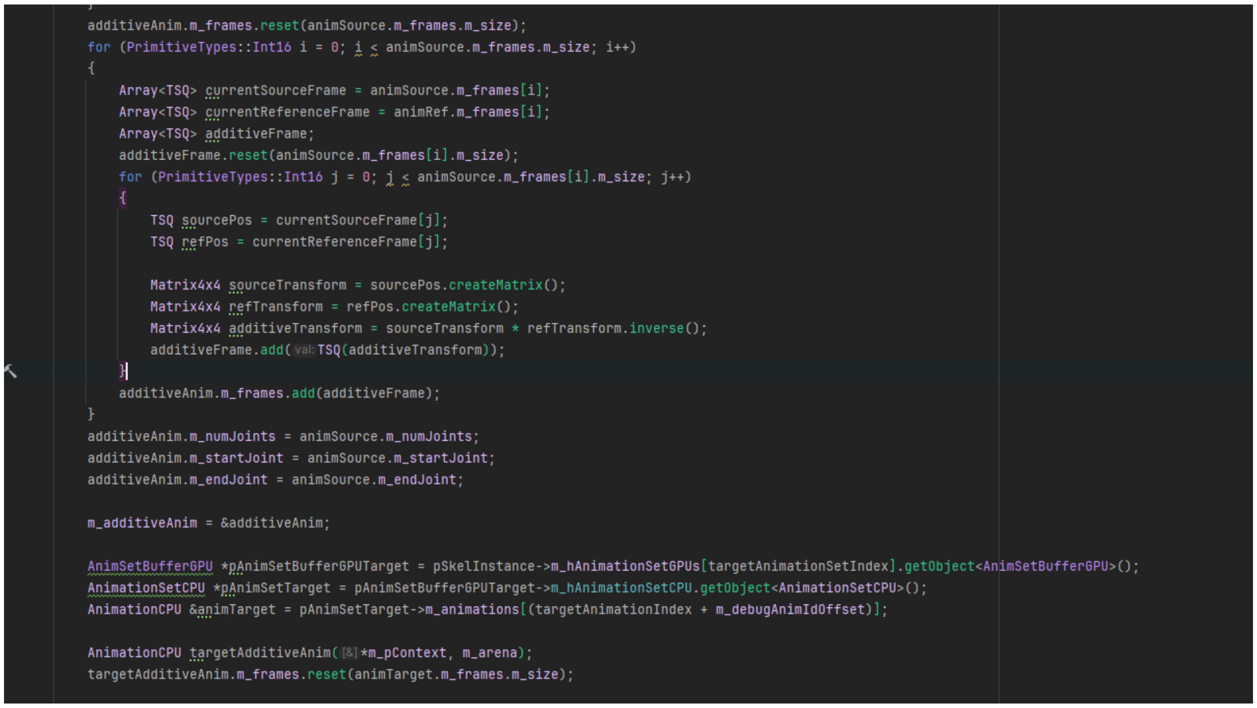


```
// pSkelInst->initFromFiles("soldier_Soldier_Skeleton.skela", "Soldier", pEvt->m_threadOwnershipMask);
pSkelInst->initFromFiles( skeletonAssetName: "vampire-t-pose_Hips.skela",  skeletonAssetPackage: "Vampire",  [&] pEvt->m_threadOwnershipMask);


// pSkelInst->setAnimSet("soldier_Soldier_Skeleton.animseta", "Soldier");
pSkelInst->setAnimSet( animsetAssetName: "vampire-action-adventure-pack_Hips.animseta",  animsetAssetPackage: "Vampire");
pSkelInst->setAnimSet( animsetAssetName: "vampire-sample-action-pack_Hips.animseta",  animsetAssetPackage: "Vampire");
```

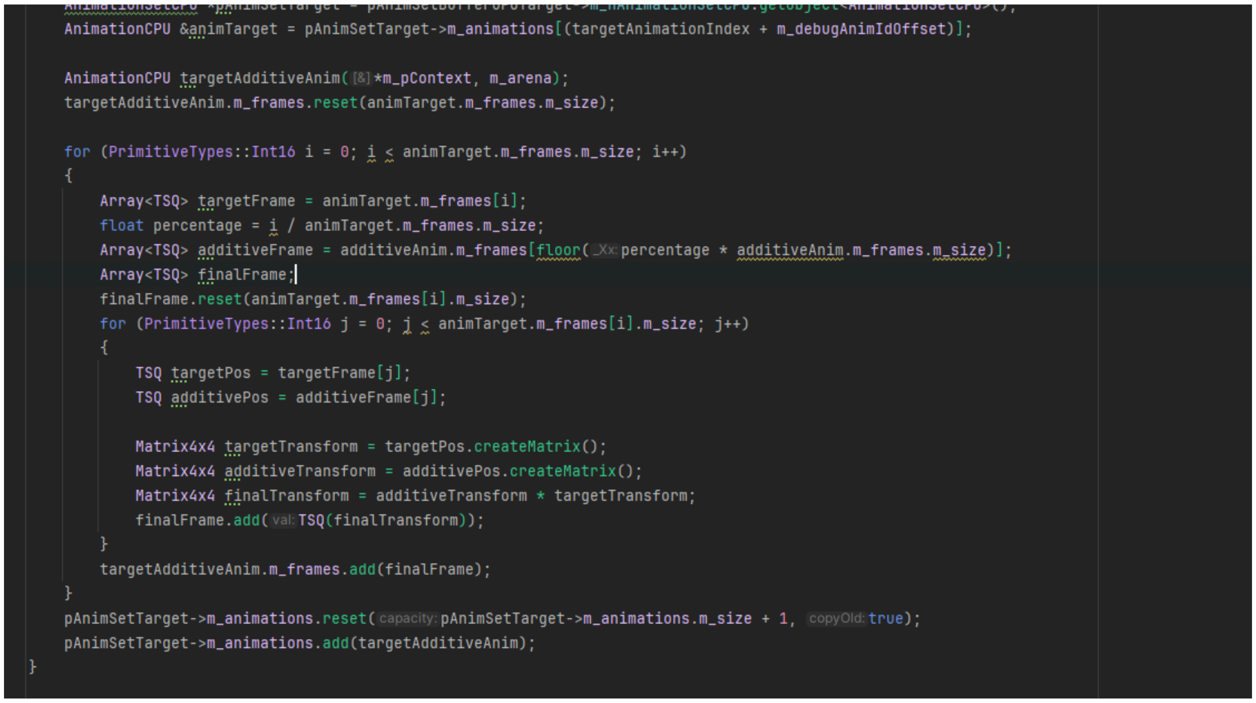
First pick an additive source: a02_aim in vampire-sample-action-pack_Hips.animseta.
Pick a reference source: a02_idle from vampire-action-adventure-pack_Hips.animseta.
Last pick a target animation source to add the additive clip on: a00_idle from vampire-sample-action-pack_Hips.animseta.
We now have our source, we will generate a new clip using the clips above



```cpp
void DefaultAnimationSM::generateAdditiveSource(PrimitiveTypes::UInt32 sourceAnimationSetIndex, PrimitiveTypes::UInt32 sourceAnimationIndex,
    PrimitiveTypes::UInt32 referenceAnimationSetIndex,PrimitiveTypes::UInt32 referenceAnimationIndex,
    PrimitiveTypes::UInt32 targetAnimationSetIndex, PrimitiveTypes::UInt32 targetAnimationIndex)
{
    Handle hParentSkinInstance = getFirstParentByType<SkeletonInstance>();
    PEASSERT( expr: hParentSkinInstance.isValid(),  format: "SM has to belong to skeleton instance");

    SkeletonInstance *pSkelInstance = hParentSkinInstance.getObject<SkeletonInstance>();
    AnimSetBufferGPU *pAnimSetBufferGPUSource = pSkelInstance->m_hAnimationSetGPUs[sourceAnimationSetIndex].getObject<AnimSetBufferGPU>();
    AnimationSetCPU *pAnimSetSource = pAnimSetBufferGPUSource->m_hAnimationSetCPU.getObject<AnimationSetCPU>();
    AnimationCPU &animSource = pAnimSetSource->m_animations[(sourceAnimationIndex + m_debugAnimIdOffset)];

    AnimSetBufferGPU *pAnimSetBufferGPURef = pSkelInstance->m_hAnimationSetGPUs[referenceAnimationSetIndex].getObject<AnimSetBufferGPU>();
    AnimationSetCPU *pAnimSetRef = pAnimSetBufferGPURef->m_hAnimationSetCPU.getObject<AnimationSetCPU>();
    AnimationCPU &animRef = pAnimSetRef->m_animations[(referenceAnimationIndex + m_debugAnimIdOffset)];

    AnimationCPU additiveAnim( [&] *m_pContext, m_arena);

    if (animSource.m_frames.m_size < animRef.m_frames.m_size)
    {
        assert(animSource.m_frames.m_size < animRef.m_frames.m_size);
    }
    additiveAnim.m_frames.reset(animSource.m_frames.m_size);
    for (PrimitiveTypes::Int16 i = 0; i < animSource.m_frames.m_size; i++)
    {
        Array<TSQ> currentSourceFrame = animSource.m_frames[i];
        Array<TSQ> currentReferenceFrame = animRef.m_frames[i];
        Array<TSQ> additiveFrame;
        additiveFrame.reset(animSource.m_frames[i].m_size);
        for (PrimitiveTypes::Int16 j = 0; j < animSource.m_frames[i].m_size; j++)
        {
            TSQ sourcePos = currentSourceFrame[j];
            TSQ refPos = currentReferenceFrame[j];
```



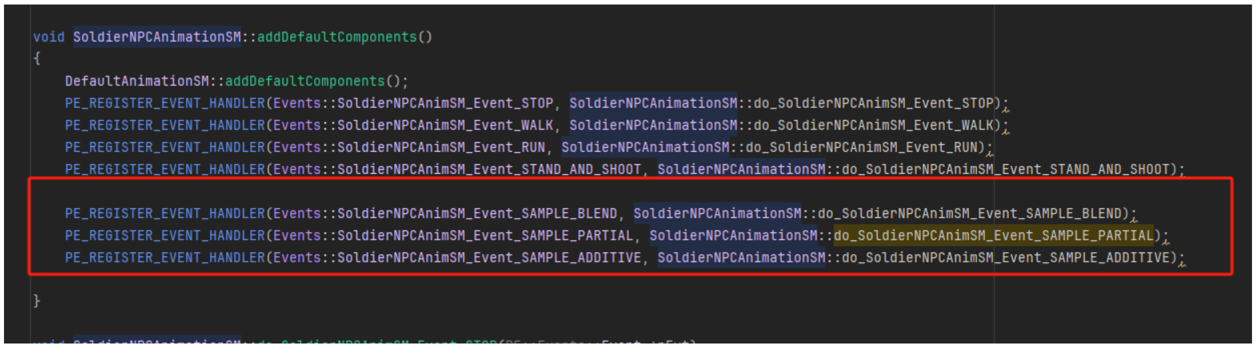
```cpp
    additiveAnim.m_frames.reset(animSource.m_frames.m_size);
    for (PrimitiveTypes::Int16 i = 0; i < animSource.m_frames.m_size; i++)
    {
        Array<TSQ> currentSourceFrame = animSource.m_frames[i];
        Array<TSQ> currentReferenceFrame = animRef.m_frames[i];
        Array<TSQ> additiveFrame;
        additiveFrame.reset(animSource.m_frames[i].m_size);
        for (PrimitiveTypes::Int16 j = 0; j < animSource.m_frames[i].m_size; j++)
        {
            TSQ sourcePos = currentSourceFrame[j];
            TSQ refPos = currentReferenceFrame[j];

            Matrix4x4 sourceTransform = sourcePos.createMatrix();
            Matrix4x4 refTransform = refPos.createMatrix();
            Matrix4x4 additiveTransform = sourceTransform * refTransform.inverse();
            additiveFrame.add( val: TSQ(additiveTransform));
        }
        additiveAnim.m_frames.add(additiveFrame);
    }
    additiveAnim.m_numJoints = animSource.m_numJoints;
    additiveAnim.m_startJoint = animSource.m_startJoint;
    additiveAnim.m_endJoint = animSource.m_endJoint;

    m_additiveAnim = &additiveAnim;

    AnimSetBufferGPU *pAnimSetBufferGPUTarget = pSkelInstance->m_hAnimationSetGPUs[targetAnimationSetIndex].getObject<AnimSetBufferGPU>();
    AnimationSetCPU *pAnimSetTarget = pAnimSetBufferGPUTarget->m_hAnimationSetCPU.getObject<AnimationSetCPU>();
    AnimationCPU &animTarget = pAnimSetTarget->m_animations[(targetAnimationIndex + m_debugAnimIdOffset)];

    AnimationCPU targetAdditiveAnim( [&] *m_pContext, m_arena);
    targetAdditiveAnim.m_frames.reset(animTarget.m_frames.m_size);
```



```cpp
    AnimationSetCPU *pAnimSetTarget = pAnimSetBufferCPUTarget->m_hAnimationSetCPU.getObject<AnimationSetCPU>();
    AnimationCPU &animTarget = pAnimSetTarget->m_animations[(targetAnimationIndex + m_debugAnimIdOffset)];

    AnimationCPU targetAdditiveAnim(*m_pContext, m_arena);
    targetAdditiveAnim.m_frames.reset(animTarget.m_frames.m_size);

    for (PrimitiveTypes::Int16 i = 0; i < animTarget.m_frames.m_size; i++)
    {
        Array<TSQ> targetFrame = animTarget.m_frames[i];
        float percentage = i / animTarget.m_frames.m_size;
        Array<TSQ> additiveFrame = additiveAnim.m_frames[floor(percentage * additiveAnim.m_frames.m_size)];
        Array<TSQ> finalFrame;
        finalFrame.reset(animTarget.m_frames[i].m_size);
        for (PrimitiveTypes::Int16 j = 0; j < animTarget.m_frames[i].m_size; j++)
        {
            TSQ targetPos = targetFrame[j];
            TSQ additivePos = additiveFrame[j];

            Matrix4x4 targetTransform = targetPos.createMatrix();
            Matrix4x4 additiveTransform = additivePos.createMatrix();
            Matrix4x4 finalTransform = additiveTransform * targetTransform;
            finalFrame.add(TSQ(finalTransform));
        }
        targetAdditiveAnim.m_frames.add(finalFrame);
    }
    pAnimSetTarget->m_animations.reset(pAnimSetTarget->m_animations.m_size + 1, true);
    pAnimSetTarget->m_animations.add(targetAdditiveAnim);
}
```

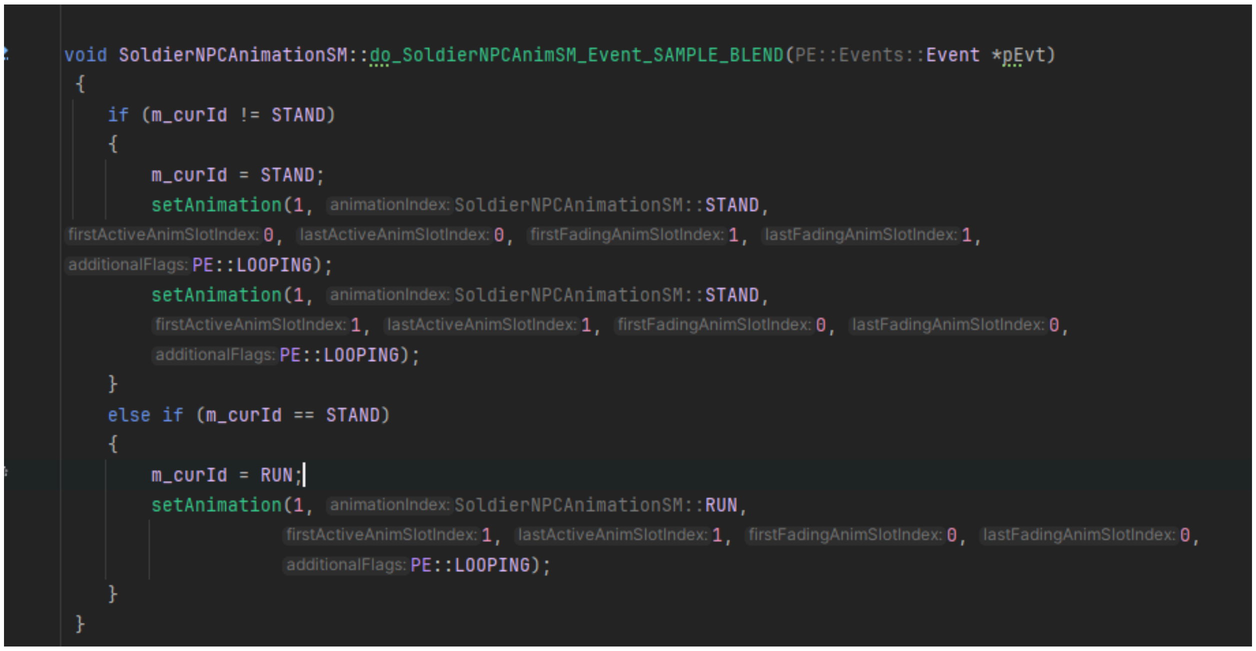
We add the new animation to the target animation set.

Then we modify SoldierNPCAnimationSM. I added three different events for 3 vampires.



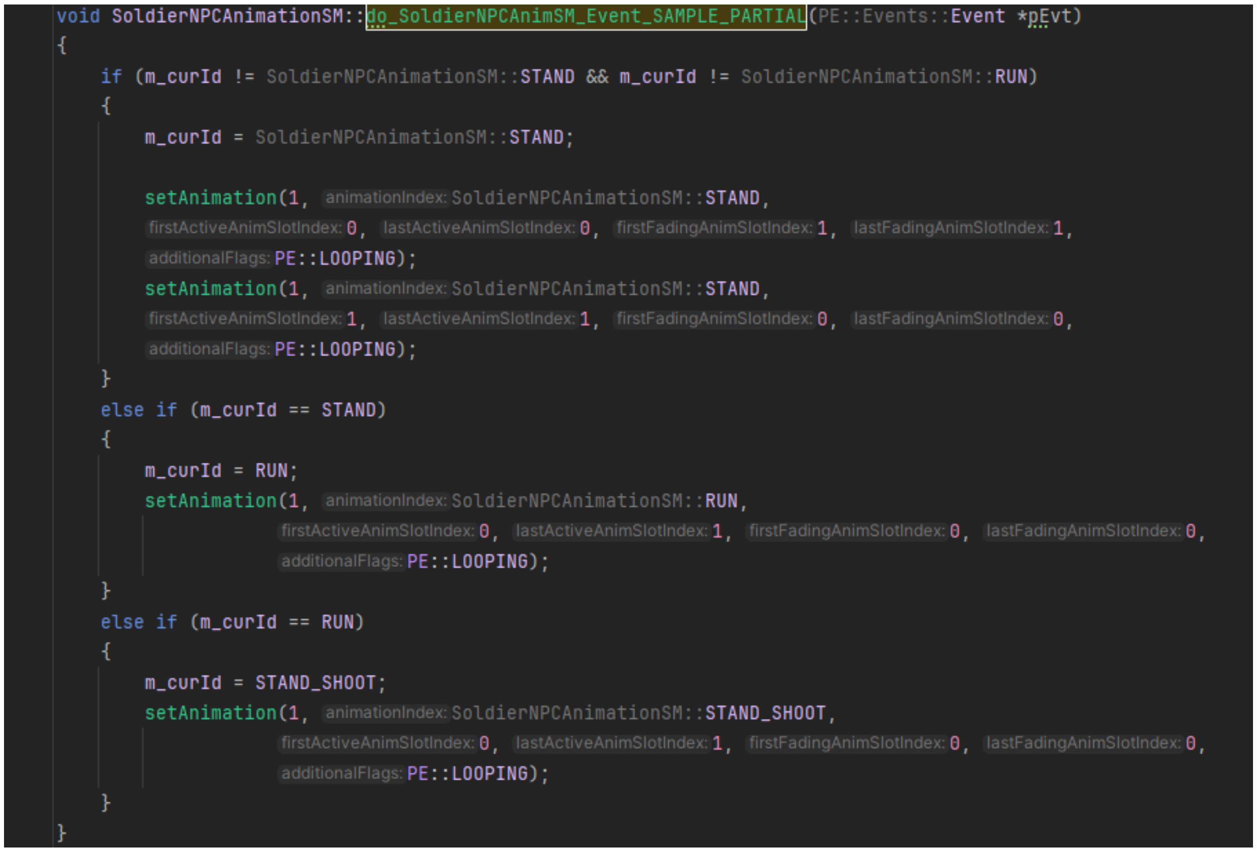
```cpp
void SoldierNPCAnimationSM::addDefaultComponents()
{
    DefaultAnimationSM::addDefaultComponents();
    PE_REGISTER_EVENT_HANDLER(Events::SoldierNPCAnimSM_Event_STOP, SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_STOP);
    PE_REGISTER_EVENT_HANDLER(Events::SoldierNPCAnimSM_Event_WALK, SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_WALK);
    PE_REGISTER_EVENT_HANDLER(Events::SoldierNPCAnimSM_Event_RUN, SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_RUN);
    PE_REGISTER_EVENT_HANDLER(Events::SoldierNPCAnimSM_Event_STAND_AND_SHOOT, SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_STAND_AND_SHOOT);

    PE_REGISTER_EVENT_HANDLER(Events::SoldierNPCAnimSM_Event_SAMPLE_BLEND, SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_SAMPLE_BLEND);
    PE_REGISTER_EVENT_HANDLER(Events::SoldierNPCAnimSM_Event_SAMPLE_PARTIAL, SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_SAMPLE_PARTIAL);
    PE_REGISTER_EVENT_HANDLER(Events::SoldierNPCAnimSM_Event_SAMPLE_ADDITIVE, SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_SAMPLE_ADDITIVE);

}

void SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_STOP(PE::Events::Event *pEvt)
```

For blend case, I blend only between idle and run



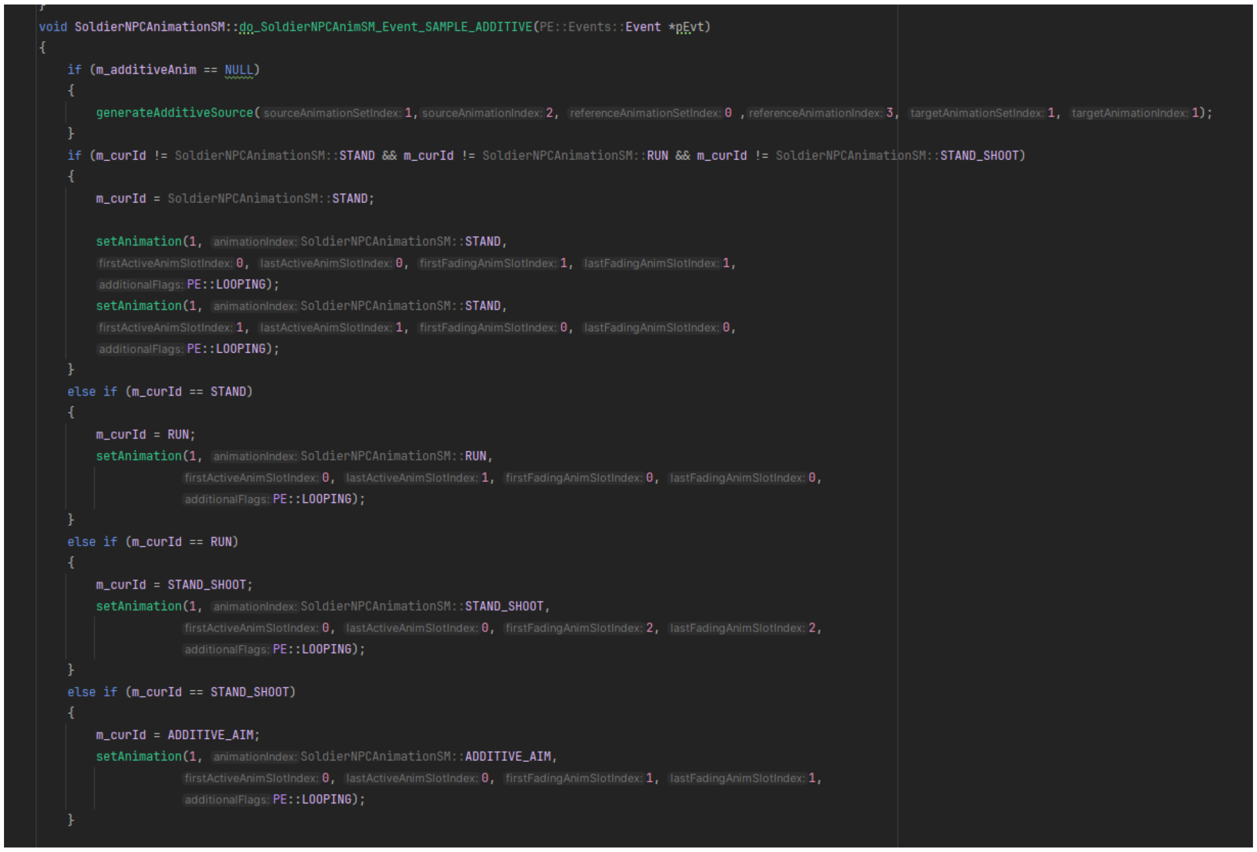
```
void SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_SAMPLE_BLEND(PE::Events::Event *pEvt)
{
    if (m_curId != STAND)
    {
        m_curId = STAND;
        setAnimation(1, animationIndex: SoldierNPCAnimationSM::STAND,
firstActiveAnimSlotIndex: 0, lastActiveAnimSlotIndex: 0, firstFadingAnimSlotIndex: 1, lastFadingAnimSlotIndex: 1,
additionalFlags: PE::LOOPING);
        setAnimation(1, animationIndex: SoldierNPCAnimationSM::STAND,
        firstActiveAnimSlotIndex: 1, lastActiveAnimSlotIndex: 1, firstFadingAnimSlotIndex: 0, lastFadingAnimSlotIndex: 0,
        additionalFlags: PE::LOOPING);
    }
    else if (m_curId == STAND)
    {
        m_curId = RUN;
        setAnimation(1, animationIndex: SoldierNPCAnimationSM::RUN,
                     firstActiveAnimSlotIndex: 1, lastActiveAnimSlotIndex: 1, firstFadingAnimSlotIndex: 0, lastFadingAnimSlotIndex: 0,
                     additionalFlags: PE::LOOPING);
    }
}
```

For Partial blending, I blend between Idle, Run, and aim, where aim is the partial animation



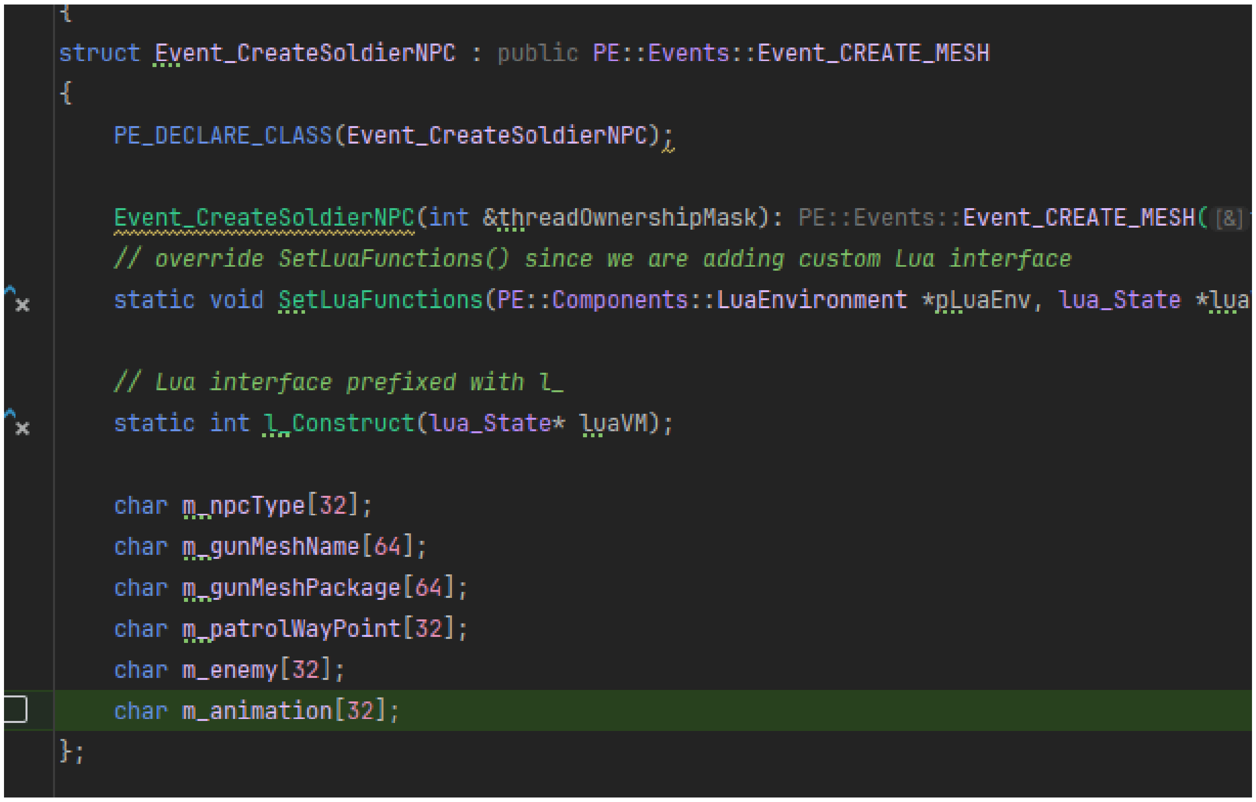
```
void SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_SAMPLE_PARTIAL(PE::Events::Event *pEvt)
{
    if (m_curId != SoldierNPCAnimationSM::STAND && m_curId != SoldierNPCAnimationSM::RUN)
    {
        m_curId = SoldierNPCAnimationSM::STAND;

        setAnimation(1, animationIndex: SoldierNPCAnimationSM::STAND,
        firstActiveAnimSlotIndex: 0, lastActiveAnimSlotIndex: 0, firstFadingAnimSlotIndex: 1, lastFadingAnimSlotIndex: 1,
        additionalFlags: PE::LOOPING);
        setAnimation(1, animationIndex: SoldierNPCAnimationSM::STAND,
        firstActiveAnimSlotIndex: 1, lastActiveAnimSlotIndex: 1, firstFadingAnimSlotIndex: 0, lastFadingAnimSlotIndex: 0,
        additionalFlags: PE::LOOPING);
    }
    else if (m_curId == STAND)
    {
        m_curId = RUN;
        setAnimation(1, animationIndex: SoldierNPCAnimationSM::RUN,
                     firstActiveAnimSlotIndex: 0, lastActiveAnimSlotIndex: 1, firstFadingAnimSlotIndex: 0, lastFadingAnimSlotIndex: 0,
                     additionalFlags: PE::LOOPING);
    }
    else if (m_curId == RUN)
    {
        m_curId = STAND_SHOOT;
        setAnimation(1, animationIndex: SoldierNPCAnimationSM::STAND_SHOOT,
                     firstActiveAnimSlotIndex: 0, lastActiveAnimSlotIndex: 1, firstFadingAnimSlotIndex: 0, lastFadingAnimSlotIndex: 0,
                     additionalFlags: PE::LOOPING);
    }
}
```

For additive blending, I blended between Idle, Run, run and aim, and the additive (aim(action pack) - idle(adventure pack)) + idle(action pack)



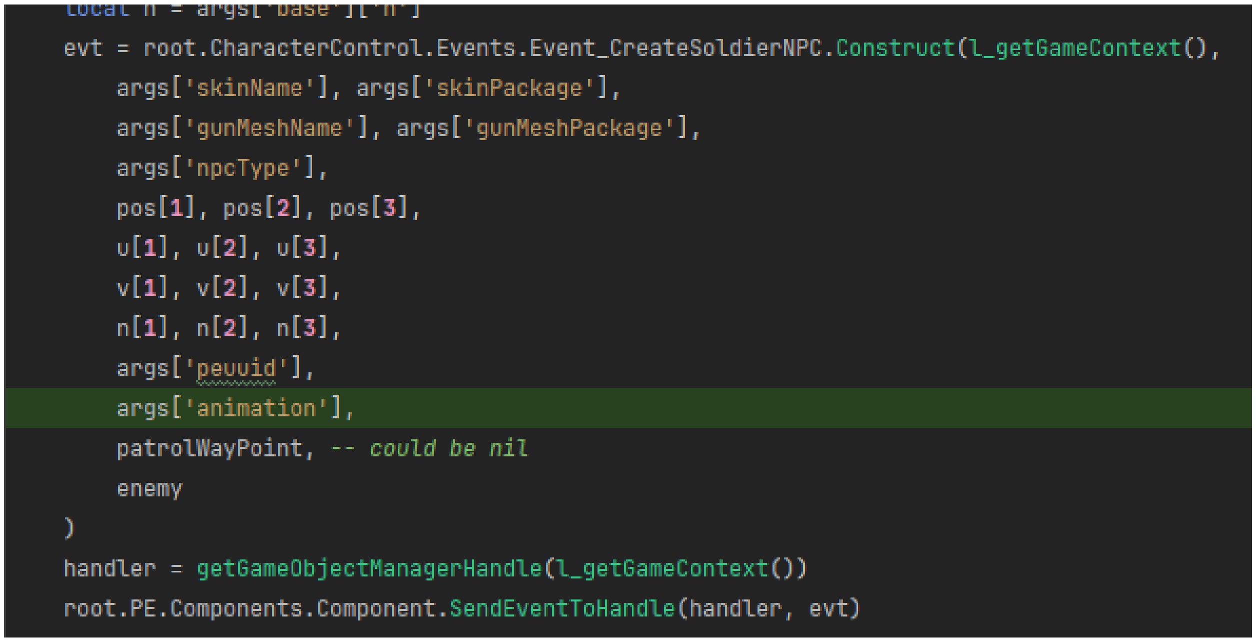
```cpp
void SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_SAMPLE_ADDITIVE(PE::Events::Event *pEvt)
{
    if (m_additiveAnim == NULL)
    {
        generateAdditiveSource( sourceAnimationSetIndex: 1, sourceAnimationIndex: 2,  referenceAnimationSetIndex: 0  , referenceAnimationIndex: 3,  targetAnimationSetIndex: 1,  targetAnimationIndex: 1);
    }
    if (m_curId != SoldierNPCAnimationSM::STAND && m_curId != SoldierNPCAnimationSM::RUN && m_curId != SoldierNPCAnimationSM::STAND_SHOOT)
    {
        m_curId = SoldierNPCAnimationSM::STAND;

        setAnimation(1,  animationIndex: SoldierNPCAnimationSM::STAND,
        firstActiveAnimSlotIndex: 0,  lastActiveAnimSlotIndex: 0,  firstFadingAnimSlotIndex: 1,  lastFadingAnimSlotIndex: 1,
        additionalFlags: PE::LOOPING);
        setAnimation(1,  animationIndex: SoldierNPCAnimationSM::STAND,
        firstActiveAnimSlotIndex: 1,  lastActiveAnimSlotIndex: 1,  firstFadingAnimSlotIndex: 0,  lastFadingAnimSlotIndex: 0,
        additionalFlags: PE::LOOPING);
    }
    else if (m_curId == STAND)
    {
        m_curId = RUN;
        setAnimation(1,  animationIndex: SoldierNPCAnimationSM::RUN,
                    firstActiveAnimSlotIndex: 0,  lastActiveAnimSlotIndex: 1,  firstFadingAnimSlotIndex: 0,  lastFadingAnimSlotIndex: 0,
                    additionalFlags: PE::LOOPING);
    }
    else if (m_curId == RUN)
    {
        m_curId = STAND_SHOOT;
        setAnimation(1,  animationIndex: SoldierNPCAnimationSM::STAND_SHOOT,
                    firstActiveAnimSlotIndex: 0,  lastActiveAnimSlotIndex: 0,  firstFadingAnimSlotIndex: 2,  lastFadingAnimSlotIndex: 2,
                    additionalFlags: PE::LOOPING);
    }
    else if (m_curId == STAND_SHOOT)
    {
        m_curId = ADDITIVE_AIM;
        setAnimation(1,  animationIndex: SoldierNPCAnimationSM::ADDITIVE_AIM,
                    firstActiveAnimSlotIndex: 0,  lastActiveAnimSlotIndex: 0,  firstFadingAnimSlotIndex: 1,  lastFadingAnimSlotIndex: 1,
                    additionalFlags: PE::LOOPING);
    }
```

At last, I added an animation parameter, that helps us specify which soldier uses which type of blending method.

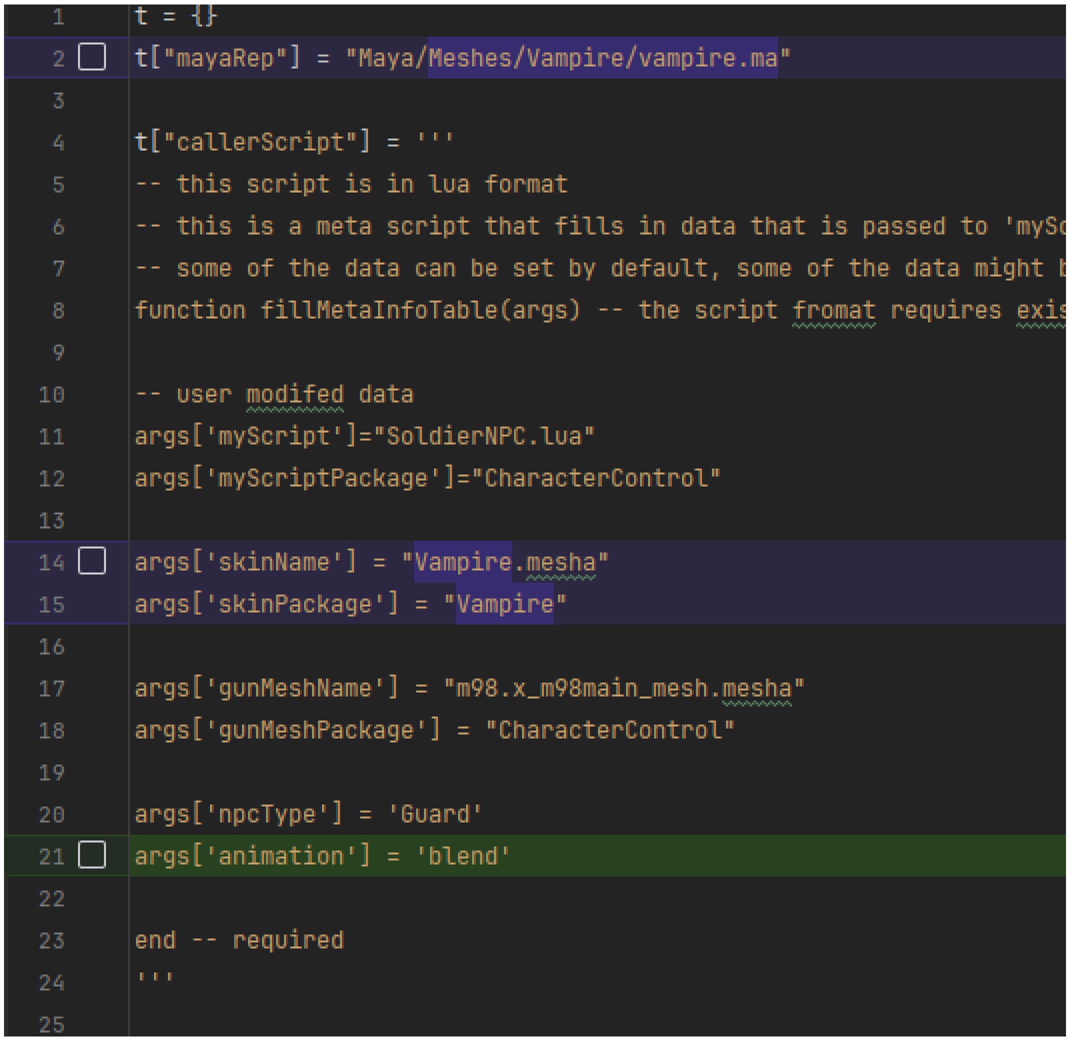


```
{
struct Event_CreateSoldierNPC : public PE::Events::Event_CREATE_MESH
{
    PE_DECLARE_CLASS(Event_CreateSoldierNPC);

    Event_CreateSoldierNPC(int &threadOwnershipMask): PE::Events::Event_CREATE_MESH(
    // override SetLuaFunctions() since we are adding custom Lua interface
    static void SetLuaFunctions(PE::Components::LuaEnvironment *pLuaEnv, lua_State *lua

    // Lua interface prefixed with l_
    static int l_Construct(lua_State* luaVM);

    char m_npcType[32];
    char m_gunMeshName[64];
    char m_gunMeshPackage[64];
    char m_patrolWayPoint[32];
    char m_enemy[32];
    char m_animation[32];
};
```

Set the Lua script to contain the new parameter

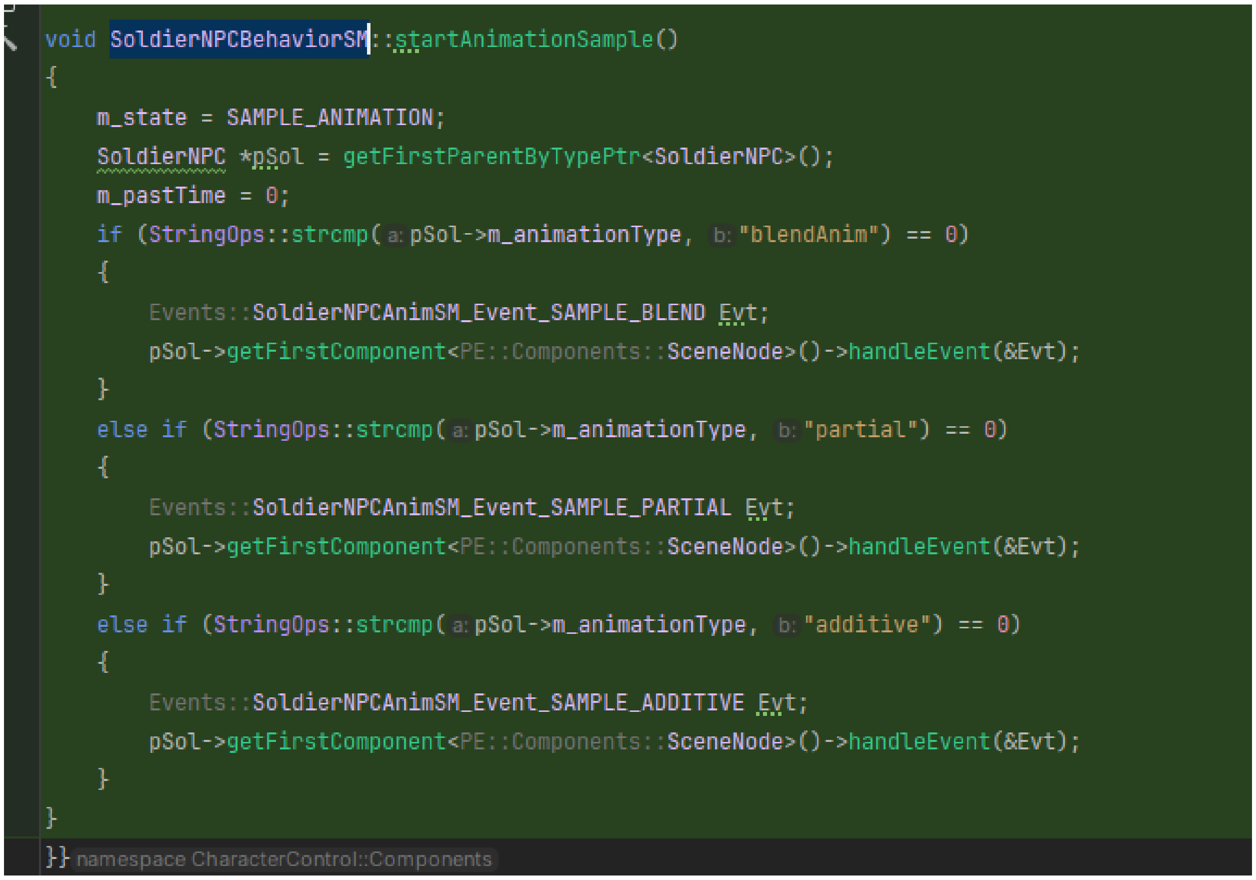


```
local n = args['base']['n']
evt = root.CharacterControl.Events.Event_CreateSoldierNPC.Construct(l_getGameContext(),
    args['skinName'], args['skinPackage'],
    args['gunMeshName'], args['gunMeshPackage'],
    args['npcType'],
    pos[1], pos[2], pos[3],
    u[1], u[2], u[3],
    v[1], v[2], v[3],
    n[1], n[2], n[3],
    args['peuuid'],
    args['animation'],
    patrolWayPoint, -- could be nil
    enemy
)
handler = getGameObjectManagerHandle(l_getGameContext())
root.PE.Components.Component.SendEventToHandle(handler, evt)
```



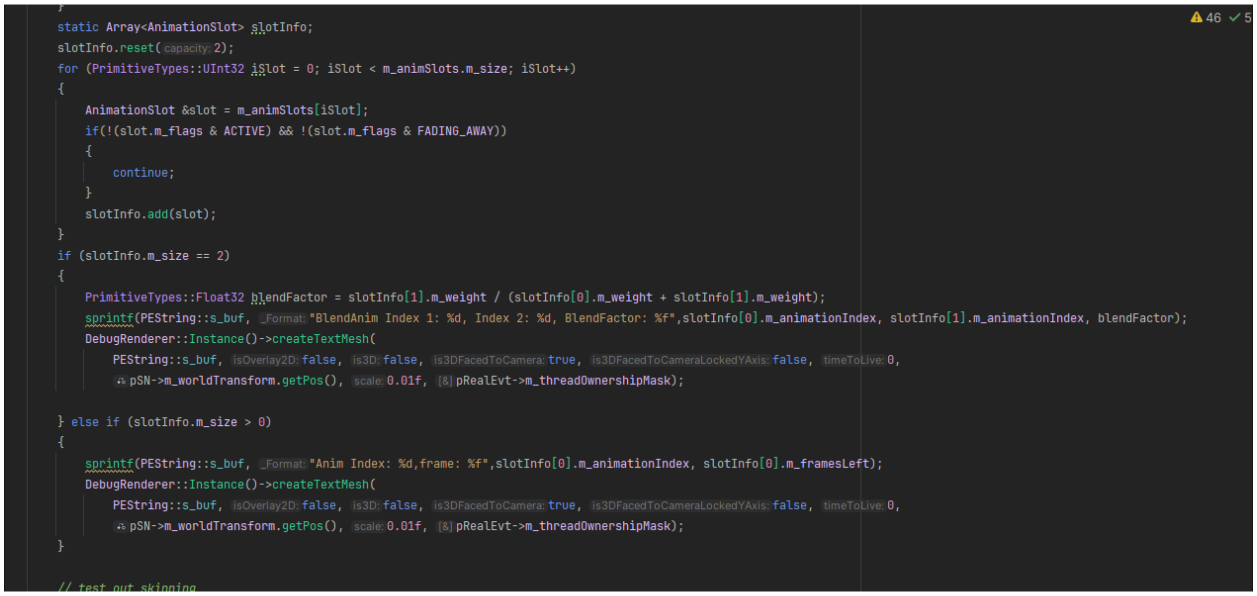
```
t = {}
t["mayaRep"] = "Maya/Meshes/Vampire/vampire.ma"

t["callerScript"] = '''
-- this script is in lua format
-- this is a meta script that fills in data that is passed to 'mySc
-- some of the data can be set by default, some of the data might b
function fillMetaInfoTable(args) -- the script fromat requires exis

-- user modifed data
args['myScript']="SoldierNPC.lua"
args['myScriptPackage']="CharacterControl"

args['skinName'] = "Vampire.mesha"
args['skinPackage'] = "Vampire"

args['gunMeshName'] = "m98.x_m98main_mesh.mesha"
args['gunMeshPackage'] = "CharacterControl"

args['npcType'] = 'Guard'
args['animation'] = 'blend'

end -- required
'''

```

Deal with different cases when starting SoldierNPCBehaviorSM.



```cpp
void SoldierNPCBehaviorSM::startAnimationSample()
{
    m_state = SAMPLE_ANIMATION;
    SoldierNPC *pSol = getFirstParentByTypePtr<SoldierNPC>();
    m_pastTime = 0;
    if (StringOps::strcmp( a: pSol->m_animationType,  b: "blendAnim") == 0)
    {
        Events::SoldierNPCAnimSM_Event_SAMPLE_BLEND Evt;
        pSol->getFirstComponent<PE::Components::SceneNode>()->handleEvent(&Evt);
    }
    else if (StringOps::strcmp( a: pSol->m_animationType,  b: "partial") == 0)
    {
        Events::SoldierNPCAnimSM_Event_SAMPLE_PARTIAL Evt;
        pSol->getFirstComponent<PE::Components::SceneNode>()->handleEvent(&Evt);
    }
    else if (StringOps::strcmp( a: pSol->m_animationType,  b: "additive") == 0)
    {
        Events::SoldierNPCAnimSM_Event_SAMPLE_ADDITIVE Evt;
        pSol->getFirstComponent<PE::Components::SceneNode>()->handleEvent(&Evt);
    }
}
}} namespace CharacterControl::Components
```

In order to display the information, I created different text meshes when there is 1 animation in the slot or if there are 2 animations blending with each other.



```cpp
    static Array<AnimationSlot> slotInfo;
    slotInfo.reset( capacity: 2);
    for (PrimitiveTypes::UInt32 iSlot = 0; iSlot < m_animSlots.m_size; iSlot++)
    {
        AnimationSlot &slot = m_animSlots[iSlot];
        if(!(slot.m_flags & ACTIVE) && !(slot.m_flags & FADING_AWAY))
        {
            continue;
        }
        slotInfo.add(slot);
    }
    if (slotInfo.m_size == 2)
    {
        PrimitiveTypes::Float32 blendFactor = slotInfo[1].m_weight / (slotInfo[0].m_weight + slotInfo[1].m_weight);
        sprintf(PEString::s_buf, _Format: "BlendAnim Index 1: %d, Index 2: %d, BlendFactor: %f",slotInfo[0].m_animationIndex, slotInfo[1].m_animationIndex, blendFactor);
        DebugRenderer::Instance()->createTextMesh(
            PEString::s_buf,  isOverlay2D: false,  is3D: false,  is3DFacedToCamera: true,  is3DFacedToCameraLockedYAxis: false,  timeToLive: 0,
            pSN->m_worldTransform.getPos(),  scale: 0.01f,  [&] pRealEvt->m_threadOwnershipMask);

    } else if (slotInfo.m_size > 0)
    {
        sprintf(PEString::s_buf, _Format: "Anim Index: %d,frame: %f",slotInfo[0].m_animationIndex, slotInfo[0].m_framesLeft);
        DebugRenderer::Instance()->createTextMesh(
            PEString::s_buf,  isOverlay2D: false,  is3D: false,  is3DFacedToCamera: true,  is3DFacedToCameraLockedYAxis: false,  timeToLive: 0,
            pSN->m_worldTransform.getPos(),  scale: 0.01f,  [&] pRealEvt->m_threadOwnershipMask);
    }

    // test out skinning
```