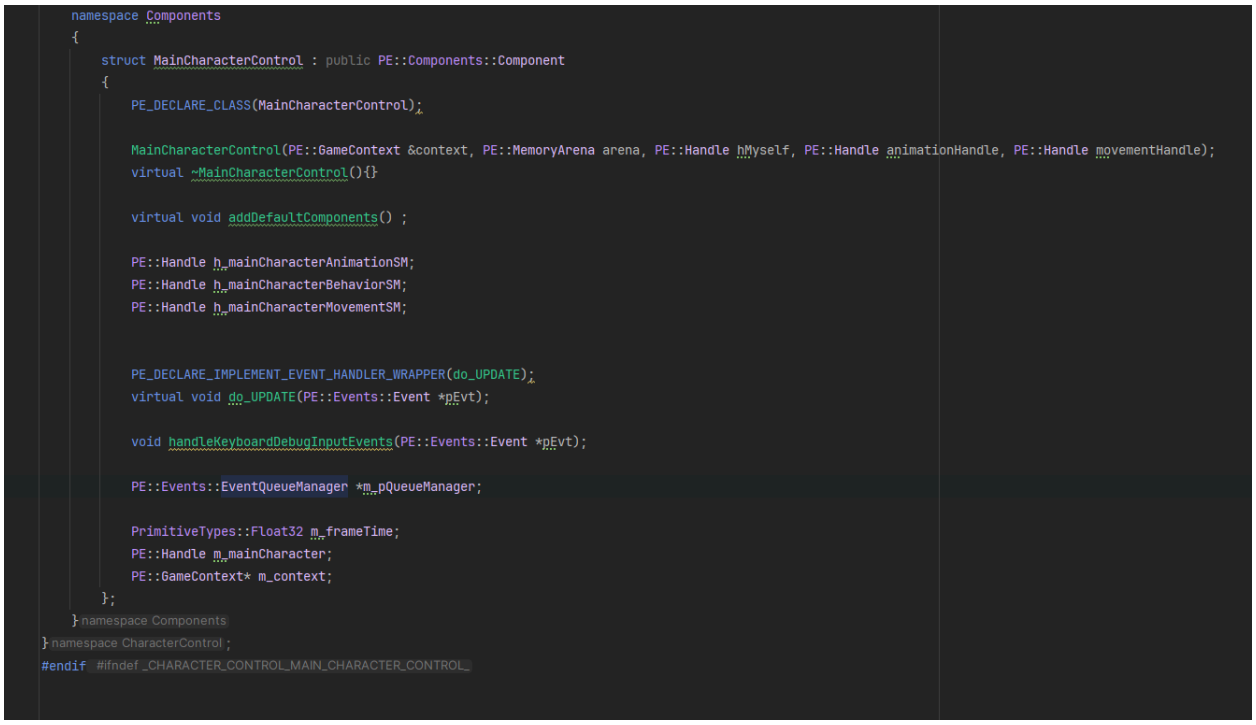
VideoLink:
Shooting bullet with a straight line
https://drive.google.com/file/d/1ezbHQCgFuYzI1t3D42VZU5_nPn8tQscY/view?usp=drive_link
Shooting bullet with a scattered butter path
https://drive.google.com/file/d/1HoLNf8oNZvCWxCg_4NeqQ4Lc7ohl0V41/view?usp=drive_link

I implemented a basic gameplay system, where the camera follows the player, and the player can control the main character by pressing WASD. The first thing I implemented was setting up a MainCharacterController struct where it takes the input event of WASD, mouse click, and ESC, and passes it to the state machines (both movement state machine and the animation state machine) of the main character. This controller also resets the state machines when no input event is coming.



```
    namespace Components
    {
        struct MainCharacterControl : public PE::Components::Component
        {
            PE_DECLARE_CLASS(MainCharacterControl);

            MainCharacterControl(PE::GameContext &context, PE::MemoryArena arena, PE::Handle hMyself, PE::Handle animationHandle, PE::Handle movementHandle);
            virtual ~MainCharacterControl(){}

            virtual void addDefaultComponents() ;

            PE::Handle h_mainCharacterAnimationSM;
            PE::Handle h_mainCharacterBehaviorSM;
            PE::Handle h_mainCharacterMovementSM;


            PE_DECLARE_IMPLEMENT_EVENT_HANDLER_WRAPPER(do_UPDATE);
            virtual void do_UPDATE(PE::Events::Event *pEvt);

            void handleKeyboardDebugInputEvents(PE::Events::Event *pEvt);

            PE::Events::EventQueueManager *m_pQueueManager;

            PrimitiveTypes::Float32 m_frameTime;
            PE::Handle m_mainCharacter;
            PE::GameContext* m_context;
        };
    }namespace Components
}namespace CharacterControl ;
#endif  #ifndef _CHARACTER_CONTROL_MAIN_CHARACTER_CONTROL_
```

```cpp
void MainCharacterControl::do_UPDATE(PE::Events::Event *pEvt)
{
    // Process input events (controller  buttons, triggers...)
    PE::Handle iqh = PE::Events::EventQueueManager::Instance()->getEventQueueHandle("input");

    // Process input event -> game event conversion
    if (iqh.getObject<PE::Events::EventQueue>()->empty())
    {
        SoldierNPCAnimationSM* animSM = h_mainCharacterAnimationSM.getObject<SoldierNPCAnimationSM>();
        SoldierNPCMovementSM* movementSM = h_mainCharacterMovementSM.getObject<SoldierNPCMovementSM>();

        PE::Handle hAnim( dbgName: "ANIMATION_STOP_EVENT",  neededSize: sizeof(SoldierNPCAnimSM_Event_STOP));
        PE::Handle h( dbgName: "MOVEMENT_EVENT",  neededSize: sizeof(SoldierNPCMovementSM_Event_MOVE));

        SoldierNPCAnimSM_Event_STOP* animEvt = new(hAnim) SoldierNPCAnimSM_Event_STOP();
        SoldierNPCMovementSM_Event_STOP* evt = new(h) SoldierNPCMovementSM_Event_STOP();

        animSM->handleEvent(animEvt);
        movementSM->handleEvent(evt);

        hAnim.release();
        h.release();
    }
```

```cpp
void MainCharacterControl::handleKeyboardDebugInputEvents(Event *pEvt)
{
    m_pQueueManager = PE::Events::EventQueueManager::Instance();
    if (Event_KEY_W_HELD::GetClassId() == pEvt->getClassId())
    {
        if(m_context->m_paused) return;

        // PE::Handle h("EVENT", sizeof(Event_FLY_CAMERA));
              //
        // Event_FLY_CAMERA *flyCameraEvt = new(h) Event_FLY_CAMERA ;
              //
        // Vector3 relativeMovement(0.0f,0.0f,1.0f);
        // flyCameraEvt->m_relativeMove = relativeMovement * Debug_Fly_Speed * m_frameTime;
        // m_pQueueManager->add(h, QT_GENERAL);
        SoldierNPCMovementSM* movementSM = h_mainCharacterMovementSM.getObject<SoldierNPCMovementSM>();
        SoldierNPCAnimationSM* animSM = h_mainCharacterAnimationSM.getObject<SoldierNPCAnimationSM>();

        PE::Handle h( dbgName: "MOVEMENT_EVENT",  neededSize: sizeof(SoldierNPCMovementSM_Event_MOVE));
        PE::Handle hAnim( dbgName: "ANIMATION_EVENT",  neededSize: sizeof(SoldierNPCAnimSM_Event_RUN));

        SoldierNPCMovementSM_Event_MOVE* moveEvent = new(h) SoldierNPCMovementSM_Event_MOVE();
        SoldierNPCAnimSM_Event_RUN* animEvent = new(hAnim) SoldierNPCAnimSM_Event_RUN();

        moveEvent->m_direction = SoldierNPCMovementSM_Event_MOVE::FRONT;
        moveEvent->m_displacement =  main_character_move_speed * m_frameTime;
        movementSM->handleEvent(moveEvent);
        animSM->handleEvent(animEvent);
        h.release();
```

Here I set the pause and show menu event to be triggered by ESC

```
    else if (Event_KEY_ESC_PRESSED::GetClassId() == pEvt->getClassId())
    {
        PE::Handle h( dbgName: "EVENT", neededSize: sizeof(Event_PAUSE));

        Event_PAUSE *pauseEvt = new(h) Event_PAUSE();

        m_pQueueManager->add( h, queueType: QT_GENERAL);

        PE::Handle hUI( dbgName: "EVENT", neededSize: sizeof(Event_JUMP_MENU));
        Event_JUMP_MENU* pEvt = new(hUI) Event_JUMP_MENU();
        pEvt->m_name = "0";
        m_pQueueManager->add( hUI, queueType: QT_GENERAL);
    }
```

I also added a follow method for CameraSceneNode that resets the camera position to the followed object during every frame.

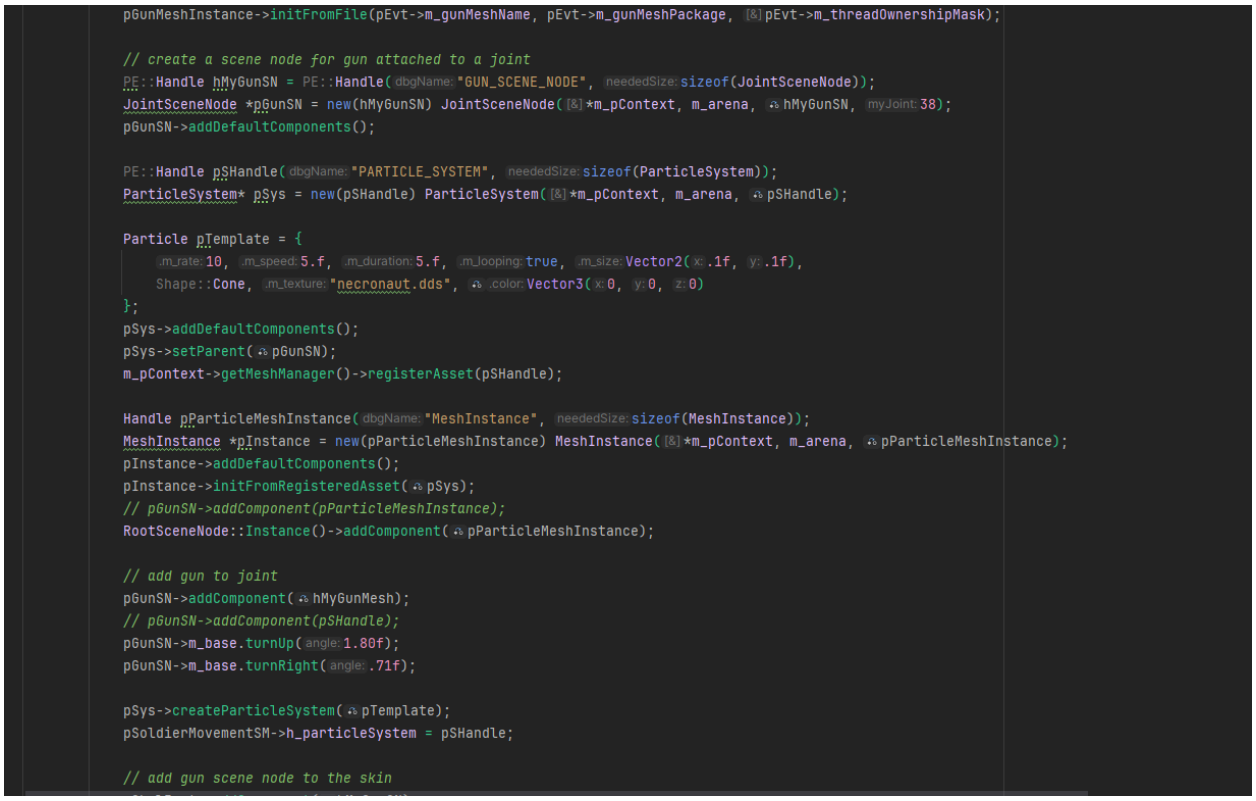
```
void CameraSceneNode::setMainCharacter(Handle mainCharacterSceneNodeHandle)
{
    m_hfocusedSceneNode = mainCharacterSceneNodeHandle;
}
```

```
void CameraSceneNode::do_CALCULATE_TRANSFORMATIONS(Events::Event *pEvt)
{
    Handle hParentSN = getFirstParentByType<SceneNode>();
    if (m_ifFollow && m_hfocusedSceneNode.isValid())
    {
        if (!m_isInitialized)
        {
            SceneNode* followedSN = m_hfocusedSceneNode.getObject<SceneNode>();
            Matrix4x4 followedMat = followedSN->m_worldTransform;
            Vector3 followedPos = followedMat.getPos();
            Vector3 frontVec = followedMat.getN();
            Vector3 upVec = followedMat.getV();
            Vector3 rightVec = followedMat.getU();


            m_base.setPos( p: followedPos - frontVec * 2 + upVec * 2);
            m_base.setU( rightVec);
            m_base.setV( upVec);
            m_base.setN( frontVec);

            m_isInitialized = true;
        } else
        {
            SceneNode* followedSN = m_hfocusedSceneNode.getObject<SceneNode>();
            Matrix4x4 followedMat = followedSN->m_worldTransform;
            Vector3 followedPos = followedMat.getPos();
            Vector3 frontVec = m_base.getN();
            Vector3 upVec = m_base.getV();
            m_base.setPos( p: followedPos - frontVec * 2 + upVec * 2);
            // m_base.turnTo(followedPos + Vector3(0, 2, 0), false);
            // m_base.setV(upVec);

        }

    }
```

For the main character creation, I modified SoldierNPC.cpp to take the keyword “main” and set the camera to follow the SceneNode. Also I assign the state machines created here to the MainCharacterController

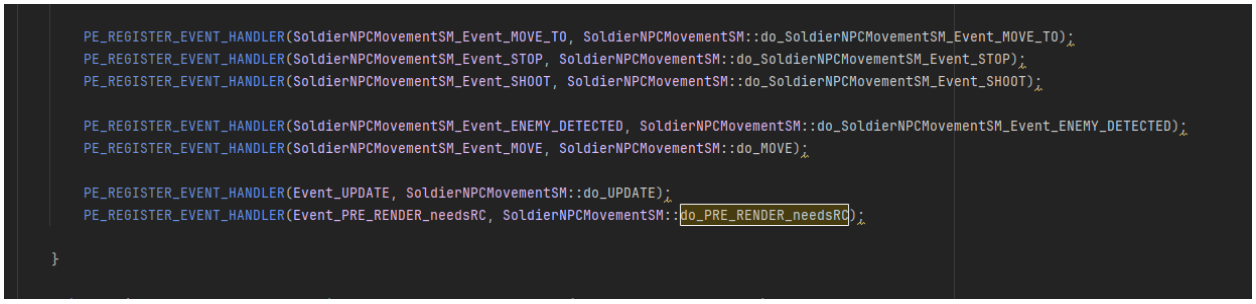
```
if (strcmp(m_npcType, "main") == 0)
{
    CameraSceneNode* cameraSN = CameraManager::Instance()->getActiveCamera()->getCamSceneNode();
    cameraSN->setMainCharacter( hSN);
    Handle h_mainCharacterControl = Handle( dbgName: "MAIN_CHARACTER_CONTROL",  neededSize: sizeof(MainCharacterControl));

    MainCharacterControl* mainCharacterControl = new(h_mainCharacterControl) MainCharacterControl( [&] context, arena,  hMyself: h_mainCharacterControl,
    mainCharacterControl->addDefaultComponents();
    RootSceneNode::Instance()->addComponent( h_mainCharacterControl);
} else
{
```

I also added the particle system and set the parent to the gun SceneNode so that particles will generate from the gun.



```
pGunMeshInstance->initFromFile(pEvt->m_gunMeshName, pEvt->m_gunMeshPackage, &pEvt->m_threadOwnershipMask);

// create a scene node for gun attached to a joint
PE::Handle hMyGunSN = PE::Handle(dbgName: "GUN_SCENE_NODE", neededSize: sizeof(JointSceneNode));
JointSceneNode *pGunSN = new(hMyGunSN) JointSceneNode(&*m_pContext, m_arena, hMyGunSN, myJoint: 38);
pGunSN->addDefaultComponents();

PE::Handle pSHandle(dbgName: "PARTICLE_SYSTEM", neededSize: sizeof(ParticleSystem));
ParticleSystem* pSys = new(pSHandle) ParticleSystem(&*m_pContext, m_arena, pSHandle);

Particle pTemplate = {
    .m_rate: 10, .m_speed: 5.f, .m_duration: 5.f, .m_looping: true, .m_size: Vector2(x: .1f, y: .1f),
    Shape::Cone, .m_texture: "necronaut.dds", .color: Vector3(x: 0, y: 0, z: 0)
};
pSys->addDefaultComponents();
pSys->setParent(pGunSN);
m_pContext->getMeshManager()->registerAsset(pSHandle);

Handle pParticleMeshInstance(dbgName: "MeshInstance", neededSize: sizeof(MeshInstance));
MeshInstance *pInstance = new(pParticleMeshInstance) MeshInstance(&*m_pContext, m_arena, pParticleMeshInstance);
pInstance->addDefaultComponents();
pInstance->initFromRegisteredAsset(pSys);
// pGunSN->addComponent(pParticleMeshInstance);
RootSceneNode::Instance()->addComponent(pParticleMeshInstance);

// add gun to joint
pGunSN->addComponent(hMyGunMesh);
// pGunSN->addComponent(pSHandle);
pGunSN->m_base.turnUp(angle: 1.80f);
pGunSN->m_base.turnRight(angle: .71f);

pSys->createParticleSystem(pTemplate);
pSoldierMovementSM->h_particleSystem = pSHandle;

// add gun scene node to the skin
```

In the MovementStateMachine, I added a custom shooting event, moving event to handle the input from MainCharacterController, and a pre render event to monitor the update of particle system.



```
    PE_REGISTER_EVENT_HANDLER(SoldierNPCMovementSM_Event_MOVE_TO, SoldierNPCMovementSM::do_SoldierNPCMovementSM_Event_MOVE_TO);
    PE_REGISTER_EVENT_HANDLER(SoldierNPCMovementSM_Event_STOP, SoldierNPCMovementSM::do_SoldierNPCMovementSM_Event_STOP);
    PE_REGISTER_EVENT_HANDLER(SoldierNPCMovementSM_Event_SHOOT, SoldierNPCMovementSM::do_SoldierNPCMovementSM_Event_SHOOT);

    PE_REGISTER_EVENT_HANDLER(SoldierNPCMovementSM_Event_ENEMY_DETECTED, SoldierNPCMovementSM::do_SoldierNPCMovementSM_Event_ENEMY_DETECTED);
    PE_REGISTER_EVENT_HANDLER(SoldierNPCMovementSM_Event_MOVE, SoldierNPCMovementSM::do_MOVE);

    PE_REGISTER_EVENT_HANDLER(Event_UPDATE, SoldierNPCMovementSM::do_UPDATE);
    PE_REGISTER_EVENT_HANDLER(Event_PRE_RENDER_needsRC, SoldierNPCMovementSM::do_PRE_RENDER_needsRC);

}
```

```cpp
void SoldierNPCMovementSM::do_MOVE(PE::Events::Event* pEvt)
{
    SoldierNPCMovementSM_Event_MOVE* pRealEvt = (SoldierNPCMovementSM_Event_MOVE*) pEvt;
    SoldierNPCMovementSM_Event_MOVE::Directions curDir = pRealEvt->m_direction;
    SceneNode* pSN = getParentsSceneNode();
    if (pSN)
    {
        CameraSceneNode *pcam = CameraManager::Instance()->getActiveCamera()->getCamSceneNode();
        Matrix4x4 cameraTransform = pcam->m_base;
        PrimitiveTypes::Float32 displacement = pRealEvt->m_displacement;
        m_state = RUNNING;
        if (curDir == SoldierNPCMovementSM_Event_MOVE::FRONT)
        {
            Vector3 frontDir = cameraTransform.getN();
            Vector3 newPos = pSN->m_base.getPos() + frontDir * displacement;
            pSN->m_base.setPos( newPos);
            pSN->m_base.turnInDirection( -frontDir, maxAngle: 3.14159f);
            pSN->m_base.turnLeft( angle: .7854f);

        } else if (curDir == SoldierNPCMovementSM_Event_MOVE::BACK)
        {
            Vector3 backDir = -cameraTransform.getN();
            Vector3 newPos = pSN->m_base.getPos() + backDir * displacement;
            pSN->m_base.setPos( newPos);
            pSN->m_base.turnInDirection( -backDir, maxAngle: 3.14159f);
            pSN->m_base.turnLeft( angle: .7854f);
        } else if (curDir == SoldierNPCMovementSM_Event_MOVE::LEFT)
        {
            Vector3 leftDir = -cameraTransform.getU();
            Vector3 newPos = pSN->m_base.getPos() + leftDir * displacement;
            pSN->m_base.setPos( newPos);
            pSN->m_base.turnInDirection( -leftDir, maxAngle: 3.14159f);
```

```cpp
void SoldierNPCMovementSM::do_SoldierNPCMovementSM_Event_SHOOT(Event* pEvt)
{
    if (m_state != SHOOTING)
    {
        m_state = SHOOTING;
        if (h_particleSystem.isValid())
        {
            ParticleSystem* pSys = h_particleSystem.getObject<ParticleSystem>();
            pSys->m_isActive = true;
        }
        // SceneNode* pSN = getParentsSceneNode();
        // pSN->m_base.turnRight(.7854f);
    }
}
```

Also we need to reset the state machine whenever no input is coming in.

```
void SoldierNPCMovementSM::do_SoldierNPCMovementSM_Event_STOP(PE::Events::Event* pEvt)
{
    if (m_state == STANDING)
    {
        return;
    }
    Events::SoldierNPCAnimSM_Event_STOP Evt;

    if (m_state == SHOOTING)
    {
        ParticleSystem* pSys = h_particleSystem.getObject<ParticleSystem>();
        pSys->m_isActive = false;
        pSys->reset();
    }
    SoldierNPC* pSol = getFirstParentByTypePtr<SoldierNPC>();
    pSol->getFirstComponent<PE::Components::SceneNode>()->handleEvent(&Evt);
    if (m_state == RUNNING)
    {
        SceneNode* pSN = getParentsSceneNode();
        pSN->m_base.turnRight( angle: .7854f);
    }
    m_state = STANDING;
}
```

For the animation state machine, I reset the animation index, and use those I need for my gameplay.

```
// for vampire demo
enum AnimId
{
    NONE = -1,
    STAND = 3,
    RUN = 0,
    STAND_SHOOT = 2,
};
```

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

void SoldierNPCAnimationSM::do_SoldierNPCAnimSM_Event_STAND_AND_SHOOT(PE::Events::Eve
{
    if (m_curId != SoldierNPCAnimationSM::STAND_SHOOT)
    {
        m_curId = SoldierNPCAnimationSM::STAND_SHOOT;
        setAnimation(1, animationIndex: SoldierNPCAnimationSM::STAND_SHOOT,
            firstActiveAnimSlotIndex: 0, lastActiveAnimSlotIndex: 1, firstFadingAnimSlotIndex: 0, lastFadingA
            additionalFlags: PE::LOOPING);
    }
}
```

For the particle system, I modified it so that it can generate particles from a custom position. When the particle system is inactive, because we cannot delete the mesh from the scene, I moved it 20 units underground.

```
void ParticleSystem::updateParticleSys(PrimitiveTypes::Float32 m_frametime)
{
    if (!m_hParticleSystemCPU.isValid())
    {
        return;
    }
    ParticleSystemCPU* pSysCPU = m_hParticleSystemCPU.getObject<ParticleSystemCPU>();
    SceneNode* root = m_hrootSceneNode.getObject<SceneNode>();
    if (m_isActive)
    {
        pSysCPU->updateParticleBuffer(m_frametime, root->m_worldTransform);

    } else
    {
        Matrix4x4 m_tempM;
        m_tempM.moveDown(distance: 20);
        pSysCPU->updateParticleBuffer(m_frametime, m_tempM);

    }

}
```

```
void ParticleSystemCPU::updateParticleBuffer(PrimitiveTypes::Float32 time, Matrix4x4 m_base)
{
    ParticleBufferCPU<ParticleCPU> *ppbcpu;
    if (!m_hParticleBufferCPU.isValid())
    {
        m_hParticleBufferCPU = Handle( dbgName: "PARTICLE_BUFFER_CPU", neededSize: sizeof(ParticleBufferCPU<ParticleCPU>));
        ppbcpu = new(m_hParticleBufferCPU) ParticleBufferCPU<ParticleCPU>( [&] *m_pContext, m_arena);
        const PrimitiveTypes::Int32 maxParticleSize = m_particleTemplate.m_duration * m_particleTemplate.m_rate;
        ppbcpu->m_values.reset(maxParticleSize);
    } else
    {
        ppbcpu = m_hParticleBufferCPU.getObject<ParticleBufferCPU<ParticleCPU>>();
    }
    m_pastTime += time;
    int totalParticles = m_pastTime * m_particleTemplate.m_rate;

    for (int j = 0; j < ppbcpu->m_values.m_size; j++)
    {
        ppbcpu->m_values[j].m_age -= time;
        if (ppbcpu->m_values[j].m_age <= 0)
        {
            ParticleCPU newParticle = { m_base,
                m_particleTemplate.m_size,
                generateVelocity( m_base), m_age: m_particleTemplate.m_duration};
            ppbcpu->m_values[j] = newParticle;
        } else
        {
            Vector3 curPos = ppbcpu->m_values[j].m_base.getPos();
            curPos += time * ppbcpu->m_values[j].velocity;
            ppbcpu->m_values[j].m_base.setPos( curPos);
        }
    }
```

For UI events, I first moved the parse variable under the game context because it is more intuitive that we can access the pause state from everywhere to customize events during gameplay.

```
    }
    else if (Event_PAUSE::GetClassId() == pGeneralEvt->getClassId())
    {
        m_pContext->m_paused = true;
    }
    else if (Event_RESUME::GetClassId() == pGeneralEvt->getClassId())
    {
        m_pContext->m_paused = false;
    }
```

I then added a restart game event to meet the resettable requirement. Listen to the restart game event in ClientGameObjectManagerAddon.

```
void ClientGameObjectManagerAddon::addDefaultComponents()
{
    GameObjectManagerAddon::addDefaultComponents();

    PE_REGISTER_EVENT_HANDLER(Event_CreateSoldierNPC, ClientGameObjectManagerAddon::do_CreateSoldierNPC);
    PE_REGISTER_EVENT_HANDLER(Event_CreateEnemyNPC, ClientGameObjectManagerAddon::do_CreateEnemyNPC);

    PE_REGISTER_EVENT_HANDLER(Event_CREATE_WAYPOINT, ClientGameObjectManagerAddon::do_CREATE_WAYPOINT);

    // note this component (game obj addon) is added to game object manager after network manager, so network manager will process this event first
    PE_REGISTER_EVENT_HANDLER(PE::Events::Event_SERVER_CLIENT_CONNECTION_ACK, ClientGameObjectManagerAddon::do_SERVER_CLIENT_CONNECTION_ACK);

    PE_REGISTER_EVENT_HANDLER(Event_MoveTank_S_to_C, ClientGameObjectManagerAddon::do_MoveTank);

    PE_REGISTER_EVENT_HANDLER(Event_RESTART_GAME, ClientGameObjectManagerAddon::do_RestartGame);

}
```

What restart does is simply resetting the main character's position to the place where it is spawn.

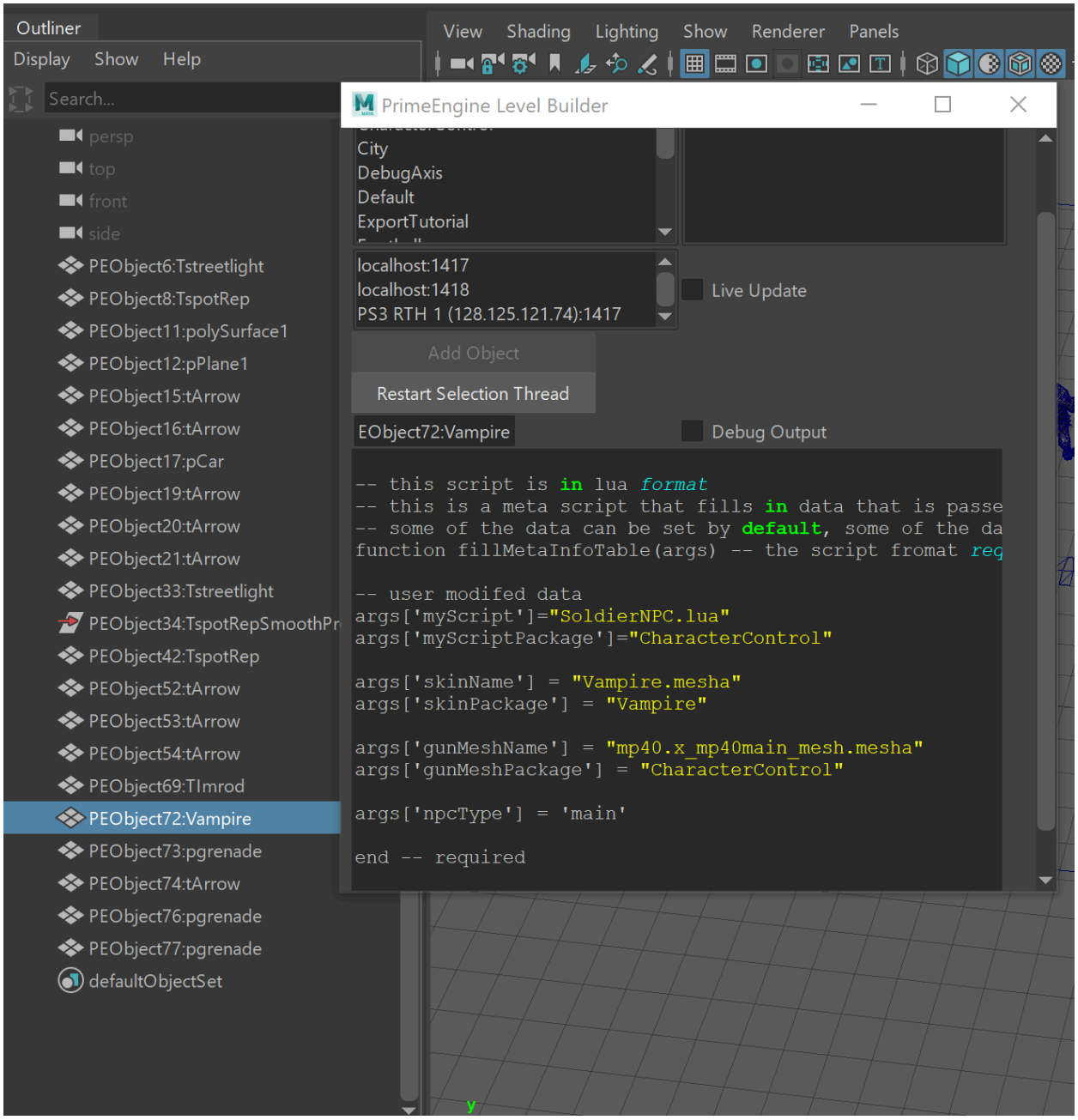
```
void ClientGameObjectManagerAddon::do_RestartGame(PE::Events::Event* pEvt)
{
    OutputDebugStringA("restarted");
    PE::Handle *pHC = m_components.getFirstPtr();
    getGameContext()->m_paused = false;

    for (PrimitiveTypes::UInt32 i = 0; i < m_components.m_size; i++, pHC++) // fast array traversal (increasing ptr)
    {
        Component *pC = (*pHC).getObject<Component>();

        if (pC->isInstanceOf<SoldierNPC>())
        {
            SoldierNPC *pWP = (SoldierNPC *)(pC);
            if (StringOps::strcmp(a: pWP->m_npcType, b: "main") == 0)
            {
                OutputDebugStringA("reset soldier");

                // equal strings, found our waypoint
                SceneNode* soldierSN = pWP->getFirstComponent<SceneNode>();
                soldierSN->m_base.setPos( p: Vector3( x: 0,  y: 0,  z: 0));
                soldierSN->m_base.turnTo( targetPos: Vector3( x: 0,  y: 0,  z: -1));
            }
        }
    }
}
```

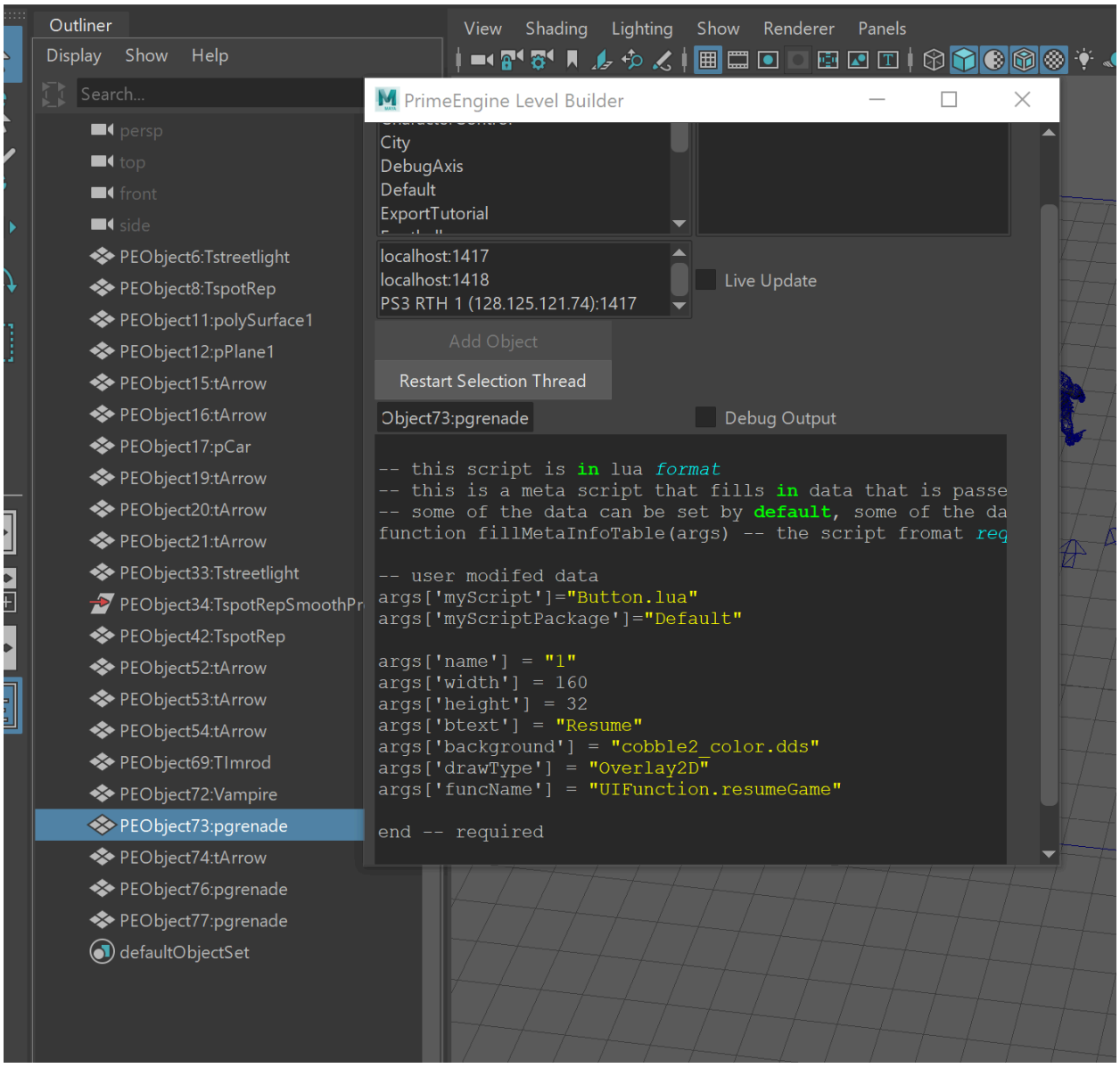
I added the UI function to my UIFunction struct so that it is managed with all other UI events.

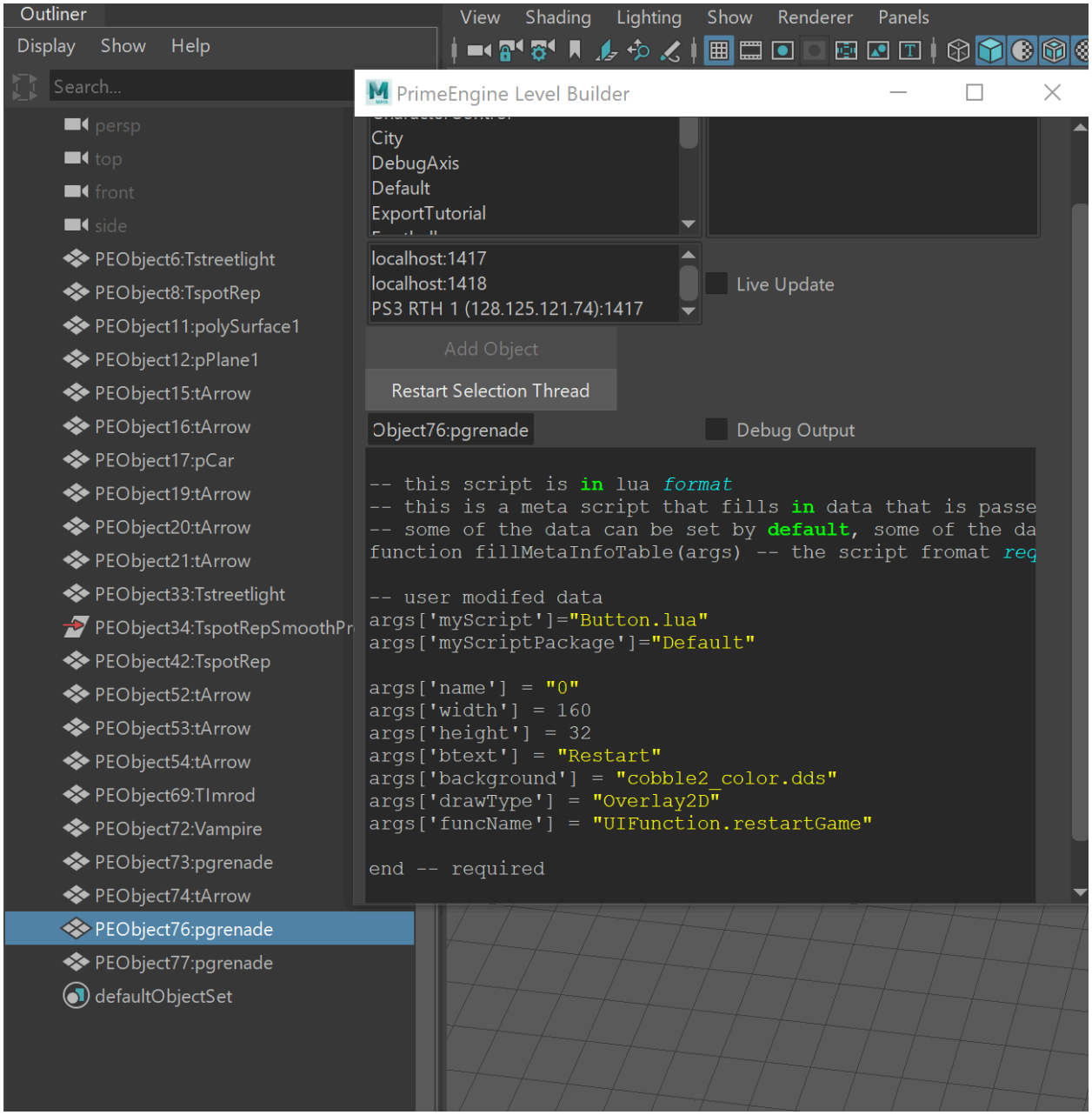
```
}
std::function<void()> UIFunction::findFunction(const char* funcName)
{
    if (std::strcmp(funcName, "UIFunction.pauseGame") == 0) {
        return [this]()->void { this->pauseGame(); };
    } else if (std::strcmp(funcName, "UIFunction.resumeGame") == 0) {
        return [this]()->void { this->resumeGame(); };
    } else if (std::strcmp(funcName, "UIFunction.jumpMenu(1)") == 0) {
        return [this]()->void { this->jumpMenu("1"); };
    } else if (std::strcmp(funcName, "UIFunction.jumpMenu(0)") == 0) {
        return [this]()->void { this->jumpMenu("0"); };
    } else if (std::strcmp(funcName, "UIFunction.jumpMenu(2)") == 0)
    {
        return [this]()->void { this->jumpMenu("2"); };
    } else if (std::strcmp(funcName, "UIFunction.exitGame") == 0) {
        return [this]()->void { this->exitGame(); };
    }
    else if (std::strcmp(funcName, "UIFunction.restartGame") == 0) {
        return [this]()->void { this->restartGame(); };
    }
    // If function not found, return an empty std::function or a default function
    return [this]()->void { this->defaultFunc(); };
}

void UIFunction::pauseGame()
```

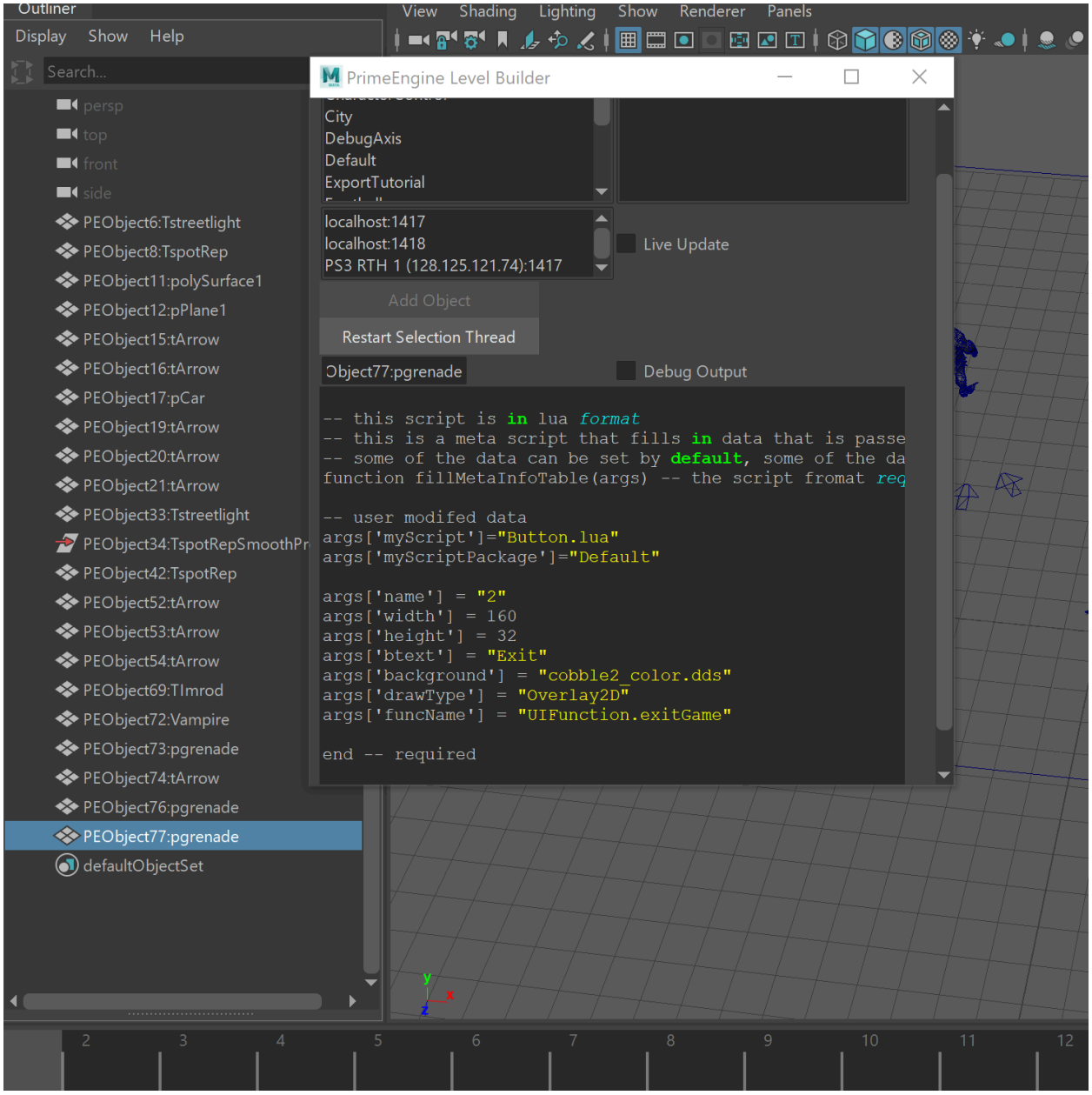
```
void UIFunction::exitGame()
{
    PE::Handle h( dbgName: "Event_EXIT_GAME", neededSize: sizeof(Events::Event_EXIT_GAME));
    Events::Event_EXIT_GAME *pEvt = new (h) Events::Event_EXIT_GAME;
    Events::EventQueueManager::Instance()->add( h, queueType: Events::QT_GENERAL);
}

void UIFunction::restartGame()
{
    PE::Handle h( dbgName: "Event_RESTART", neededSize: sizeof(Events::Event_RESTART_GAME));
    Events::Event_RESTART_GAME *pEvt = new (h) Events::Event_RESTART_GAME;
    Events::EventQueueManager::Instance()->add( h, queueType: Events::QT_GENERAL);
    UIManager::Instance()->setAllUIActive(false);
}

void UIFunction::defaultFunc()
{
    OutputDebugStringA("UIFunction::findFunction: function not implemented");
}

}namespace Components {
```

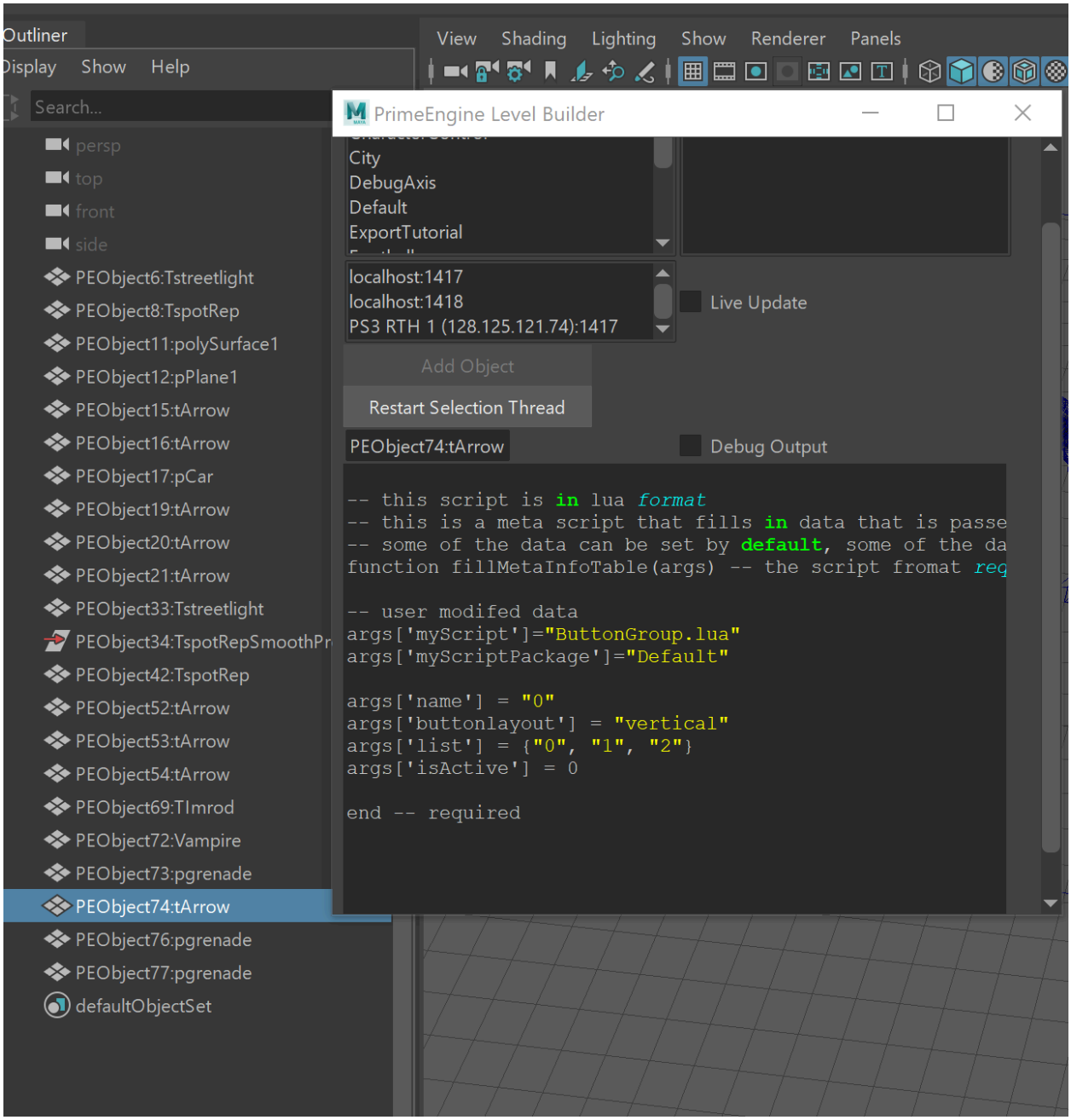
Last, I modified maya level editor and added all my customized parameter into it to setup my scene.


Outliner
Display Show Help
Search...
persp
top
front
side
PEObject6:Tstreetlight
PEObject8:TspotRep
PEObject11:polySurface1
PEObject12:pPlane1
PEObject15:tArrow
PEObject16:tArrow
PEObject17:pCar
PEObject19:tArrow
PEObject20:tArrow
PEObject21:tArrow
PEObject33:Tstreetlight
PEObject34:TspotRepSmoothPr
PEObject42:TspotRep
PEObject52:tArrow
PEObject53:tArrow
PEObject54:tArrow
PEObject69:TImrod
PEObject72:Vampire
PEObject73:pgrenade
PEObject74:tArrow
PEObject76:pgrenade
PEObject77:pgrenade
defaultObjectSet
View Shading Lighting Show Renderer Panels
PrimeEngine Level Builder
City
DebugAxis
Default
ExportTutorial
localhost:1417
localhost:1418
PS3 RTH 1 (128.125.121.74):1417
Live Update
Add Object
Restart Selection Thread
EObject72:Vampire
Debug Output
-- this script is in lua format
-- this is a meta script that fills in data that is passe
-- some of the data can be set by default, some of the da
function fillMetaInfoTable(args) -- the script fromat req
-- user modifed data
args['myScript']="SoldierNPC.lua"
args['myScriptPackage']="CharacterControl"
args['skinName'] = "Vampire.mesha"
args['skinPackage'] = "Vampire"
args['gunMeshName'] = "mp40.x_mp40main_mesh.mesha"
args['gunMeshPackage'] = "CharacterControl"
args['npcType'] = 'main'
end -- required


Outliner
Display Show Help
Search...
persp
top
front
side
PEObject6:Tstreetlight
PEObject8:TspotRep
PEObject11:polySurface1
PEObject12:pPlane1
PEObject15:tArrow
PEObject16:tArrow
PEObject17:pCar
PEObject19:tArrow
PEObject20:tArrow
PEObject21:tArrow
PEObject33:Tstreetlight
PEObject34:TspotRepSmoothPr
PEObject42:TspotRep
PEObject52:tArrow
PEObject53:tArrow
PEObject54:tArrow
PEObject69:TImrod
PEObject72:Vampire
PEObject73:pgrenade
PEObject74:tArrow
PEObject76:pgrenade
PEObject77:pgrenade
defaultObjectSet
View Shading Lighting Show Renderer Panels
PrimeEngine Level Builder
City
DebugAxis
Default
ExportTutorial
localhost:1417
localhost:1418
PS3 RTH 1 (128.125.121.74):1417
Live Update
Add Object
Restart Selection Thread
Object73:pgrenade
Debug Output
-- this script is in lua format
-- this is a meta script that fills in data that is passe
-- some of the data can be set by default, some of the da
function fillMetaInfoTable(args) -- the script fromat req
-- user modifed data
args['myScript']="Button.lua"
args['myScriptPackage']="Default"
args['name'] = "1"
args['width'] = 160
args['height'] = 32
args['btext'] = "Resume"
args['background'] = "cobble2_color.dds"
args['drawType'] = "Overlay2D"
args['funcName'] = "UIFunction.resumeGame"
end -- required


Outliner
Display Show Help
Search...
persp
top
front
side
PEObject6:Tstreetlight
PEObject8:TspotRep
PEObject11:polySurface1
PEObject12:pPlane1
PEObject15:tArrow
PEObject16:tArrow
PEObject17:pCar
PEObject19:tArrow
PEObject20:tArrow
PEObject21:tArrow
PEObject33:Tstreetlight
PEObject34:TspotRepSmoothPr
PEObject42:TspotRep
PEObject52:tArrow
PEObject53:tArrow
PEObject54:tArrow
PEObject69:TImrod
PEObject72:Vampire
PEObject73:pgrenade
PEObject74:tArrow
PEObject76:pgrenade
PEObject77:pgrenade
defaultObjectSet
View Shading Lighting Show Renderer Panels
PrimeEngine Level Builder
City
DebugAxis
Default
ExportTutorial
localhost:1417
localhost:1418
PS3 RTH 1 (128.125.121.74):1417
Live Update
Add Object
Restart Selection Thread
Object76:pgrenade
Debug Output
-- this script is in lua format
-- this is a meta script that fills in data that is passe
-- some of the data can be set by default, some of the da
function fillMetaInfoTable(args) -- the script fromat req
-- user modifed data
args['myScript']="Button.lua"
args['myScriptPackage']="Default"
args['name'] = "0"
args['width'] = 160
args['height'] = 32
args['btext'] = "Restart"
args['background'] = "cobble2_color.dds"
args['drawType'] = "Overlay2D"
args['funcName'] = "UIFunction.restartGame"
end -- required


Outliner
Display Show Help
Search...
persp
top
front
side
PEObject6:Tstreetlight
PEObject8:TspotRep
PEObject11:polySurface1
PEObject12:pPlane1
PEObject15:tArrow
PEObject16:tArrow
PEObject17:pCar
PEObject19:tArrow
PEObject20:tArrow
PEObject21:tArrow
PEObject33:Tstreetlight
PEObject34:TspotRepSmoothPr
PEObject42:TspotRep
PEObject52:tArrow
PEObject53:tArrow
PEObject54:tArrow
PEObject69:TImrod
PEObject72:Vampire
PEObject73:pgrenade
PEObject74:tArrow
PEObject76:pgrenade
PEObject77:pgrenade
defaultObjectSet
View Shading Lighting Show Renderer Panels
PrimeEngine Level Builder
City
DebugAxis
Default
ExportTutorial
localhost:1417
localhost:1418
PS3 RTH 1 (128.125.121.74):1417
Live Update
Add Object
Restart Selection Thread
Object77:pgrenade
Debug Output
-- this script is in lua format
-- this is a meta script that fills in data that is passe
-- some of the data can be set by default, some of the da
function fillMetaInfoTable(args) -- the script fromat req
-- user modifed data
args['myScript']="Button.lua"
args['myScriptPackage']="Default"
args['name'] = "2"
args['width'] = 160
args['height'] = 32
args['btext'] = "Exit"
args['background'] = "cobble2_color.dds"
args['drawType'] = "Overlay2D"
args['funcName'] = "UIFunction.exitGame"
end -- required
2 3 4 5 6 7 8 9 10 11 12


Outliner
Display Show Help
Search...
persp
top
front
side
PEObject6:Tstreetlight
PEObject8:TspotRep
PEObject11:polySurface1
PEObject12:pPlane1
PEObject15:tArrow
PEObject16:tArrow
PEObject17:pCar
PEObject19:tArrow
PEObject20:tArrow
PEObject21:tArrow
PEObject33:Tstreetlight
PEObject34:TspotRepSmoothPr
PEObject42:TspotRep
PEObject52:tArrow
PEObject53:tArrow
PEObject54:tArrow
PEObject69:TImrod
PEObject72:Vampire
PEObject73:pgrenade
PEObject74:tArrow
PEObject76:pgrenade
PEObject77:pgrenade
defaultObjectSet
View Shading Lighting Show Renderer Panels
PrimeEngine Level Builder
City
DebugAxis
Default
ExportTutorial
localhost:1417
localhost:1418
PS3 RTH 1 (128.125.121.74):1417
Live Update
Add Object
Restart Selection Thread
PEObject74:tArrow
Debug Output
-- this script is in lua format
-- this is a meta script that fills in data that is passe
-- some of the data can be set by default, some of the da
function fillMetaInfoTable(args) -- the script fromat req
-- user modifed data
args['myScript']="ButtonGroup.lua"
args['myScriptPackage']="Default"
args['name'] = "0"
args['buttonlayout'] = "vertical"
args['list'] = {"0", "1", "2"}
args['isActive'] = 0
end -- required