Video Link:
https://drive.google.com/file/d/1eWTmqgEpQQviL_hNa8Tqb_c2Ta4VXH3W/view?usp=sharing

For milestone1, I implemented a button, a buttongroup, and several data driven menus to control the gameplay.
I initially added a new Sprite mesh class that extends Mesh. Similar to Textmesh, I drew a single quad with width and height passed

```
namespace PE {
    namespace Components {

        struct SpriteMesh : public Mesh
        {
            PE_DECLARE_CLASS(SpriteMesh);

            // Constructor -------------------------------------------------------------
            SpriteMesh(PE::GameContext &context, PE::MemoryArena arena, Handle hMyself) : Mesh([&]context, arena, hMyself)
            {
                m_loaded = false;
            }

            virtual ~SpriteMesh(){}

            virtual void addDefaultComponents();

            PE_DECLARE_IMPLEMENT_EVENT_HANDLER_WRAPPER(do_GATHER_DRAWCALLS);
            virtual void do_GATHER_DRAWCALLS(Events::Event *pEvt);

            void loadFromSource_needsRC(const char *fileName, const char *techName, int &threadOwnershipMask, float width, float height);

            PrimitiveTypes::Float32 m_width, m_height;
            PrimitiveTypes::Bool m_loaded;
            Handle m_meshCPU;
        };

    }namespace Components ; // namespace Components
```

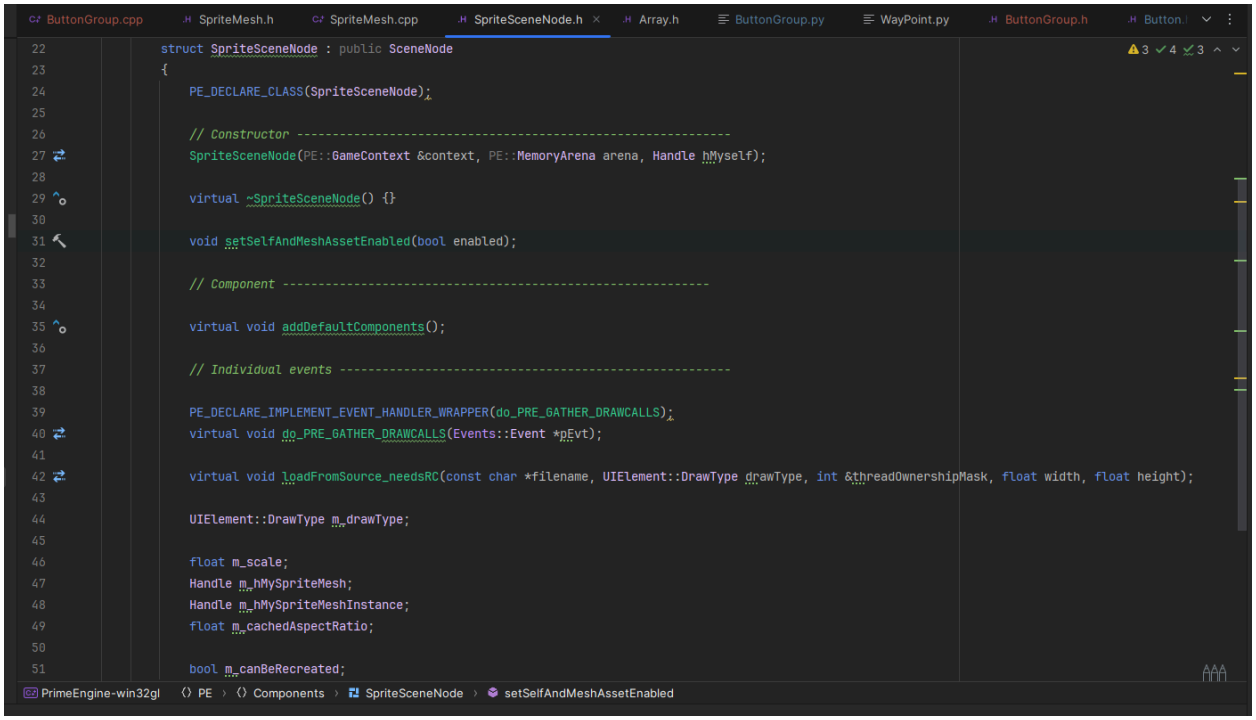
ButtonGroup.cpp | SpriteMesh.h | SpriteMesh.cpp | Array.h | ButtonGroup.py | Button.py | WayPoint.py | ButtonGroup.h

```
52
53      PositionBufferCPU *pVB = mcpu.m_hPositionBufferCPU.getObject<PositionBufferCPU>();
54      TexCoordBufferCPU *pTCB = mcpu.m_hTexCoordBufferCPU.getObject<TexCoordBufferCPU>();
55      // ColorBufferCPU *pCB = mcpu.m_hColorBufferCPU.getObject<ColorBufferCPU>();
56
57      NormalBufferCPU *pNB = mcpu.m_hNormalBufferCPU.getObject<NormalBufferCPU>();
58      IndexBufferCPU *pIB = mcpu.m_hIndexBufferCPU.getObject<IndexBufferCPU>();
59      pVB->m_values.reset(capacity: 4 * 3); // 4 verts * (x,y,z)
60      pTCB->m_values.reset(capacity: 4 * 2);
61      pNB->m_values.reset( capacity: 4 * 3);
62      pIB->m_values.reset(capacity: 6); // 2 tris
63      // pCB->m_values.add(1.0f, 1.0f, 0);
64
65      pIB->m_indexRanges[0].m_start = 0;
66      pIB->m_indexRanges[0].m_end = 5;
67      pIB->m_indexRanges[0].m_minVertIndex = 0;
68      pIB->m_indexRanges[0].m_maxVertIndex = 3;
69
70      pIB->m_minVertexIndex = pIB->m_indexRanges[0].m_minVertIndex;
71      pIB->m_maxVertexIndex = pIB->m_indexRanges[0].m_maxVertIndex;
72
73      m_width = width;
74      m_height = height;
75
76      // float pixSizex = 1.0f / 1280.0f;
77      // float pixSizey = 1.0f / 720.f;
78
79      pVB->m_values.add(0, 0, 0); // top lef
80      pVB->m_values.add(width / 16, 0, 0); // top right
81      pVB->m_values.add(width / 16, -height / 16, 0);
```

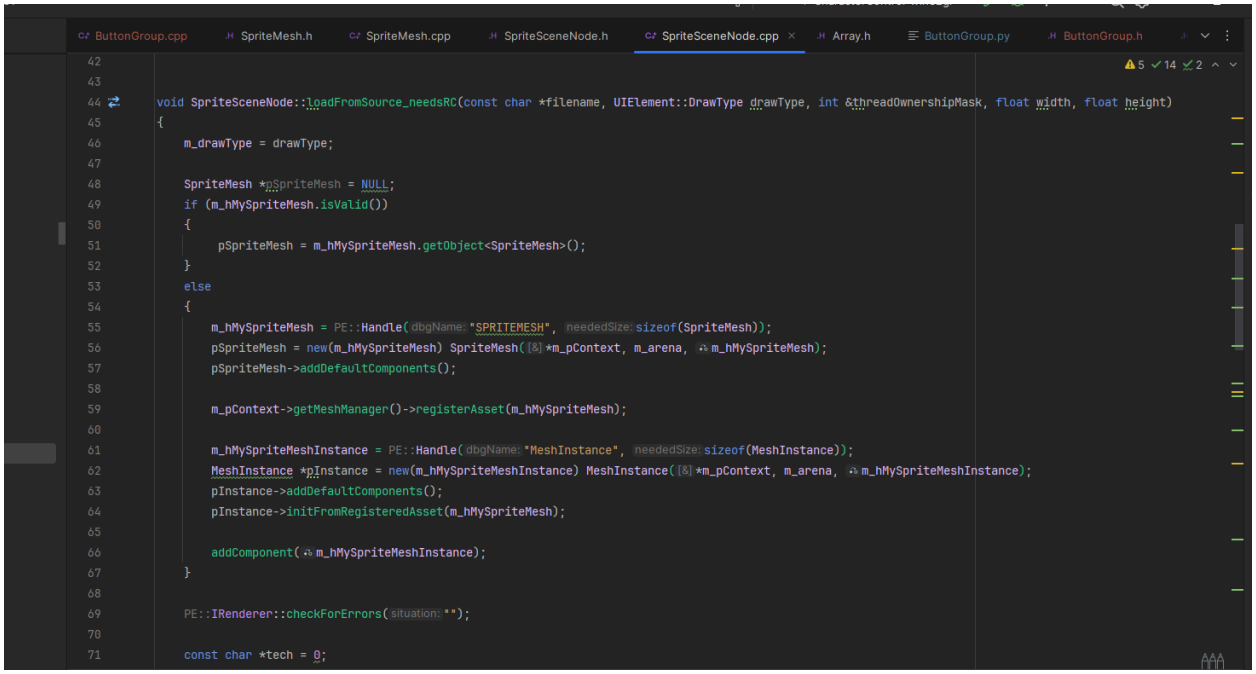
I drew a single quad that has 4 vertices and 2 triangles and scaled the vertex indices.

```
77      // float pixSizey = 1.0f / 720.f;
78
79      pVB->m_values.add(0, 0, 0); // top lef
80      pVB->m_values.add(width / 16, 0, 0); // top right
81      pVB->m_values.add(width / 16, -height / 16, 0);
82      pVB->m_values.add(0, -height / 16, 0);
83
84      pIB->m_values.add(0, 1, 2);
85      pIB->m_values.add(2, 3, 0);
86
87      pTCB->m_values.add(0 ,0); // top left
88      pTCB->m_values.add(1, 0); // top right
89      pTCB->m_values.add(1, 1);
90      pTCB->m_values.add(0, 1);
91
92      pNB->m_values.add(0, 0, 0);
93      pNB->m_values.add(0, 0, 0);
94      pNB->m_values.add(0, 0, 0);
95      pNB->m_values.add(0, 0, 0);
96
97      if (!m_loaded)
98      {
99          // first time creating gpu mesh
100         loadFromMeshCPU_needsRC([&]mcpu, [&]threadOwnershipMask);
101
```

Then I created a SpriteSceneNode similar to TextSceneNode. The background texture file, width and height gets passed in.




```cpp
SpriteSceneNode::SpriteSceneNode(PE::GameContext &context, PE::MemoryArena arena, Handle hMyself)
: SceneNode([&]context, arena, hMyself)
{
    m_cachedAspectRatio = 1.0f;
    m_scale = 1.0f;
    if (IRenderer* pS = context.getGPUScreen())
        m_cachedAspectRatio = float(pS->getWidth()) / float(pS->getHeight());
}

void SpriteSceneNode::addDefaultComponents()
{
    SceneNode::addDefaultComponents();
    // event handlers
    PE_REGISTER_EVENT_HANDLER(Events::Event_PRE_GATHER_DRAWCALLS, SpriteSceneNode::do_PRE_GATHER_DRAWCALLS);
}

void SpriteSceneNode::setSelfAndMeshAssetEnabled(bool enabled)
{
    setEnabled(enabled);

    if (m_hMySpriteMesh.isValid())
    {
        m_hMySpriteMesh.getObject<Component>()->setEnabled(enabled);
    }
}
```



```
ButtonGroup.cpp | SpriteMesh.h | SpriteMesh.cpp | SpriteSceneNode.h | SpriteSceneNode.cpp | Array.h | ButtonGroup.py | ButtonGroup.h
```

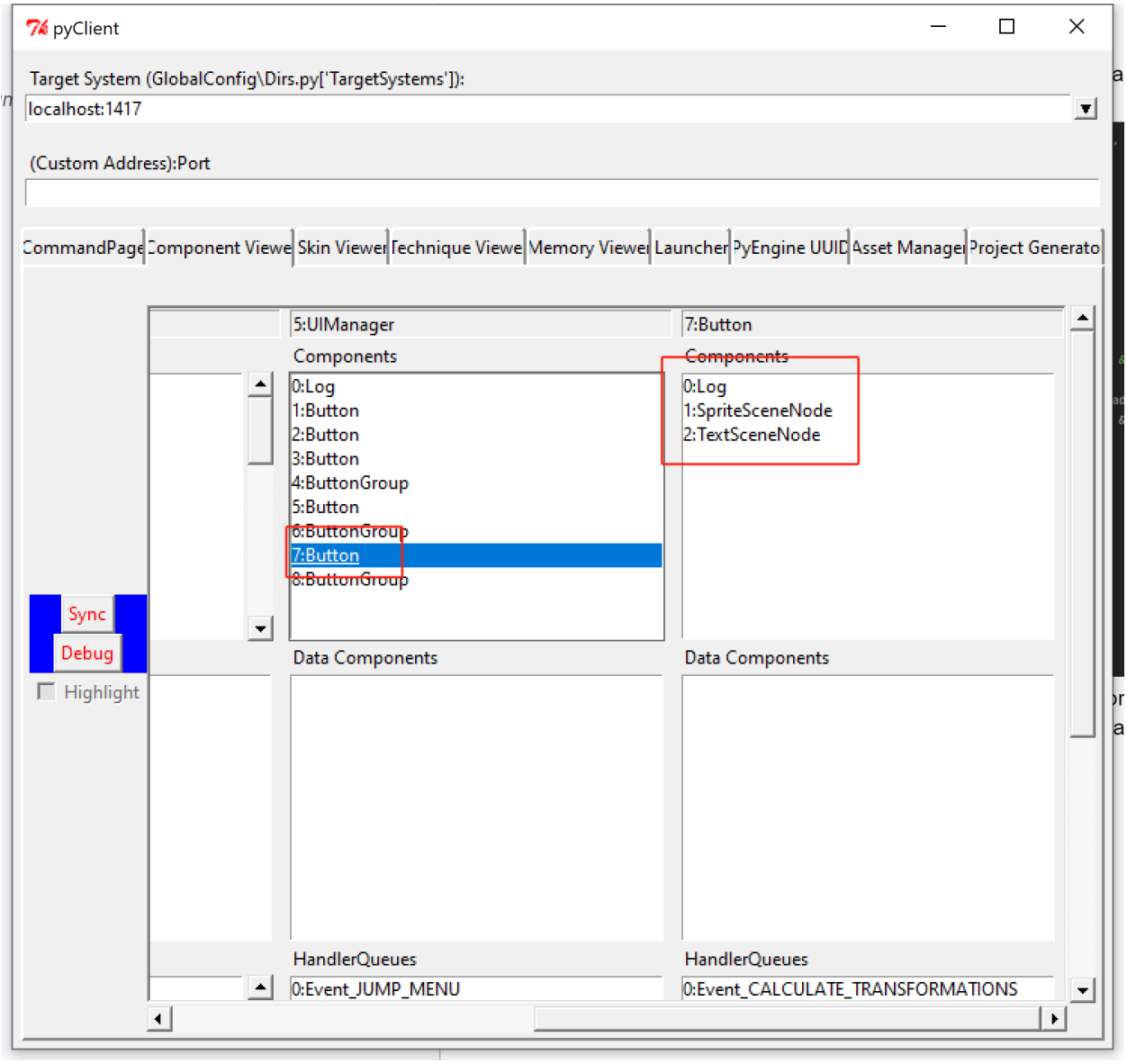
```
void SpriteSceneNode::loadFromSource_needsRC(const char *filename, UIElement::DrawType drawType, int &threadOwnershipMask, float width, float height)
{
    m_drawType = drawType;

    SpriteMesh *pSpriteMesh = NULL;
    if (m_hMySpriteMesh.isValid())
    {
        pSpriteMesh = m_hMySpriteMesh.getObject<SpriteMesh>();
    }
    else
    {
        m_hMySpriteMesh = PE::Handle(dbgName: "SPRITEMESH", neededSize: sizeof(SpriteMesh));
        pSpriteMesh = new(m_hMySpriteMesh) SpriteMesh([&] *m_pContext, m_arena, m_hMySpriteMesh);
        pSpriteMesh->addDefaultComponents();

        m_pContext->getMeshManager()->registerAsset(m_hMySpriteMesh);

        m_hMySpriteMeshInstance = PE::Handle(dbgName: "MeshInstance", neededSize: sizeof(MeshInstance));
        MeshInstance *pInstance = new(m_hMySpriteMeshInstance) MeshInstance([&] *m_pContext, m_arena, m_hMySpriteMeshInstance);
        pInstance->addDefaultComponents();
        pInstance->initFromRegisteredAsset(m_hMySpriteMesh);

        addComponent(m_hMySpriteMeshInstance);
    }

    PE::IRenderer::checkForErrors(situation: "");

    const char *tech = 0;
```

```
    PE::IRenderer::checkForErrors(situation: "");

    const char *tech = 0;
    if (drawType == UIElement::Overlay2D_3DPos || drawType == UIElement::Overlay2D)
        tech = "StdMesh_2D_Diffuse_A_RGBIntensity_Tech";
    if (drawType == UIElement::InWorldFacingCamera)
        tech = "StdMesh_Diffuse_Tech";

    pSpriteMesh->loadFromSource_needsRC(filename, tech, [&] threadOwnershipMask, width, height);
}

void SpriteSceneNode::do_PRE_GATHER_DRAWCALLS(Events::Event *pEvt)
{
    Events::Event_PRE_GATHER_DRAWCALLS *pDrawEvent = NULL;
    pDrawEvent = (Events::Event_PRE_GATHER_DRAWCALLS *)(pEvt);

    Matrix4x4 projectionViewWorldMatrix = pDrawEvent->m_projectionViewTransform;
    Matrix4x4 worldMatrix;

    if (!m_hMySpriteMeshInstance.isValid())
        return;

    SpriteMesh *pSpriteMesh = m_hMySpriteMesh.getObject<SpriteMesh>();

    if (m_drawType == UIElement::InWorldFacingCamera)
    {
        m_worldTransform.turnTo(pDrawEvent->m_eyePos);
```

Then I created a Button file. The button takes a string and a filepath, a name, drawtype, width, height, and an onclick function.

```
Button(PE::GameContext& context, PE::MemoryArena arena, Handle hMyself, Events::Event_CREATE_BUTTON *pEvt);

virtual ~Button(){}

virtual void addDefaultComponents();

PE_DECLARE_IMPLEMENT_EVENT_HANDLER_WRAPPER(do_PRE_GATHER_DRAWCALLS);
virtual void do_PRE_GATHER_DRAWCALLS(Events::Event *pEvt);

virtual void setSelfAndMeshAssetEnabled(bool isActive);

// void loadFromString_needsRC(const char *str, DrawType drawType, int &threadOwnershipMask);

void createTextSceneNode(const char *str, DrawType drawType, int &threadOwnershipMask);
void createSpriteSceneNode(const char * bgFile, DrawType drawType, int &threadOwnershipMask, float width, float height);

PE_DECLARE_IMPLEMENT_EVENT_HANDLER_WRAPPER(do_TEST_ONCLICK);
void do_TEST_ONCLICK(Events::Event *pEvt);

UIFunction* m_funcMap;

char* m_buttonText;
char* m_buttonImageSource;
char* m_name;

Handle m_hMySpriteSN;
Handle m_hMyTextSN;
```

In Button, it loads the background file and creates a spriteSceneNode. It then loads the text and creates a TextSceneNode. Both of these components are added under Button.


pyClient
Target System (GlobalConfig\Dirs.py['TargetSystems']):
localhost:1417
(Custom Address):Port
CommandPage
Component Viewe
Skin Viewer
Technique Viewe
Memory Viewer
Launcher
PyEngine UUID
Asset Manager
Project Generato
5:UIManager
Components
0:Log
1:Button
2:Button
3:Button
4:ButtonGroup
5:Button
6:ButtonGroup
7:Button
8:ButtonGroup
7:Button
Components
0:Log
1:SpriteSceneNode
2:TextSceneNode
Sync
Debug
Highlight
Data Components
Data Components
HandlerQueues
0:Event_JUMP_MENU
HandlerQueues
0:Event_CALCULATE_TRANSFORMATIONS

Then I added a ButtonGroup to manage different groups of buttons. The ButtonGroup takes the initial position, a layout style, a name, and a list of buttons inside, labeled by button names.

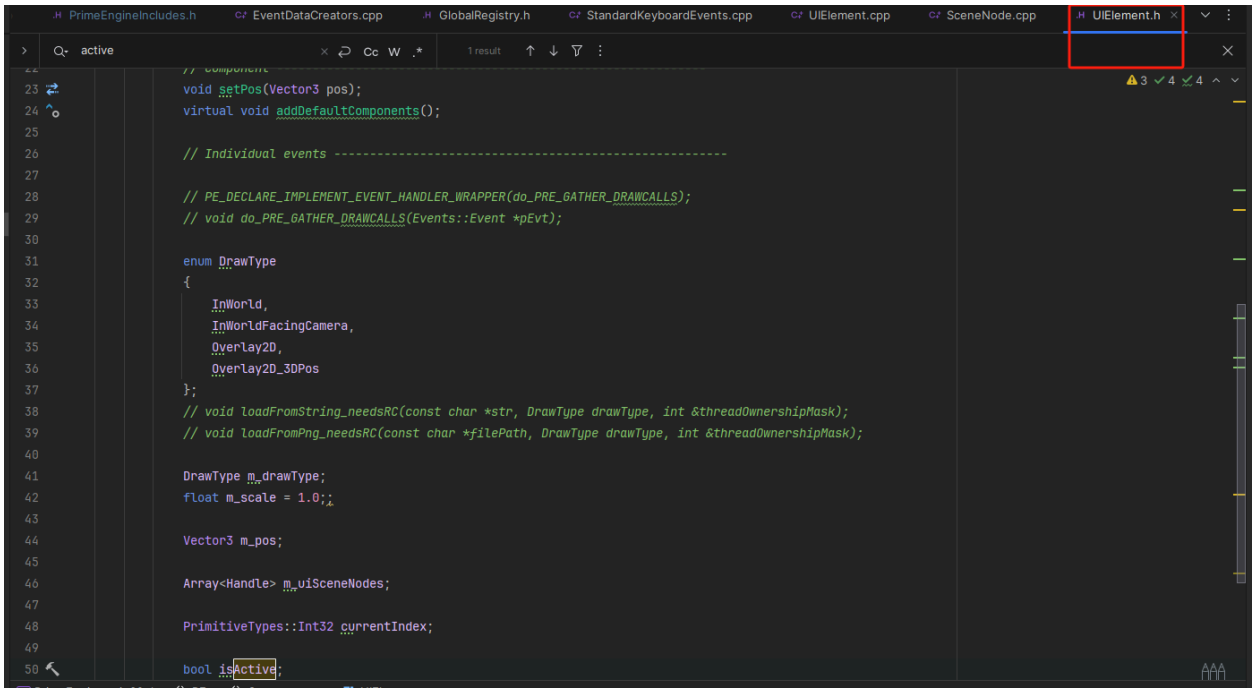
```
namespace Components {

    struct ButtonGroup : UIElement
    {
        enum Layout
        {
            Vertical,
            Horizontal
        };
        PE_DECLARE_CLASS(ButtonGroup);

        ButtonGroup( PE::GameContext &context, PE::MemoryArena arena, PE::Handle hMyself, const Events::Event_CREATE_BUTTONGROUP *pEvt);
        virtual void addDefaultComponents();

        PE_DECLARE_IMPLEMENT_EVENT_HANDLER_WRAPPER(do_PRE_GATHER_DRAWCALLS);
        virtual void do_PRE_GATHER_DRAWCALLS(Events::Event *pEvt);

        const char* m_name;
        Layout m_layout;
        Array<Handle> m_ButtonList;
        char* m_buttonNames[8];
        bool m_isButtonSet = false;
    };
} // namespace Components
} // namespace PE
```

```
    : UIElement([&]context, arena, hMyself)
    {
        m_ButtonList.reset(capacity: 8);
        m_name = pEvt->m_name;
        isActive = pEvt->m_active;
        if (strcmp(pEvt->m_layout, "vertical") == 0)
        {
            m_layout = Vertical;
        } else if (strcmp(pEvt->m_layout, "horizontal") == 0)
        {
            m_layout = Horizontal;
        } else
        {
            m_layout = Vertical;
        }

        for (int i = 0; i < 8; i++)
        {
            if (strlen(pEvt->m_bNames[i]) > 0)
            {
                m_buttonNames[i] = (char*)pEvt->m_bNames[i];
            }
            else
            {
                m_buttonNames[i] = "";
            }
        }
    }
```

In ButtonGroup, I calculate the position of its child buttons based on the group's position.

```
{
    Vector3 curPos = Vector3(m_pos);

    if (m_isButtonSet)
    {
        for (int i = 0; i < m_ButtonList.m_size; i++)
        {
            Handle h = m_ButtonList[i];
            if (h.isValid())
            {
                Button* curButton = h.getObject<Button>();
                curButton->setSelfAndMeshAssetEnabled(isActive);
                if (m_layout == Vertical)
                {
                    curButton->setPos( curPos);
                    curPos.m_y -=  (float)(curButton->m_height + 32) / 720.f;
                }
                if (m_layout == Horizontal)
                {
                    curButton->setPos( curPos);
                    curPos.m_x += (float)(curButton->m_width + 32) / 1280.f;
                }
            }
        }
    } else
```

I created a UIElement class so that all UI elements will extend this class.



Each UIElement will have a list of SceneNodes. Currently, it can either be a SpriteSceneNode or a TextSceneNode.

```
void UIElement::setSelfAndMeshAssetEnabled(bool enabled)
{
    setEnabled(enabled);

    for (int i = 0 ; i < m_uiSceneNodes.m_size; i++)
    {
        Handle curSceneNodeHandle = m_uiSceneNodes[i];
        if (curSceneNodeHandle.isValid())
        {
            Component* curComponent = curSceneNodeHandle.getObject<Component>();
            curComponent->setEnabled(enabled);
        }
    }
}


} namespace Components ; // namespace Components
} namespace PE ; // namespace PE
```

Then I created a UIManager. It has a list that contains all UI elements. It also handles all creation, enable and disable of all UIElements, and some events regarding UI transitions.

```
// Constructor -----------------------------------------------------------
UIManager(PE::GameContext &context, PE::MemoryArena arena, Handle hMyself);
virtual ~UIManager(){}
// Methods      ----------------------------------------------------------

void setUIActive(Array<PrimitiveTypes::Int32> UIIndices, bool isActive);

PE_DECLARE_IMPLEMENT_EVENT_HANDLER_WRAPPER(method_: do_PRE_GATHER_DRAWCALLS);
virtual void do_PRE_GATHER_DRAWCALLS(Events::Event *pEvt);

PE_DECLARE_IMPLEMENT_EVENT_HANDLER_WRAPPER(method_: do_JUMP_MENU);
virtual void do_JUMP_MENU(Events::Event *pEvt);

void postPreDraw(int &threadOwnershipMask);
// Component -------------------------------------------------------------

virtual void addDefaultComponents();

void createButton(Events::Event_CREATE_BUTTON* pEvt);
void createButtonGroup(Events::Event_CREATE_BUTTONGROUP* pEvt);

// Individual events -----------------------------------------------------

Array<Handle> m_UIList;

PrimitiveTypes::Int32 m_maxWidth;
```

```
62      virtual void addDefaultComponents();
63
64      void createButton(Events::Event_CREATE_BUTTON* pEvt);
65      void createButtonGroup(Events::Event_CREATE_BUTTONGROUP* pEvt);
66
67      // Individual events -----------------------------------------------------
68
69      Array<Handle> m_UIList;
70
71      PrimitiveTypes::Int32 m_maxWidth;
72      PrimitiveTypes::Int32 m_maxHeight;
73      float m_cachedAspectRatio;
74
75      private:
76          static Handle s_myHandle;
77          static const int NUM_MaxUI = 64;
78          int m_last;
79          GameContext *m_context;
80
81
82  };
83
84  } namespace Components ; // namespace Components
85  } namespace PE ; // namespace PE
86  #endif  #ifndef __PYENGINE_2_0_UIMANAGER_H__
87
```

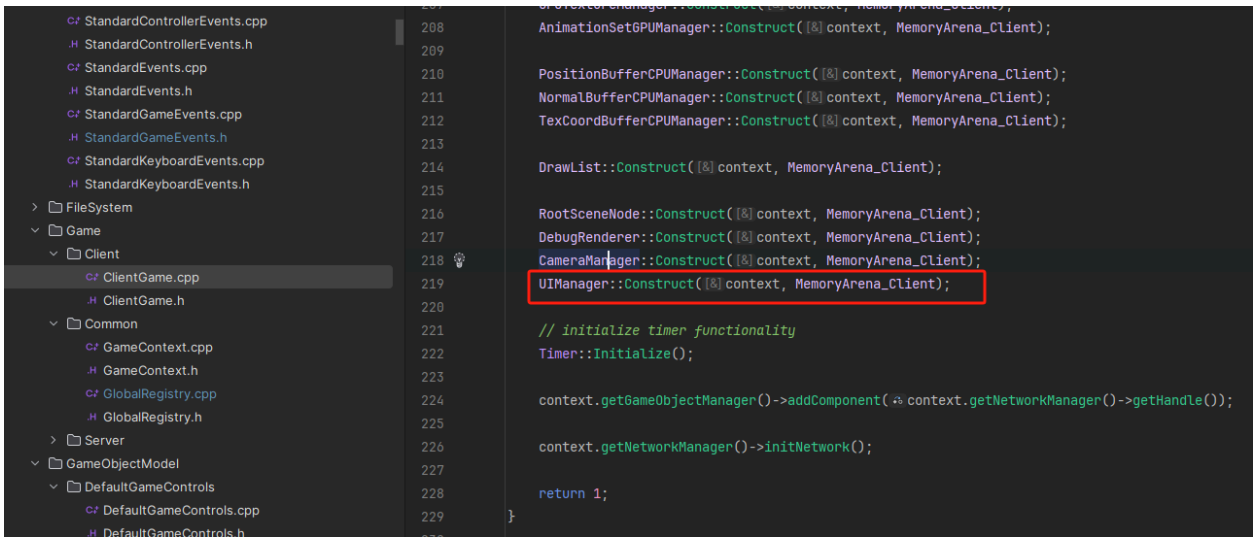
```cpp
void UIManager::createButton(Event_CREATE_BUTTON *pRealEvent)
{
    if (m_UIList.m_size >= NUM_MaxUI)
    {
        return;
    }
    m_pContext->getGPUScreen()->AcquireRenderContextOwnership([&]pRealEvent->m_threadOwnershipMask);
    Handle h = PE::Handle(dbgName:"BUTTON", neededSize:sizeof(Button));
    Button *pButton = new(h) Button([&]*m_pContext, m_arena, h, pRealEvent);
    pButton->addDefaultComponents();
    addComponent(h);
    Vector3 pos;
    if (pButton->m_drawType == UIElement::Overlay2D)
    {
        pos.m_x = -1.0f + 2.0f * (pRealEvent->m_pos.m_x / UIManager::Instance()->m_maxWidth);
        pos.m_y = -1.0f + 2.0f * (1.0f -(pRealEvent->m_pos.m_y / UIManager::Instance()->m_maxHeight));
    } else
    {
        pos = pRealEvent->m_pos;
    }
    pButton->setPos(pos);
    pButton->createSpriteSceneNode(pButton->m_buttonImageSource, pButton->m_drawType, [&]m_context->m_gameThreadThreadOwnershipMask, pButton->m_width, pB
    pButton->createTextSceneNode(pButton->m_buttonText, pButton->m_drawType, [&]m_context->m_gameThreadThreadOwnershipMask);
    pButton->setSelfAndMeshAssetEnabled(pRealEvent->isActive);
    pButton->currentIndex = m_UIList.m_size;
    m_UIList.add(h);
}
```

```cpp
void UIManager::createButtonGroup(Event_CREATE_BUTTONGROUP *pRealEvent)
{
    if (m_UIList.m_size >= NUM_MaxUI)
    {
        return;
    }
    Handle h = PE::Handle(dbgName:"BUTTONGROUP", neededSize:sizeof(ButtonGroup));
    ButtonGroup *pButtonGroup = new(h) ButtonGroup([&]*m_pContext, m_arena, h, pRealEvent);
    pButtonGroup->addDefaultComponents();
    addComponent(h);
    Vector3 pos;
    pos.m_x = -1.0f + 2.0f * (pRealEvent->m_pos.m_x / UIManager::Instance()->m_maxWidth);
    pos.m_y = -1.0f + 2.0f * (1.0f -(pRealEvent->m_pos.m_y / UIManager::Instance()->m_maxHeight));
    pButtonGroup->setPos(pos);
    m_UIList.add(h);
}
```

```cpp
void UIManager::setUIActive(Array<PrimitiveTypes::Int32> UIIndices, bool isActive)
{
    for (int i = 0; i < UIIndices.m_size; i++)
    {
        Handle hCurElement = m_UIList[i];
        UIElement* curUI = hCurElement.getObject<UIElement>();
        curUI->setSelfAndMeshAssetEnabled(isActive);
    }
}
```

It is constructed when Game is started.

```cpp
// Singleton -------------------------------------------------------------

void UIManager::Construct(PE::GameContext &context, PE::MemoryArena arena)
{
    Handle handle(dbgName: "UIManager", neededSize: sizeof(UIManager));
    UIManager *pUIManager = new(handle) UIManager([&] context, arena, handle);
    pUIManager->addDefaultComponents();
    // Singleton
    SetInstanceHandle(handle);
    RootSceneNode::Instance()->addComponent(handle);
}

// Constructor -----------------------------------------------------------
UIManager::UIManager(PE::GameContext &context, PE::MemoryArena arena, Handle hMyself)
: Component([&] context, arena, hMyself)
{
    m_UIList.reset(capacity: NUM_MaxUI);
    m_cachedAspectRatio = 1.0f;
    m_context = &context;
    if (IRenderer* pS = context.getGPUScreen())
    {
        m_maxWidth = pS->getWidth();
        m_maxHeight = pS->getHeight();
        m_cachedAspectRatio = float(pS->getWidth()) / float(pS->getHeight());
    }

}
```



Then I created a Click Event for the application

```cpp
struct Event_LEFT_BUTTON_UP : public Event {
    PE_DECLARE_CLASS(Event_LEFT_BUTTON_UP);
    Event_LEFT_BUTTON_UP() : m_x(0), m_y(0) {}

    virtual ~Event_LEFT_BUTTON_UP(){}
    PrimitiveTypes::Int16 m_x, m_y;
};

struct Event_RIGHT_BUTTON_UP : public Event {
    PE_DECLARE_CLASS(Event_RIGHT_BUTTON_UP);
    virtual ~Event_RIGHT_BUTTON_UP(){}
    PrimitiveTypes::Int16 m_x, m_y;
};
```

Trigger the click event whenever the mouse button is released after clicking.

```
    //Check for Button Down events
    if ((GetKeyState(VK_LBUTTON) & 0x80) != 0)
    {
        m_leftButtonPressed = true;
    }
    if (m_leftButtonPressed && ((GetKeyState(VK_LBUTTON) & 0x80) == 0))
    {
        POINT pt;
        Handle h(dbgName: "EVENT", neededSize: sizeof(Event_LEFT_BUTTON_UP));
        Event_LEFT_BUTTON_UP * pEvt = new (h) Event_LEFT_BUTTON_UP;
        GetCursorPos(&pt);
        ScreenToClient(hWnd: pWinApp->getWindowHandle(), &pt);
        pEvt->m_x = pt.x;
        pEvt->m_y = pt.y;
        m_pQueueManager->add( h, queueType: Events::QT_INPUT);
        m_leftButtonPressed = false;
    }
    //Check the mouse right button is pressed or not
```

```
    //Check the mouse right button is pressed or not
    if ((GetKeyState(VK_RBUTTON) & 0x80) != 0)
    {
        m_rightButtonPressed = true;
    }
    if (m_rightButtonPressed && ((GetKeyState(VK_RBUTTON) & 0x80) == 0))
    {
        POINT pt;
        Handle h(dbgName: "EVENT", neededSize: sizeof(Event_RIGHT_BUTTON_UP));
        new (h) Event_RIGHT_BUTTON_UP;
        Event_RIGHT_BUTTON_UP * pEvt = new (h) Event_RIGHT_BUTTON_UP;
        GetCursorPos(&pt);
        ScreenToClient(hWnd: pWinApp->getWindowHandle(), &pt);
        pEvt->m_x = pt.x;
        pEvt->m_y = pt.y;
        m_pQueueManager->add( h, queueType: Events::QT_INPUT);
        m_rightButtonPressed = false;
    }
```

Pass the event into the game queue

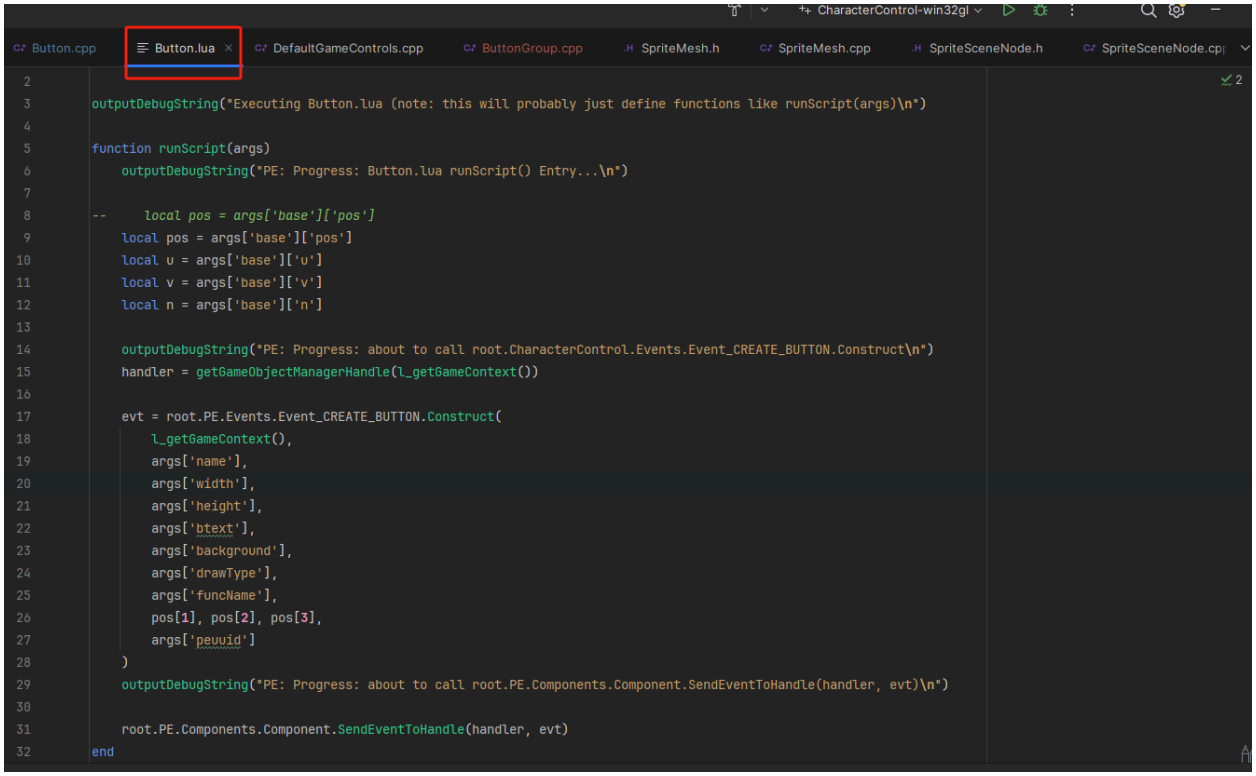
```
}
if (Event_LEFT_BUTTON_UP::GetClassId() == pEvt->getClassId())
{
    Handle h(dbgName: "EVENT", neededSize: sizeof(Event_LEFT_BUTTON_UP));
    Event_LEFT_BUTTON_UP *clickEvt = new(h) Event_LEFT_BUTTON_UP ;
    Event_LEFT_BUTTON_UP * pRealEvent = (Event_LEFT_BUTTON_UP*)(pEvt);
    clickEvt->m_x = pRealEvent->m_x;
    clickEvt->m_y = pRealEvent->m_y;

    m_pQueueManager->add(h, queueType: QT_GENERAL);
}
if (Event_RIGHT_BUTTON_UP::GetClassId() == pEvt->getClassId())
{
    Handle h(dbgName: "EVENT", neededSize: sizeof(Event_RIGHT_BUTTON_UP));
    Event_RIGHT_BUTTON_UP *clickEvt = new(h) Event_RIGHT_BUTTON_UP ;
    Event_RIGHT_BUTTON_UP * pRealEvent = (Event_RIGHT_BUTTON_UP*)(pEvt);
    clickEvt->m_x = pRealEvent->m_x;
    clickEvt->m_y = pRealEvent->m_y;

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

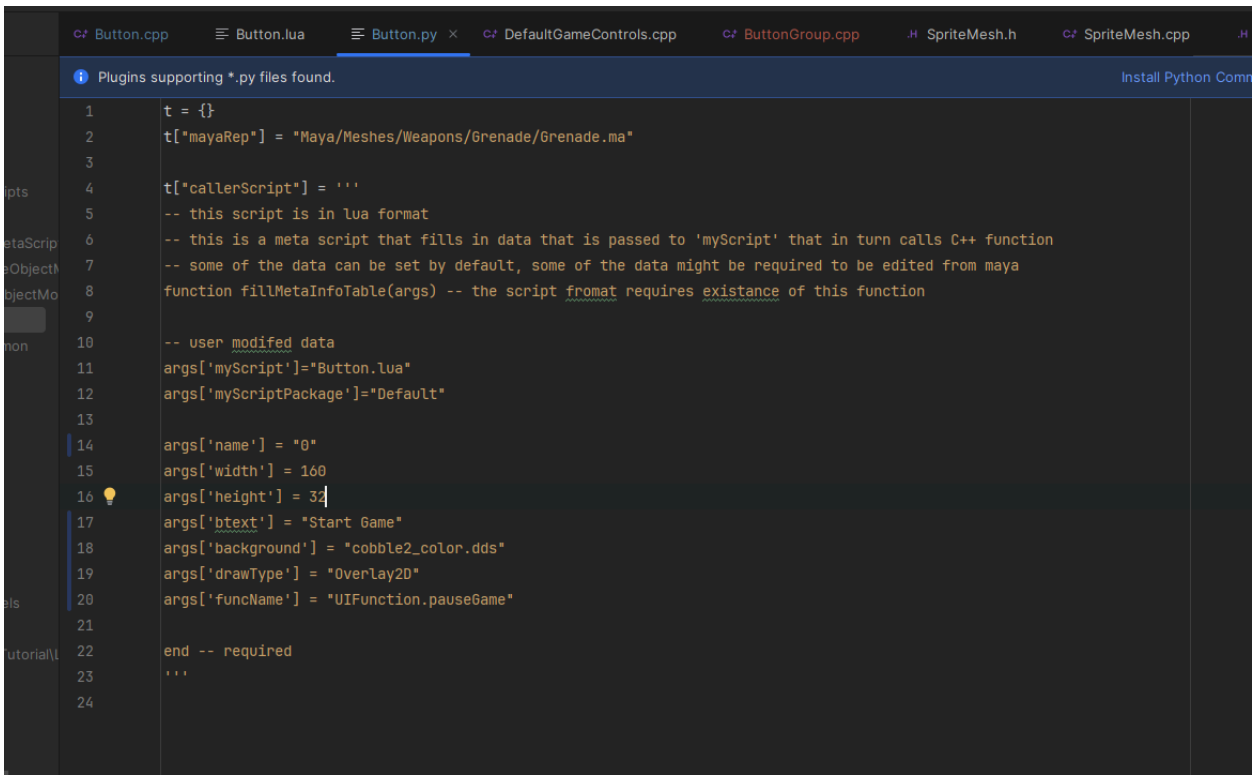
The button receives the event and text if the mouse is on the button.

```
void Button::addDefaultComponents()
{
    UIElement::addDefaultComponents();
    // event handlers
    PE_REGISTER_EVENT_HANDLER(Events::Event_PRE_GATHER_DRAWCALLS, Button::do_PRE_GATHER_DRAWCALLS);
    PE_REGISTER_EVENT_HANDLER(Events::Event_LEFT_BUTTON_UP, Button::do_TEST_ONCLICK);
}
```

```
void Button::do_TEST_ONCLICK(Events::Event *pEvt)
{
    Events::Event_LEFT_BUTTON_UP *pRealEvent = (Events::Event_LEFT_BUTTON_UP*) (pEvt);
    float xPos = (m_pos.m_x + 1.0f) / 2 * UIManager::Instance()->m_maxWidth;
    float yPos = (1.0f - ((m_pos.m_y + 1.0f) / 2)) * UIManager::Instance()->m_maxHeight;

    if (pRealEvent->m_x > xPos &&  pRealEvent->m_x < xPos + m_width &&  pRealEvent->m_y > yPos &&  pRealEvent->m_y < yPos + m_height)
    {
        onClick();
    }
}
```

I then created a data driven lua element that can pass data into button and buttongroup, which will auto generate UI based on what we created in maya. For Button,





We also need a createButton lua event to pass data into our code.

```
namespace PE
{
    namespace Events{
        struct Event_CREATE_BUTTON : Event_CREATE_MESH
        {
            PE_DECLARE_CLASS(Event_CREATE_BUTTON);
            Event_CREATE_BUTTON(int &threadOwnershipMask): Event_CREATE_MESH([&]threadOwnershipMask){}

            // override SetLuaFunctions() since we are adding custom Lua interface
            static void SetLuaFunctions(PE::Components::LuaEnvironment *pLuaEnv, lua_State *luaVM);

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

            char m_name[32];
            int m_width, m_height;
            char m_text[256];
            char m_background[256];
            char m_drawType[32];
            char m_funcName[64];

            Vector3 m_pos;

            bool isActive;

            PEUUID m_peuuid; // unique object id
        };
    }namespace Events
```

```
namespace PE {
    namespace Events
    {
        PE_IMPLEMENT_CLASS1(Event_CREATE_BUTTON, PE::Events::Event);

        void Events::Event_CREATE_BUTTON::SetLuaFunctions(PE::Components::LuaEnvironment *pLuaEnv, lua_State *luaVM)
        {
            static const struct luaL_Reg l_Event_CREATE_BUTTON[] = {
                {.name: "Construct", .func: l_Construct},
                {.name: NULL, .func: NULL} // sentinel
            };

            // register the functions in current lua table which is the table for Event_CreateButton
            luaL_register(luaVM, libname: 0, l_Event_CREATE_BUTTON);
        }
        int Events::Event_CREATE_BUTTON::l_Construct(lua_State* luaVM)
        {
            PE::Handle h(dbgName: "CREATE_BUTTON_EVENT", neededSize: sizeof(Events::Event_CREATE_BUTTON));

            // get arguments from stack
```

```
30
31          // get arguments from stack
32          int numArgs, numArgsConst;
33          numArgs = numArgsConst = 12;
34
35          PE::GameContext *pContext = (PE::GameContext*)(lua_touserdata(luaVM, -numArgs--));
36
37          Event_CREATE_BUTTON *pEvt = new(h) Event_CREATE_BUTTON([&]pContext->m_gameThreadThreadOwnershipMask);
38
39          const char* name = lua_tostring(luaVM, -numArgs--);
40          float width = (float)lua_tonumber(luaVM, -numArgs--);
41          float height = (float)lua_tonumber(luaVM, -numArgs--);
42          const char* text = lua_tostring(luaVM, -numArgs--);
43          const char* bgFile = lua_tostring(luaVM, -numArgs--);
44          const char* drawType = lua_tostring(luaVM, -numArgs--);
45          const char* funcName = lua_tostring(luaVM, -numArgs--);
46
47          Vector3 buttonPos;
48          buttonPos.m_x = (float)lua_tonumber(luaVM, -numArgs--);
49          buttonPos.m_y = (float)lua_tonumber(luaVM, -numArgs--);
50          buttonPos.m_z = (float)lua_tonumber(luaVM, -numArgs--);
51
52          pEvt->m_peuuid = LuaGlue::readPEUUID(luaVM, -numArgs--);
53          StringOps::writeToString(name, pEvt->m_name, maxSize: 255);
54          StringOps::writeToString(drawType, pEvt->m_drawType, maxSize: 255);
55          pEvt->m_width = width;
56          pEvt->m_height = height;
57          StringOps::writeToString(text, pEvt->m_text, maxSize: 255);
58          StringOps::writeToString(bgFile, pEvt->m_background, maxSize: 255);
59          StringOps::writeToString(funcName, pEvt->m_funcName, maxSize: 255);
60
61          pEvt->m_pos = buttonPos;
62
63          lua_pop(luaVM, numArgsConst);
```

For ButtonGroup,

```
1   -- expected arguments:
2
3   outputDebugString("Executing ButtonGroup.lua \n")
4
5   function runScript(args)
6       outputDebugString("PE: Progress: ButtonGroup.lua runScript() Entry...\n")
7
8   --      local pos = args['base']['pos']
9       local pos = args['base']['pos']
10      local list = args['list']
11
12      outputDebugString("PE: Progress: about to call root.CharacterControl.Events.Event_CREATE_BUTTONGROUP.Construct\n")
13      handler = getGameObjectManagerHandle(l_getGameContext())
14
15      evt = root.PE.Events.Event_CREATE_BUTTONGROUP.Construct(
16          l_getGameContext(),
17          args['name'],
18          args['buttonlayout'],
19          pos[1], pos[2], pos[3],
20          args['isActive'],
21          args['peuuid'],
22          list[1], list[2], list[3]
23      )
24      outputDebugString("PE: Progress: about to call root.PE.Components.Component.SendEventToHandle(handler, evt)\n")
25
26      root.PE.Components.Component.SendEventToHandle(handler, evt)
27  end
28
```

```
t = {}
t["mayaRep"] = "Maya/Meshes/Arrow/arrow.mb"

t["callerScript"] = '''
-- this script is in lua format
-- this is a meta script that fills in data that is passed to 'myScript' that in turn calls C++ function
-- some of the data can be set by default, some of the data might be required to be edited from maya
function fillMetaInfoTable(args) -- the script fromat requires existance of this function

-- user modifed data
args['myScript']="ButtonGroup.lua"
args['myScriptPackage']="Default"

args['name'] = "0"
args['buttonlayout'] = "vertical"
args['list'] = {"0", "2", "3"}
args['isActive'] = 1

end -- required
'''
```

```
namespace PE{
    namespace Events{
        struct Event_CREATE_BUTTONGROUP : Event_CREATE_MESH
        {
            PE_DECLARE_CLASS(Event_CREATE_BUTTONGROUP);

            Event_CREATE_BUTTONGROUP(int &threadOwnershipMask): Event_CREATE_MESH([&]threadOwnershipMask)
            {

            }

            // override SetLuaFunctions() since we are adding custom Lua interface
            static void SetLuaFunctions(PE::Components::LuaEnvironment *pLuaEnv, lua_State *luaVM);

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

            char m_name[32];
            char m_layout[32];
            const char* m_bNames[8];
            PrimitiveTypes::Int16 m_active;

            PEUUID m_peuuid; // unique object id
        };
    } namespace Events
    namespace Components {

        struct ButtonGroup : UIElement
```

```
PE_IMPLEMENT_CLASS1(Event_CREATE_BUTTONGROUP, PE::Events::Event);
void Events::Event_CREATE_BUTTONGROUP::SetLuaFunctions(PE::Components::LuaEnvironment *pLuaEnv, lua_State *luaVM)
{
    static const struct luaL_Reg l_Event_CREATE_BUTTONGROUP[] = {
        {name: "Construct", func: l_Construct},
        {name: NULL, func: NULL} // sentinel
    };

    // register the functions in current lua table which is the table for Event_CreateButton
    luaL_register(luaVM, libname: 0, l_Event_CREATE_BUTTONGROUP);
}
int Events::Event_CREATE_BUTTONGROUP::l_Construct(lua_State* luaVM)
{
    PE::Handle h(dbgName: "CREATE_BUTTON_EVENT", neededSize: sizeof(Events::Event_CREATE_BUTTONGROUP));

    // get arguments from stack
    int numArgs, numArgsConst;
    numArgs = numArgsConst = 11;

    PE::GameContext *pContext = (PE::GameContext*)(lua_touserdata(luaVM, -numArgs--));

    Event_CREATE_BUTTONGROUP *pEvt = new(h) Event_CREATE_BUTTONGROUP([&]pContext->m_gameThreadThreadOwnershipMask);

    const char* name = lua_tostring(luaVM, -numArgs--);
    const char* layout = lua_tostring(luaVM, -numArgs--);

    Vector3 buttonGroupPos;
    buttonGroupPos.m_x = (float)lua_tonumber(luaVM, -numArgs--);
    buttonGroupPos.m_y = (float)lua_tonumber(luaVM, -numArgs--);
```

```
        pEvt->m_active = isActive;

        pEvt->m_peuuid = LuaGlue::readPEUUID(luaVM, -numArgs--);
        pEvt->m_pos = buttonGroupPos;

        StringOps::writeToString(name, pEvt->m_name, maxSize: 255);
        StringOps::writeToString(layout, pEvt->m_layout, maxSize: 255);

        int i = 0;
        while (i < 8)
        {
            if (!lua_isnil(luaVM, -numArgs) && numArgs > 0)
            {
                const char* curButtonName = lua_tostring(luaVM, -numArgs--);
                pEvt->m_bNames[i] = curButtonName;
            }
            else
            {
                pEvt->m_bNames[i] = "";
            }
            i++;
        }

        lua_pop(luaVM, numArgsConst);
        LuaGlue::pushTableBuiltFromHandle(luaVM, h);

        return 1;
    }
```

Then I added the function mapping for button becaus in Maya we can only specify a string.

```
{
    namespace Components
    {

        struct UIFunction: Component
        {
            PE_DECLARE_CLASS(UIFunction);

            UIFunction(PE::GameContext& context, PE::MemoryArena arena, Handle hMyself):Component([&]context, arena, hMyself){}
            virtual ~UIFunction() {}

            virtual void addDefaultComponents();

            std::function<void()> findFunction(const char* funcName);

            void pauseGame();

            void resumeGame();

            void jumpMenu(char* menuName);

            void exitGame();

            void defaultFunc();

        };

    } namespace Components ;
} namespace PE
```

```
        }
        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(); };
            }
            // If function not found, return an empty std::function or a default function
            return [this]()->void { this->defaultFunc(); };
        }

        void UIFunction::pauseGame()
        {
            PE::Handle h( dbgName: "Event_PAUSE", neededSize: sizeof(Events::Event_PAUSE));
            Events::Event_PAUSE *pEvt = new (h) Events::Event_PAUSE;
            Events::EventQueueManager::Instance()->add(h, queueType: Events::QT_GENERAL);
            jumpMenu("1");
        }
```

```cpp
void UIFunction::resumeGame()
{
    PE::Handle h(dbgName: "Event_RESUME", neededSize: sizeof(Events::Event_RESUME));
    Events::Event_RESUME *pEvt = new (h) Events::Event_RESUME;
    Events::EventQueueManager::Instance()->add(h, queueType: Events::QT_GENERAL);
    jumpMenu("0");
}

void UIFunction::jumpMenu(char* menuName)
{
    PE::Handle h(dbgName: "Event_PAUSE", neededSize: sizeof(Events::Event_JUMP_MENU));
    Events::Event_JUMP_MENU *pEvt = new (h) Events::Event_JUMP_MENU;
    pEvt->m_name = menuName;
    Events::EventQueueManager::Instance()->add(h, queueType: Events::QT_GENERAL);
}

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

} namespace Components ;
```

I created an event respectively,

```
 8
 9            PE_IMPLEMENT_CLASS1(Event_FLY_CAMERA, Event);
10            PE_IMPLEMENT_CLASS1(Event_ROTATE_CAMERA, Event);
11  +         PE_IMPLEMENT_CLASS1(Event_PAUSE, Event);
12  +         PE_IMPLEMENT_CLASS1(Event_RESUME, Event);
13  +         PE_IMPLEMENT_CLASS1(Event_EXIT_GAME, Event);
14  +         PE_IMPLEMENT_CLASS1(Event_JUMP_MENU, Event);
15  +
16    };
17    };
```

```
34              Vector3 m_relativeRotate; //2D screenspace rotate
35      };
36  +
37  + struct Event_PAUSE : public Event {
38  +           PE_DECLARE_CLASS(Event_PAUSE);
39
40  +           Event_PAUSE() {}
41  +           virtual ~Event_PAUSE(){}
42  + };
43  +
44  + struct Event_RESUME : public Event {
45  +           PE_DECLARE_CLASS(Event_RESUME);
46  +
47  +           Event_RESUME() {}
48  +           virtual ~Event_RESUME(){}
49  + };
50  +
51  + struct Event_EXIT_GAME : public Event {
52  +           PE_DECLARE_CLASS(Event_EXIT_GAME);
53  +
54  +           Event_EXIT_GAME() {}
55  +           virtual ~Event_EXIT_GAME(){}
56  + };
57  +
58  + struct Event_JUMP_MENU : public PE::Events::Event {
59  +           PE_DECLARE_CLASS(Event_JUMP_MENU);
60  +
61  +           Event_JUMP_MENU(){}
62  +           virtual ~Event_JUMP_MENU(){}
63  +
64  +           PrimitiveTypes::Int16 m_index;
65  + };
66      }; // namespace Events
67      }; // namespace PE
68
```

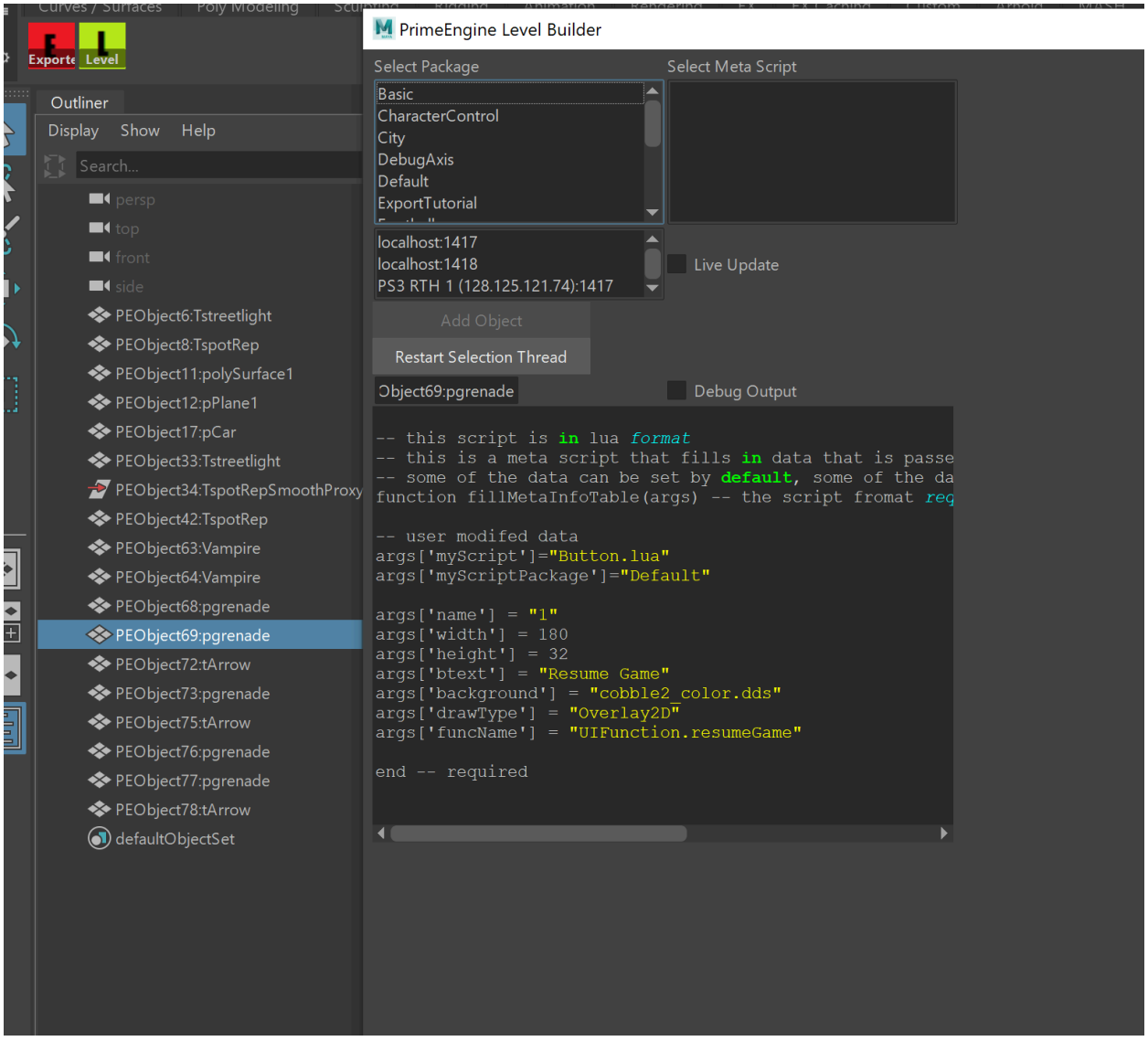
In GameObjectManager, I passed all creation event into it, and let it pass it to the handling component

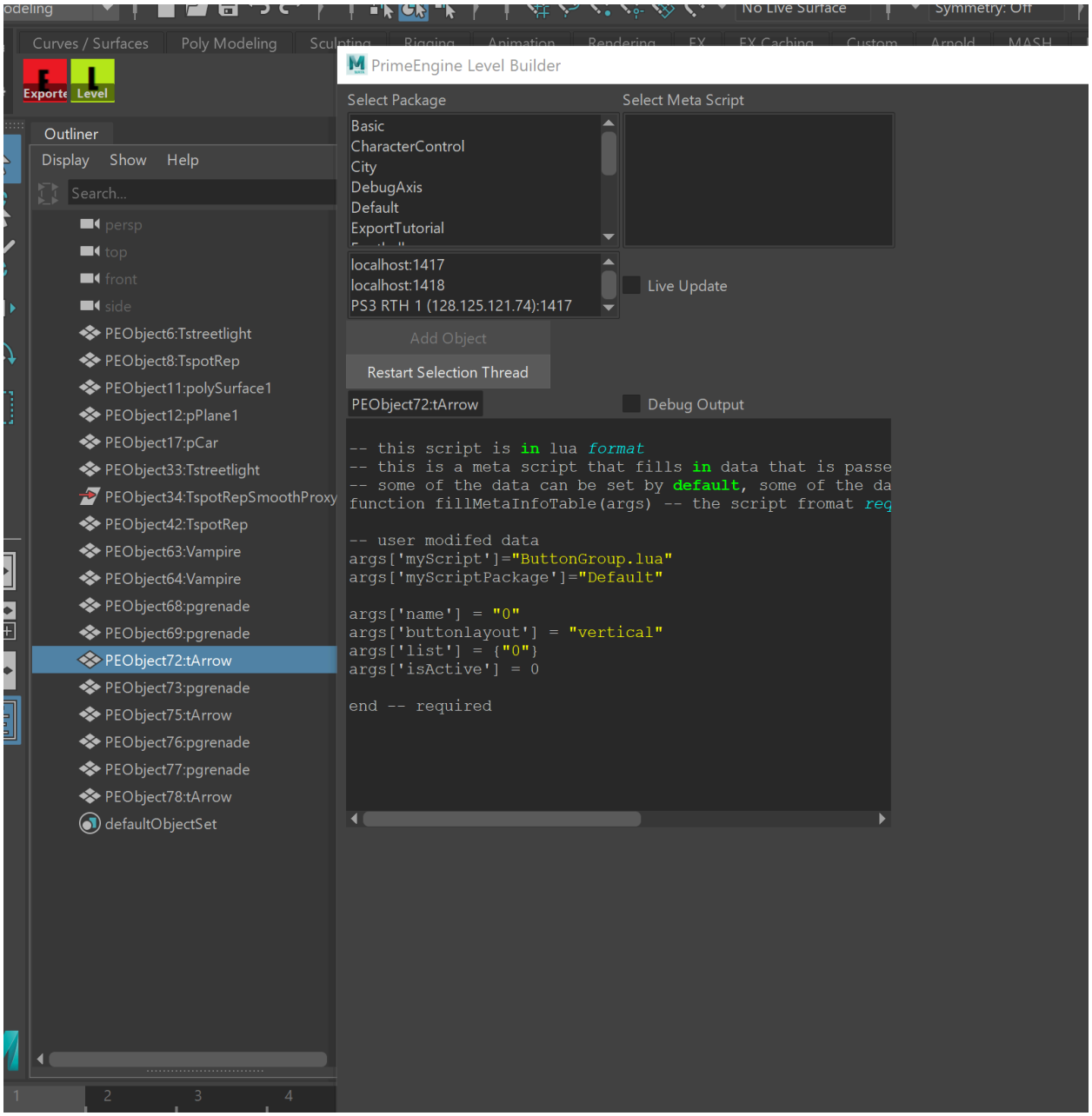
```
56              m_pContext->getLuaEnvironment()->pop();
57      }
58  + void GameObjectManager::do_CREATE_BUTTON(Event *pEvt)
59  + {
60  +           Event_CREATE_BUTTON *pRealEvent = (Event_CREATE_BUTTON*) pEvt;
61  +           UIManager::Instance()->createButton(pRealEvent);
62  +           m_pContext->getGPUScreen()->ReleaseRenderContextOwnership(pRealEvent->m_threadOwnershipMask);
63  + }
64
65      void GameObjectManager::do_CREATE_MESH(Events::Event *pEvt)
66      {
```

For pausing and resuming event, I simply added a variable to stop frametime from increasing

```
44              {
45                              // physics kick off
46                      m_pContext->getPhysicsManager()->handleEvent(pGeneralEvt);
47  +           }
48  +           else if (Event_PAUSE::GetClassId() == pGeneralEvt->getClassId())
49  +           {
50  +                   isPaused = true;
51  +           }
52  +           else if (Event_RESUME::GetClassId() == pGeneralEvt->getClassId())
53  +           {
54  +                   isPaused = false;
55              }
56              else
57              {

78          m_pContext->getLog()->m_isActivated = false;
79
80              float gameThreadPostDrawFrameTime = m_hTimer.getObject<Timer>()->TickAndGetTimeDeltaInSeconds();
81  +
82  +           if (isPaused)

83  +           {
84  +                   m_frameTime = 0;
85  +           } else
86  +           {
87  +                   m_frameTime = gameTimeBetweenFrames + gameThreadPreDrawFrameTime + gameThreadDrawWaitFrameTime +
    gameThreadDrawFrameTime + gameThreadPostDrawFrameTime;
88  +           }
89
90          if (m_frameTime > 10.0f)
91              m_frameTime = 0.1f;
```

Last, I added the newly created elements into Maya and exported a new level file.





Loading this file gives the video. Although the video looks simple, there is an extendable architecture behind this. If there could be more time, I will add a scalable and alignable text for the background. A more flexible button group with scalability and alignability. Also there could be a customized single color background behind each element.