CS4162 Virtual Worlds

Final Report for the group of

Philip Waldron 14173026
Robert McCahill 14150611
David Vaughan 14155974

Contribution

Philip Waldron: Built some aspects of the helicopter body, built the missiles and the missile launcher. Wrote and/or edited the following scripts; Movement Script, Left and Right Launcher Button Scripts, Left and Right Missile Launcher Scripts, Machine Gun Button Script, Missile Velocity Script and the Smoke script. Edited the 3D model used for the body of the helicopter.

David Vaughan: Built some aspects of the helicopter body and built the the machine gun, bullets and seats. Wrote and/or edited the following scripts; Bullets Script, Machine Gun Firing Script.

Robert McCahill: Built some aspects of the helicopter body and built the rotors. Wrote and/or edited the following scripts; Main Window Rotation Script, Left and Right Window Rotation Scripts, Main Rotor Rotation Script, Secondary Rotor Rotation Script and the Sitting Script.

All group member contributed equally to the project.

Functions of Interactive Object

The Apache Attack Helicopter has numerous interactive functions. The first of which being its windows. The windows can be opened and closed by being clicked on. This is so the user can enter the helicopter.

There are two interactive seats inside the helicopter. The back seat is the pilot seat, and the front is the co-pilot seat. If the user sits in the pilot seat, they will have instructions on what they need to do to fly the helicopter appear in the chat. They may ask for help on what exactly they can do, or tell the helicopter to 'Start' in chat. This will cause the propellers on the helicopter to rotate and a timer to count down from three seconds as it powers up. Once the timer counts down, the user will now be able to fly the helicopter by using their normal keyboard movement keys. The user may say 'Stop' in the chat to power down the helicopter.

The co-pilots seat will make the users camera go into a 'first-person' perspective. The user will have three buttons in front of them. Each of which fire one of the helicopters three weapons. The left and right buttons fire the left and right missile launchers respectively. The middle button fires the helicopters chain gun.

Description of Scripts

Machine Gun Firing Script

```
vector relativePosOffset = <0.0, -0.2, 0.0>;
rotation relativeRot = <0.0, 0.0, 0.00, 0.250>;
integer startParam = 1;
integer count = 0;
integer listen_handle;
make_particles()
{llParticleSystem([
        PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_EXPLODE,
        PSYS_SRC_MAX_AGE, 0.,
        PSYS_PART_MAX_AGE, 1.,
        PSYS_SRC_BURST_RATE, 0.01,
        PSYS_SRC_BURST_PART_COUNT, 500,
        PSYS_SRC_BURST_RADIUS, 0.01,
        PSYS_SRC_BURST_SPEED_MIN, 1.,
        PSYS_SRC_BURST_SPEED_MAX, 2.,
        PSYS_SRC_ACCEL, <0.0,0.0,-1.0>,
        PSYS_PART_START_COLOR, <1.000, 0.863, 0.000>,
        PSYS_PART_END_COLOR, <1.000, 0.863, 0.000>,
        PSYS_PART_START_ALPHA, 0.9,
        PSYS_PART_END_ALPHA, 0.0,
        PSYS_PART_START_SCALE, <.05,.1,0>,
        PSYS_PART_END_SCALE, <.01,.1,0>,
        PSYS_PART_FLAGS,
        PSYS_PART_EMISSIVE_MASK |
        PSYS_PART_INTERP_COLOR_MASK |
        PSYS_PART_INTERP_SCALE_MASK |
        PSYS_PART_FOLLOW_VELOCITY_MASK |
        PSYS_PART_WIND_MASK]);}
default
{
    state_entry()
    {
        listen_handle = llListen(3027, "", NULL_KEY, "");
    }
    listen(integer channel, string name, key id, string message)
    {
        list words = llParseString2List(message, [" ", ","], [""]);
        if (llList2String(words, 0) == "Fire")
        {
            do
            {
                vector myPos = llGetPos();
                vector rezPos = myPos+relativePosOffset;
                rotation myRot = llGetRot();
                rotation rezRot = relativeRot*myRot;
                llRezAtRoot("Bullet", rezPos, ZERO_VECTOR, rezRot,
startParam);
                llSleep(2.0);
                count = count + 1;
            }
            while (count < 50);
            llParticleSystem([]);
            count = 0;
            llResetScript();
        }
    }
}
```

The script shown above is the script which gets placed into the helicopters 'Chain gun'. This script was made so that the chain gun could have a particle effect when it was firing. The beginning of the script declares a particle system called 'make_particles'. The basic template of this particle system was taken from the Virtual Worlds lab which introduced us to particle systems. It had its different values altered until the particle system was similar to the effect that was desired.

Its default entry state was set so that it would listen to a specific chat channel. The chain gun would hear its command to run when a button was pressed and the button's script sent down the information required. The chain gun then sends a message to chat, to indicate that it is firing. This script was written by our team, while using the virtual worlds lecture slides as a guide.

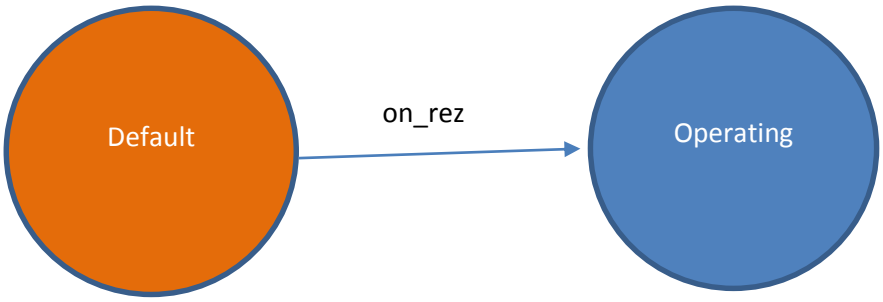
When the gun fired, it rezzed an object called 'Bullet' and applied the variables and parameters to this object. It continually rezzed these bullets until a while loop counter hit the number of bullets that are allowed to be fired. This script was created by our team with help from the LSL Wikipedia.

Bullet Script

```
default
{
    on_rez(integer start_param)
    {
        state operating;
    }
}

state operating
{
    state_entry()
    {
        llSetStatus(STATUS_PHYSICS, TRUE);
        llApplyImpulse(0.4*llRot2Left(llGetRot())*-1.0, 1);
    }

    land_collision(vector pos)
    {
        llSleep(0.2);
    }
}
```

This script was applied to the bullet the rezzed when the chain gun was interacted with. On being rezzed, it enters a state by the name of 'operating'. In this state, physics gets applied to the object, along with impulsion to make the bullet move on being rezzed. When the bullet collides with land, it will disappear after a brief period of time.

Transition Diagram for Bullet Script



Missile Launcher Scripts

```
vector relativePosOffset = <0.0, 0.0, -1.0>;
rotation relativeRot = <0.707107, 0.707107, 0.707107, 0.707107>; // Rotated
to be in line with prim
integer startParam = 10;
integer listen_handle;

default
{
    state_entry()
    {
        listen_handle = llListen(3028, "", NULL_KEY, "");
    }

    listen(integer channel, string name, key id, string message)
    {
        list words = llParseString2List(message, [" ", ","], [""]);
        if (llList2String(words, 0) == "Fire")
        {
            vector myPos = llGetPos();
            vector rezPos = myPos+relativePosOffset;
            rotation myRot = llGetRot();
            rotation rezRot = relativeRot*myRot;
            llRezAtRoot("Missile", rezPos, ZERO_VECTOR, rezRot,
startParam);
            llSleep(2.0);
        }
    }
}
```

The left and right missile launcher scripts in a very similar fashion to the machine gun firing script. Each missile launcher will listen to a different channel. The script will listen for the message "Fire" to be sent in the corresponding chat channel (Through a button). When "Fire" is heard, the script will rez a missile primitive using the variables to orientate it correctly with the launcher current position and rotation. The script were created using the LSL wiki and lecture slides as a basic guide.

Weapon Button Scripts

```
default
{
    touch_start(integer total_number)
    {
        llSay(3028, "Fire");
        llSleep(2.0);
        llResetScript();
    }
}
```

The above script is the same for all three buttons to fire the left and right missile launchers and the machine gun, with the only difference being the channel in which the "Fire" message is sent. When the button is pressed, the message will be sent to the appropriate channel, to heard by the linked object listening to that channel.

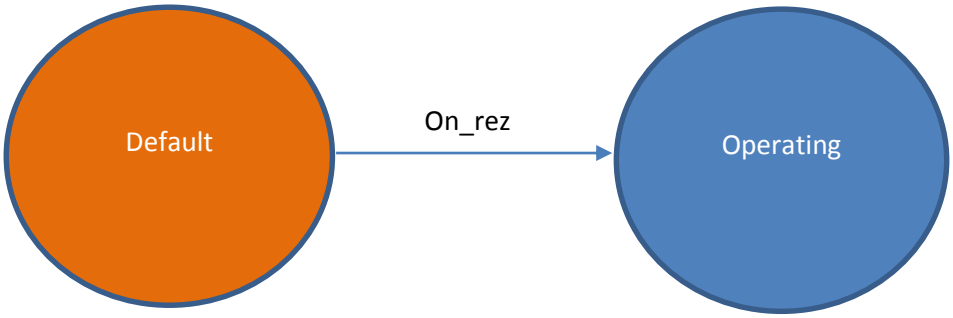
Missile Velocity Script

```
default
{
    on_rez(integer start_param)
    {
        state operating;
    }
}

state operating
{
    state_entry()
    {
        llSetStatus(STATUS_PHYSICS, TRUE);
        llApplyImpulse(10*llRot2Left(llGetRot())*-1.0, 1);
    }

    land_collision(vector pos)
    {
        llSleep(0.2);
        llDie();
    }
      }
```

The above script is placed in the missile primitive, which is placed in the missile launcher. The default state of this script changes the state to operating when the missile in rezzed. When operating, physics is activated for the missile, so force may be applied to it. An impulse is then applied to the missile in the direction that the front of the missile is facing.

When the missile detects a collision with another surface, after 0.2 seconds the missile will disappear.

Transition Diagram for Missile Velocity Script



Smoke Script

```
default
{
    state_entry()
    {
        llParticleSystem([
            PSYS_PART_FLAGS ,
            PSYS_PART_WIND_MASK |
            PSYS_PART_INTERP_COLOR_MASK |
            PSYS_PART_INTERP_SCALE_MASK |
            PSYS_PART_EMISSIVE_MASK |
            PSYS_PART_FOLLOW_VELOCITY_MASK |
            PSYS_PART_EMISSIVE_MASK,

            PSYS_SRC_PATTERN,           PSYS_SRC_PATTERN_ANGLE_CONE,
            PSYS_SRC_TEXTURE,           "sprite-particle-cloud",
            PSYS_SRC_MAX_AGE,           0.0,
            PSYS_PART_MAX_AGE,          5.0,
            PSYS_SRC_BURST_RATE,        0.1,
            PSYS_SRC_BURST_PART_COUNT,  20,
            PSYS_SRC_BURST_RADIUS,      0.5,
            PSYS_SRC_BURST_SPEED_MAX,   .5,
            PSYS_SRC_ACCEL,             <0.0,0,.05>,
            PSYS_PART_START_COLOR,      <1.0,1.0,1.0>,
            PSYS_PART_END_COLOR,        <1.0,1.0,1.0>,
            PSYS_PART_START_ALPHA,      0.9,
            PSYS_PART_END_ALPHA,        0.0,
            PSYS_PART_START_SCALE,      <.25,.25,.25>,
            PSYS_PART_END_SCALE,        <.75,.75,.75>,
            PSYS_SRC_ANGLE_BEGIN,       0 * DEG_TO_RAD,
            PSYS_SRC_ANGLE_END,         45 * DEG_TO_RAD,
            PSYS_SRC_OMEGA,             <0.0,0.0,0.0>
                ]);
    }
}
```

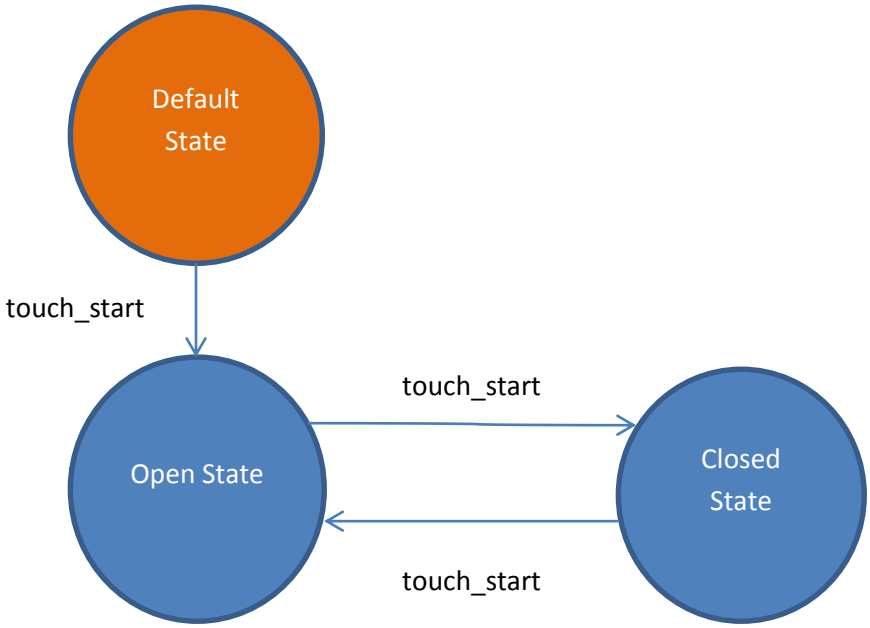
The above script is placed into the missile primitive to give it particle properties. It takes on the image "sprite-particle-cloud" for the particle texture". This particle system is taken from the lab particle systems, with values edited and some properties removed and added.

Window Rotation Scripts

```
vector center_of_rotation = <0.0, 0.0, 0.25>;
vector rot = <0.0, 90.0, 0.0>;

rotate(vector rot)
{
    //creates a rotation constant
    rotation vRotArc = llEuler2Rot(rot * DEG_TO_RAD);

    //updates center_of_rotation to the current local rotation
    vector local_center_of_rotation = center_of_rotation * llGetLocalRot();

    //rotates center_of_rotation to get the motion caused by the desired rotation
    vector vPosRotOffset = local_center_of_rotation * vRotArc;

    //difference between the two - equal to the difference between the current
    //and the new position of the prim
    vector vPosOffsetDiff = local_center_of_rotation - vPosRotOffset;

    //the new position of the prim relative to the root prim
    vector vPosNew = llGetLocalPos() + vPosOffsetDiff;

    //the new rotation of the prim relative to the root prim
    rotation vRotNew = llGetLocalRot() * vRotArc;

    llSetPrimitiveParams( [PRIM_POSITION, vPosNew, PRIM_ROTATION,
vRotNew/llGetRootRotation()] );
}

default //initially closed
{
    touch_start(integer num_detected)
    {
        if (llDetectedKey(0) == llGetOwner()) state open;
    }
}

state open
{
    state_entry()
    {
        rotate(rot);
    }

    touch_start(integer num_detected)
    {
        state closed;
    }
}

state closed
{
    state_entry()
    {
        rotate(-rot);
    }

    touch_start(integer num_detected)
    {
        state open;
    }
```

The above script is the window rotation script to allow entry in and out of the helicopter. It is placed in each window prim with variables adjusted accordingly depending on the window in question. We picked this script as it is independent from the prims it's linked with which prevents the entire object from rotating. In its default state it waits for a touch event to occur. If the owner clicks on the prim, it changes its state to open. Upon entry the rotation event occurs, and the script is then idle until another touch event occurs, on which it transitions to the closed state. Upon entering it reverses the rotation to return the prim to its original rotation, and remains idle until another touch event occurs. The base of the script was originally borrowed from an object from the labs.

Transition Diagram for Window Rotation Scripts



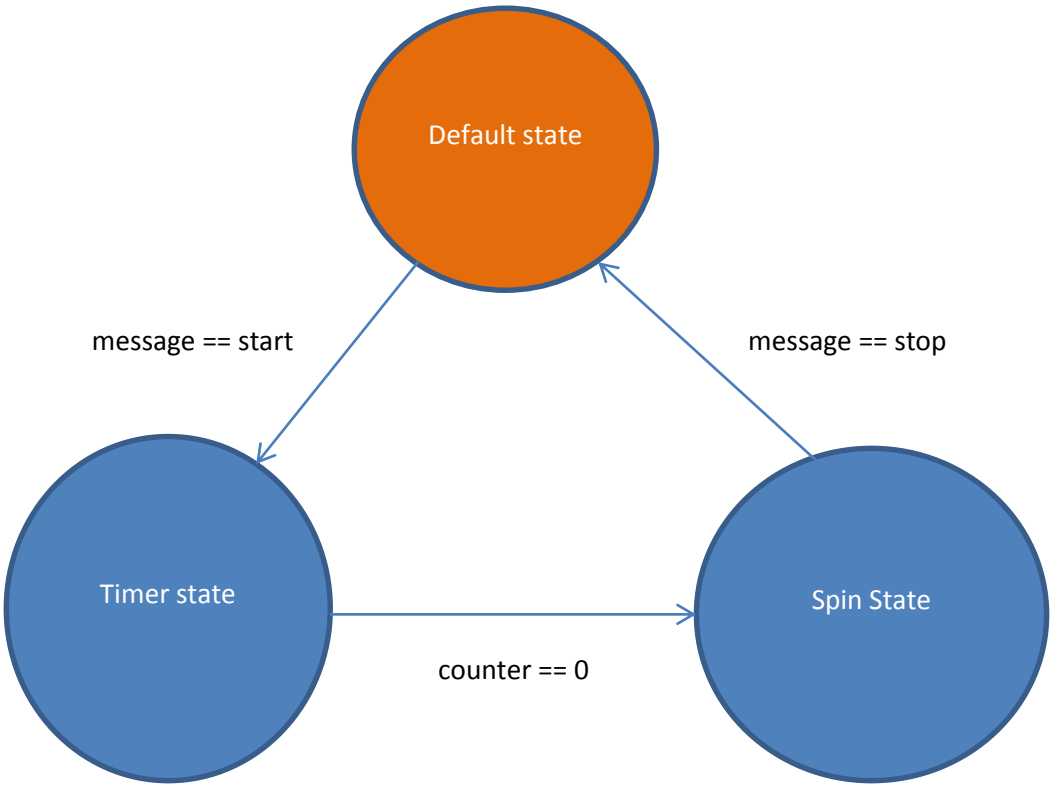
Propeller Script

```
float counter = 4.0;
float gap = 1.0;
integer listen_handle;
default
{
     state_entry()
     {
          counter = 4.0;
          listen_handle = llListen(0,"",NULL_KEY,"");
          llTargetOmega(llRot2Up(llGetLocalRot()), 0, 0);
     }

     listen(integer channel, string name, key id, string message)
     {
          message = llToLower(message);
          if (message == "start")
          {
               state initiateTimer;
          }

     }
}
state initiateTimer
{
     state_entry()
     {
          llSetTimerEvent(1.0);
     }
     timer()
     {
          counter = counter - gap;
          if(counter > 0)
          {
               llSay(0, (string)counter+"...");
          }
          if(counter == 0)
          {
               llSay(0, "Starting!");
               state spin;
          }
     }
}
state spin
{
     state_entry()
     {
          listen_handle = llListen(0,"",NULL_KEY,"");
          llTargetOmega(llRot2Up(llGetLocalRot()), 10, 1.0);
     }

     listen(integer channel, string name, key id, string message)
     {
          message = llToLower(message);
          if(message == "stop")
          {
               state default;
          }
     }

}
```

The above script controls the spinning of the propeller blades. In its default state it ensures the propeller is not rotating by setting the speed to 0 and waits for a listen event. If the message `received` **is "start" then the code transitions to the initiateTimer event. A timer event starts, with a counter having the initial value of 4.0. An if statement detects when the timer is above zero, so when the timer starts and goes to 3 it goes into the if event and says the time remaining into chat. Upon reaching 0, the first if statement is no longer true and the script goes into the second if statement, initiating a transition to the spin state.**

In the spin state, The propeller is set to start spinning immediately at a set rotation speed, and a listen event is set up. If someone types "stop" into the default chat channel, the if statement succeeds and the state transitions back to default, upon which the propeller is set to stop spinning, and the listen event is set up again if the user wishes to start the timer again. One propeller holds the main script, while the rest hold a "secondary" propeller script that is the same in every way but does not contain the llSay command in the timer event to prevent duplication of messages.

Transition Diagram for Propeller Rotation Script



Sitting Script

```
//GLOBAL VARIABLES
string animation = "ANIMATIONNAME";                //change to animation name
vector sittarget = < 0.0, 0.3, -0.2>;               //adjust sittarget offset
vector sitangle = < 0.0, 0.0, 90.0>;                //adjust sittarget rotation in
degrees

//WHO CAN USE THIS?  ONLY OWNER?
integer only_owner = 0;                            //toggle access: 1 = only owner, 0
= anyone

integer OBJHIDE_ON = 0;                            //toggle object hide (poseball
action): 1 = on, 0 = off


//OTHER GLOBAL VARIABLE DECLARATIONS
rotation sitrotation;
key owner;
key sitter = NULL_KEY;
integer SITTING;
integer LN;
integer LS;

//USER-DEFINED FUNCTION
integer test_sit() {
     if(LS == 1) {
          if(llAvatarOnLinkSitTarget(LN) != NULL_KEY) return 1;
          else return 0;
     }
     else if(LS == 0) {
          if(llAvatarOnSitTarget() != NULL_KEY) return 1;
          else return 0;
     }
     else return 0;
}

//START HERE
default {
     state_entry() {
          SITTING = 0;
          if(only_owner == 1) owner = llGetOwner();
          if(llGetNumberOfPrims() > 1) {
               LN = llGetLinkNumber();
               LS = 1;
          }
          else {
               LN = LINK_SET;
               LS = 0;
          }
          sitrotation = llEuler2Rot(sitangle * DEG_TO_RAD);          //deg to rad &
--> rot
          llSitTarget(sittarget,sitrotation);                        //set sittarget
          llSetClickAction(CLICK_ACTION_SIT);                        //click action
to sit
          if(SITTXT_ON) llSetSitText(SITTEXT);                       //set sit menu
text
          if(OBJHIDE_ON) llSetLinkAlpha(LN,1.0,ALL_SIDES);           //show object
     }

     on_rez(integer num) {
          llResetScript();
     }

     changed(integer change) {
          if(change & CHANGED_LINK) {                                //links
changed!
```

```
            if(SITTING == 0 && LS == 1) sitter = llAvatarOnLinkSitTarget(LN);
            if(SITTING == 0 && LS == 0) sitter = llAvatarOnSitTarget();
            if(only_owner == 1 && sitter != owner && sitter != NULL_KEY) {
                llUnSit(sitter);
                llInstantMessage(sitter,"You are not permitted to sit here.");
                sitter = NULL_KEY;
            }
            else if(SITTING == 0 && sitter != NULL_KEY) {          //someone
sitting?
                SITTING = 1;
                llRequestPermissions(sitter,PERMISSION_TRIGGER_ANIMATION);
            }
            else if(SITTING == 1 && test_sit() == 0) {             //I think
someone stood up!
                if(llGetPermissions() & PERMISSION_TRIGGER_ANIMATION)
llStopAnimation(animation);
                if(HOV_ON) llSetText(HOVERTEXT,COLOR,1.0);         //show hover
text
                if(OBJHIDE_ON) llSetLinkAlpha(LN,1.0,ALL_SIDES);   //show object
                SITTING = 0;
                sitter = NULL_KEY;
            }
        }
    }

    run_time_permissions(integer perm) {
        if(perm & PERMISSION_TRIGGER_ANIMATION) {
            if(HOV_ON) llSetText("",COLOR,1.0);                    //hide hover
text
            if(OBJHIDE_ON) llSetLinkAlpha(LN,0.0,ALL_SIDES);       //hide object
            llStopAnimation("sit");                                //stop normal
sit
            llStartAnimation(animation);                           //and use this
animation
        }
    }
}
```

The above script allows an object to be sat on simply by left clicking on it and has very little edited from the original script. However, the hover text value is set to zero as it was un-needed and the parameters for where the avatar sits were slightly adjusted to suit the seat in the helicopter.

Movement Script

```
// * Portions based on a public-domain flight script from Jack Digeridoo.
// * Portions based on hoverboard script by Linden Lab.

integer sit = FALSE;
integer listenTrack;
integer brake = TRUE;

float X_THRUST = 25;
float Z_THRUST = 25;

float xMotor;
float zMotor;

key agent;
key pilot;

vector SIT_POS = <0.0, 0.1, 0.0>;
vector SIT_ANGLE = < 0.0, 0.0, 90.0>;
rotation SIT_ROTATION;
vector CAM_OFFSET = <0.0, -30.0, 10.0>;
vector CAM_ANG = <0.0, -20.0, 90.0>;

listenState(integer on) // start/stop listen
{
    if (listenTrack)
    {
        llListenRemove(listenTrack);
        listenTrack = 0;
    }
    if (on) listenTrack = llListen(0, "", pilot, "");
}

help()
{
    llWhisper(0,"Welcome user, you may use these commands to fly");
    llWhisper(0,"Say START to begin moving.");
    llWhisper(0,"Say STOP to stop moving.");
    llWhisper(0,"PgUp/PgDn or E/C = hover up/down");
    llWhisper(0,"Arrow keys or WASD = forward, back, left, right");
    llWhisper(0,"Say HELP to display help.");
}

default
{
    state_entry()
    {
        llSetClickAction(CLICK_ACTION_SIT);                         //click
action to sit
        llSetSitText("PILOT");
        SIT_ROTATION = llEuler2Rot(SIT_ANGLE * DEG_TO_RAD);
        llSitTarget(SIT_POS, SIT_ROTATION);
        llSetCameraEyeOffset(CAM_OFFSET);
        llSetCameraAtOffset(CAM_ANG);

        //SET VEHICLE PARAMETERS
        llSetVehicleType(VEHICLE_TYPE_AIRPLANE);
```

```
        llSetVehicleVectorParam( VEHICLE_LINEAR_FRICTION_TIMESCALE, <200,
20, 20> );

        //uniform angular friction
        llSetVehicleFloatParam( VEHICLE_ANGULAR_FRICTION_TIMESCALE, 2 );

        //linear motor
        llSetVehicleVectorParam( VEHICLE_LINEAR_MOTOR_DIRECTION, <0, 0, 0>
);
        llSetVehicleFloatParam( VEHICLE_LINEAR_MOTOR_TIMESCALE, 2 );
        llSetVehicleFloatParam( VEHICLE_LINEAR_MOTOR_DECAY_TIMESCALE, 120
);

        //angular motor
        llSetVehicleVectorParam( VEHICLE_ANGULAR_MOTOR_DIRECTION, <0, 0, 0>
);
        llSetVehicleFloatParam( VEHICLE_ANGULAR_MOTOR_TIMESCALE, 0 );
        llSetVehicleFloatParam( VEHICLE_ANGULAR_MOTOR_DECAY_TIMESCALE, .4);

        //hover
        llSetVehicleFloatParam( VEHICLE_HOVER_HEIGHT, 2 );
        llSetVehicleFloatParam( VEHICLE_HOVER_EFFICIENCY, 0 );
        llSetVehicleFloatParam( VEHICLE_HOVER_TIMESCALE, 10000 );
        llSetVehicleFloatParam( VEHICLE_BUOYANCY, 1.0 );

        //no linear deflection
        llSetVehicleFloatParam( VEHICLE_LINEAR_DEFLECTION_EFFICIENCY, 0 );
        llSetVehicleFloatParam( VEHICLE_LINEAR_DEFLECTION_TIMESCALE, 5 );

        //no angular deflection
        llSetVehicleFloatParam( VEHICLE_ANGULAR_DEFLECTION_EFFICIENCY, 0);
        llSetVehicleFloatParam( VEHICLE_ANGULAR_DEFLECTION_TIMESCALE, 5);

        //no vertical attractor
        llSetVehicleFloatParam( VEHICLE_VERTICAL_ATTRACTION_EFFICIENCY, 1
);
        llSetVehicleFloatParam( VEHICLE_VERTICAL_ATTRACTION_TIMESCALE, 1 );

        //banking
        llSetVehicleFloatParam( VEHICLE_BANKING_EFFICIENCY, 1);
        llSetVehicleFloatParam( VEHICLE_BANKING_MIX, .5);
        llSetVehicleFloatParam( VEHICLE_BANKING_TIMESCALE, 0.01);

        //default rotation of local frame
        llSetVehicleRotationParam( VEHICLE_REFERENCE_FRAME, <0,0,0,1>);

        //remove these flags
        llRemoveVehicleFlags(VEHICLE_FLAG_NO_DEFLECTION_UP
            | VEHICLE_FLAG_HOVER_WATER_ONLY
            | VEHICLE_FLAG_LIMIT_ROLL_ONLY
            | VEHICLE_FLAG_HOVER_TERRAIN_ONLY
            | VEHICLE_FLAG_HOVER_GLOBAL_HEIGHT
            | VEHICLE_FLAG_HOVER_UP_ONLY
            | VEHICLE_FLAG_LIMIT_MOTOR_UP);

    }

    //LISTEN FOR USER COMMANDS IN CHAT
    listen(integer channel, string name, key id, string message)
    {
        message = llToLower(message);
```

```
        if (message == "help")
        {
            help();
        }

        if (message == "stop")
        {
            brake = TRUE;
            llWhisper(0,"System shut down. You must say START to resume
flight.");
            llSetStatus(STATUS_PHYSICS, FALSE);
        }
        else if (message == "start")
        {
            brake = FALSE;
            llWhisper(0,"System power up. You may begin flight. Say STOP to
end flight.");
            llSetStatus(STATUS_PHYSICS, TRUE);
        }
    }

    //DETECT SITTING/UNSITTING AND GIVE PERMISSIONS
    changed(integer change)
    {
        agent = llAvatarOnSitTarget();
        if(change & CHANGED_LINK)
        {
            if((agent == NULL_KEY) && (sit))
            {
                //
                //  Avatar gets off vehicle
                //
                llSetStatus(STATUS_PHYSICS, FALSE);
                //llStopAnimation("");
                llMessageLinked(LINK_SET, 0, "unseated", "");
                llStopSound();
                llReleaseControls();
                sit = FALSE;
                listenState(FALSE);
                llSay(120, "unseat");
            }
            else if ((agent != NULL_KEY) && (!sit))
            {
                //
                // Avatar gets on vehicle
                //
                pilot = llAvatarOnSitTarget();
                sit = TRUE;
                llRequestPermissions(pilot, PERMISSION_TAKE_CONTROLS |
PERMISSION_TRIGGER_ANIMATION);
                listenState(TRUE);
                llWhisper(0,"Welcome pilot " + llKey2Name(pilot) + ", say
START to power up helicopter and STOP to power down helicopter. Say HELP to
get controls.");
                llMessageLinked(LINK_SET, 0, "seated", "");
            }
        }
    }

    //CHECK PERMISSIONS AND TAKE CONTROLS
    run_time_permissions(integer perm)
```

```
    {
        if (perm & (PERMISSION_TAKE_CONTROLS))
        {
            llTakeControls(CONTROL_UP | CONTROL_DOWN | CONTROL_FWD |
CONTROL_BACK | CONTROL_RIGHT | CONTROL_LEFT | CONTROL_ROT_RIGHT |
CONTROL_ROT_LEFT, TRUE, FALSE);
        }
        if (perm & PERMISSION_TRIGGER_ANIMATION)
        {
            llStopAnimation("sit");
        }
    }

    //FLIGHT CONTROLS
    control(key id, integer level, integer edge)
    {
        vector angular_motor;
        integer motor_changed;

        if ((level & CONTROL_FWD) || (level & CONTROL_BACK))
        {
            if (edge & CONTROL_FWD) xMotor = X_THRUST;
            if (edge & CONTROL_BACK) xMotor = -X_THRUST;
        }
        else
        {
            xMotor = 0;
        }

        if ((level & CONTROL_UP) || (level & CONTROL_DOWN))
        {
            if (level & CONTROL_UP)
            {
                zMotor = Z_THRUST;
            }
            if (level & CONTROL_DOWN)
            {
                zMotor = -Z_THRUST;
            }
        }
        else
        {
            zMotor = 0;
        }

        llSetVehicleVectorParam(VEHICLE_LINEAR_MOTOR_DIRECTION,
<xMotor,0,zMotor>);

        if (level & CONTROL_RIGHT)
        {
            angular_motor.x = 1.5;
            angular_motor.y /= 8;
        }
        if (level & CONTROL_LEFT)
        {
            angular_motor.x = -1.5;
            angular_motor.y /= 8;
        }

        if (level & CONTROL_ROT_RIGHT)
        {
```

```
            angular_motor.x = 1.5;
            angular_motor.y /= 8;
        }

        if (level & CONTROL_ROT_LEFT)
        {
            angular_motor.x = -1.5;
            angular_motor.y /= 8;
        }

        llSetVehicleVectorParam(VEHICLE_ANGULAR_MOTOR_DIRECTION,
angular_motor);
    }
      }
```

The above script was taken from an online source for developing vehicle movement. Many aspects arbitrary aspects for a helicopter were cut, with values changed and parts changed to more suit our helicopter.

The script is set to take chat commands only from the person sitting on the primitive the script is in. The commands "help", "start" and "stop" can be taken. Help will print out a list of the possible commands, start will enable physics for the vehicle and give the pilot permission to take control of the vehicle, and stop will remove permissions to control the vehicle and disable physics.

When you sit on the vehicle, the camera angle is set to a fixed position. After permissions to take controls of the vehicle is given to you (after the vehicle is started), the helicopter will hover the player may use keyboard inputs to move the vehicle up, down, forward, back and to turn the vehicle. These functions are enabled through the angular motor and the linear motor. When the corresponding movement key is pressed, thrust is applied to the specified direction in the appropriate motor.

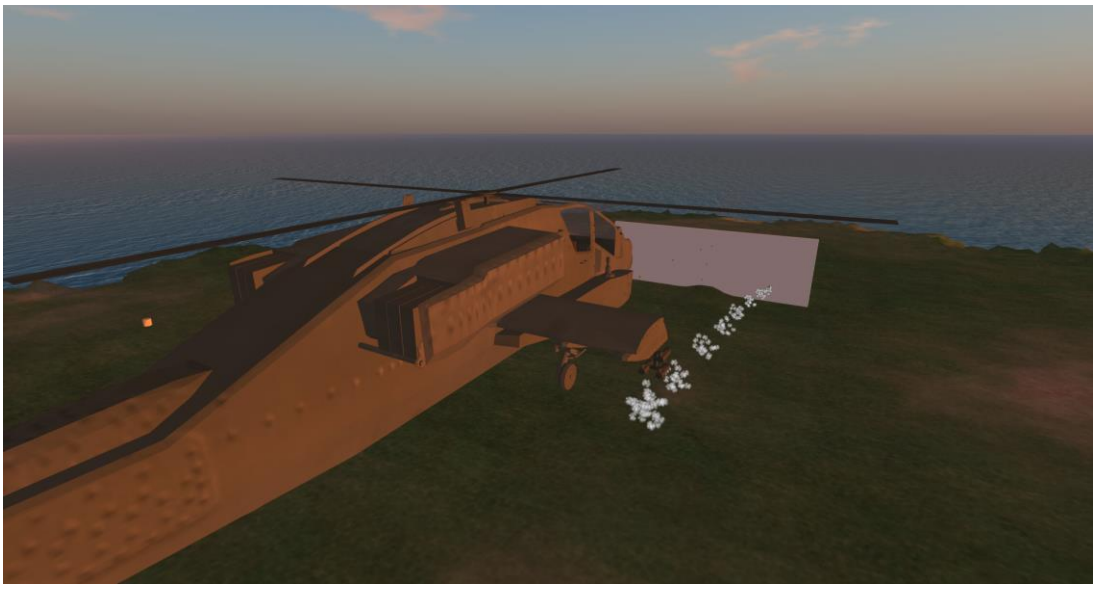
Description of 3D Model Used

We used an Apache 3d model taken from an online source to create the initial body of the apache. Blender was used to make changes to the model, such as removing all parts of the helicopter that had functions, as to create these parts in Open Sim itself. These parts were the main and tail propellers, the missile launchers, the machine gun and the windows.

Blender was then used to give the model thickness, so it would have a tangible surface on both the inside and outside the helicopter. Without thickness, surfaces would appear invisible when looking from the inside of the helicopter to the outside.

Finally Blender was used to reduce the complexity of the model by reducing the number of vertices, before exporting it as a Collada file.

Screenshots



References

LSL Wiki: http://wiki.secondlife.com/wiki/LSL_Portal
Base script for helicopter movement: https://freeslscripts.wordpress.com/about/
Information on the AH-64 Apache: http://en.wikipedia.org/wiki/Boeing_AH-64_Apache
Base Apache model: http://tf3dm.com/3d-model/apache-57610.html
Helicopter texture used:
http://www.cgtextures.com/texview.php?id=50013&PHPSESSID=h5v4a43f85tin9og4hkbiilf02