**STUDENT CONTRIBUTION**

**Brian Dooley – 15123529**

·    Put forward the idea of a tank model and conceptualised functionality of the objects and scripts within it.

·    Designed base body and control panel.

·    Designed the smokescreen object and scripting of its particle effect and sound played.

·    Designed the exhaust pipe and scripting of its particle effect.

·    Implementation of the left and right rotation script in the tank. \*\*\*

**James Gillatt-Haughton – 15157776**

·    Took on the role of project leader.

·    Tested and made sure all scripts for communication from the control panel to each individual script were implemented correctly.

·    Sourced the sounds files used throughout the tank from Battlefield 4.

·    Designed the main barrel of the tank and scripting of its shooting mechanism.

·    Implemented the firing and reloading sounds used in the script.

·    Sourced the texture for the body of the tank.

·    Implementation of the left and right rotation script of the tank. \*\*\*

**Steven Fitzgerald – 15184234**

·    Took charge of the building and completion of the hatch on top of the tank.

·    Wrote the scripts for opening of hatch and associated particle effects.

·    Sourced the sound file played during the opening of the hatch and its implementation in the script.

·    Scripted the fireworks display that is emitted when the hatch is opened

·    Sourced US flag texture for flag underneath hatch.

**Artem Semenov – 15164748**

·    Designed and built the wheels and tracks used on the tank model.

·    Designed and created the texture used for the tracks.

·    Sourced texture for the particle effect seen emitting from the rear of the tank and the texture for the tank wheels.

·    Created the script for the rotation of the wheels and associated particle effect.

·    Implemented sound file into script to simulate engine sounds.

**DESCRIPTION OF FUNCTIONS**

**CONTROL PANEL**

The control panel has 7 buttons that are used to interact with the model of the tank. Each button is clickable and will recede for a second before popping back out. The buttons are different colours as per their functionality. Left and right rotation are cyan and orange respectively. Firing of the cannon is red. The smoke screen button is grey. Rotation of the wheels is green. The opening of the hatch is blue and finally there is a black reset button to reset the tanks position.

**LEFT / RIGHT ARROW**

        When the left arrow is pressed the tank will rotate left for a set amount of time. Similarly, when the right arrow is pressed the tank will turn right for a set amount of time. When either arrow is pressed the wheels will start to rotate and engine sounds will be heard. The position of the tank will only reset upon firing the cannon or pressing the reset button.

**FIRE BUTTON**

        The fire button, when pressed, will cause a projectile to shoot out of the main barrel. Smoke residue will expel from the barrel after the projectile has been fired. A firing sound will be heard initially followed by a brief silence before finally a reloading sound will be heard. There is a reload time of 2 seconds. A new projectile will be available for use after each one is fired.

**SMOKE SCREEN**

        When the smoke screen button is pressed, it will emit a large cloud of smoke concealing the tank to afford its crew time to do evasive manoeuvres. A sound will play when the smoke screen is deployed. The smoke will continue to emit for 5 seconds and by this time the entire tank will be concealed before the smoke particles fade away completely.

**WHEEL ROTATION**

        When the green button is pressed, at the bottom right of the control panel, it will start the rotation of the wheels to simulate forward movement of the tank. Dirt particles will be seen emitting from the rear of both tracks. At the same time, engine sounds will be heard and fumes will be expelled from the exhaust at the rear of the tank. The entire process will continue for 5 seconds.

**HATCH RELEASE / FIREWORKS**

        Pressing the blue button on the control panel will open the hatch on top of the tank. A US flag, attached to the underside of the hatch, will be revealed and a sound clip will be played. A fireworks display will emit from the hatch in the US colours of red white and blue. The whole display will continue until the fireworks display ends.

**RESET BUTTON**

        When the black reset button is pressed the tank will be reset back to its default position if it had been turned left or right.

**Source Code**

*(All the button scripts simply shout on a channel to activate other scripts on the tank.)*

**ButtonLeft-**

default

{

   touch\_start(integer num)

   {

       llShout(-691337, "lturn");

   }

}

**ButtonRight-**

default

{

   touch\_start(integer num)

   {

       llShout(-691337, "lturn");

   }

}

**ButtonFire-**

default

{

   touch\_start(integer num)

   {

       llSetPos(llGetLocalPos()-<0.02,0.0,0.02>);

       llShout(-123, "reset");

       llShout(-6942069, "boom boom");

       llSleep(0.5);

       llSetPos(llGetLocalPos()-<-0.02,0.0,-0.02>);

   }

}

**ButtonSmoke-**

default

{

   touch\_start(integer num)

   {

       llSetPos(llGetLocalPos()-<0.02,0.0,0.02>);

       llShout(-69, "smoke");

       llSleep(0.5);

       llSetPos(llGetLocalPos()-<-0.02,0.0,-0.02>);

   }

}

**ButtonMove-**

default

{

   touch\_start(integer num)

   {

       llSetPos(llGetLocalPos()-<0.02,0.0,0.02>);

       llShout(-420, "vroom vroom");

       llSleep(0.5);

       llSetPos(llGetLocalPos()-<-0.02,0.0,-0.02>);

   }

}

**ButtonHatch-**

default

{

   touch\_start(integer num)

   {

       llSetPos(llGetLocalPos()-<0.02,0.0,0.02>);

       llShout(-1337, "hatch");

       llSleep(0.5);

       llSetPos(llGetLocalPos()-<-0.02,0.0,-0.02>);

   }

}

**ButtonReset-**

default

{

   touch\_start(integer num)

   {

       llSetPos(llGetLocalPos()-<0.02,0.0,0.02>);

       llShout(-123, "reset");

       llSleep(0.5);

       llSetPos(llGetLocalPos()-<-0.02,0.0,-0.02>);

   }

}

*(The hatch script uses stages which are described well bellow)*

**HatchFinal-**

integer iteration = 0;

integer listenHandle;

redToWhite()

{

   llParticleSystem([

                     PSYS\_SRC\_PATTERN, PSYS\_SRC\_PATTERN\_ANGLE\_CONE, //Sets it to an conical pattern exploding up from the hatch

                     PSYS\_SRC\_ANGLE\_BEGIN, 2, PSYS\_SRC\_ANGLE\_END, 225,

                     PSYS\_PART\_START\_COLOR, <1.0, 0, 0>, PSYS\_PART\_END\_COLOR, <1.0,1.0,1.0>, //Colour that the particle will start on and transition to end on.

                     PSYS\_SRC\_MAX\_AGE, 2.0,      PSYS\_PART\_MAX\_AGE, 3,   //Source max age at 0 means particles will continously spawn, particle max age is X seconds

                     PSYS\_SRC\_BURST\_RATE, 2,   PSYS\_SRC\_BURST\_PART\_COUNT, 300, // X New particles created every X seconds.

                     PSYS\_SRC\_ACCEL, <0.0, 0.0,-1.0>,    PSYS\_SRC\_BURST\_RADIUS, 0.04,

                     PSYS\_SRC\_BURST\_SPEED\_MIN, 1.5,      PSYS\_SRC\_BURST\_SPEED\_MAX, 5.0,

                     PSYS\_PART\_START\_SCALE, <0.5,0.5,0.5>, PSYS\_PART\_END\_SCALE, <0.2,0.2,0.2>,

                     PSYS\_PART\_FLAGS, PSYS\_PART\_INTERP\_COLOR\_MASK

                   ]);

}

whiteToBlue()

{

   llParticleSystem([

                     PSYS\_SRC\_PATTERN, PSYS\_SRC\_PATTERN\_ANGLE\_CONE, //Sets it to an conical pattern exploding up from the hatch

                     PSYS\_SRC\_ANGLE\_BEGIN, 2, PSYS\_SRC\_ANGLE\_END,225,

                     PSYS\_PART\_START\_COLOR, <1.0,1.0, 1.0>, PSYS\_PART\_END\_COLOR, <0,0,1.0>, //Colour that the particle will start on and transition to end on.

                     PSYS\_SRC\_MAX\_AGE, 2.0,      PSYS\_PART\_MAX\_AGE, 3,   //Source max age at 0 means particles will continously spawn, particle max age is X seconds

                     PSYS\_SRC\_BURST\_RATE, 2,   PSYS\_SRC\_BURST\_PART\_COUNT, 300, // X New particles created every X seconds.

                     PSYS\_SRC\_ACCEL, <0.0, 0.0,-1.0>,    PSYS\_SRC\_BURST\_RADIUS, 0.04,

                     PSYS\_SRC\_BURST\_SPEED\_MIN, 1.5,      PSYS\_SRC\_BURST\_SPEED\_MAX, 5.0,

                     PSYS\_PART\_START\_SCALE, <0.5,0.5,0.5>, PSYS\_PART\_END\_SCALE, <0.2,0.2,0.2>,

                     PSYS\_PART\_FLAGS, PSYS\_PART\_INTERP\_COLOR\_MASK

                   ]);

}

default //closed state

{

    state\_entry()

   {

       vector v1 = <0.0,0.0,-135.0> \* DEG\_TO\_RAD;

       rotation hatch = llEuler2Rot(v1);

       llSetRot(hatch \* llGetRot());

       llParticleSystem([]);

       iteration = 0;

       listenHandle = llListen(-1337, "", NULL\_KEY, "hatch");

   }

   listen(integer channel, string name, key id, string message)

   {

       state open;

   }

}

   state open

   {

       state\_entry()

       {

           vector v1 = <0.0,0.0,135.0> \* DEG\_TO\_RAD;

           rotation hatch = llEuler2Rot(v1);

           llSetRot(hatch \* llGetRot());

           llPlaySound("Trumpets", 1.0);

           state loop;

       }

   }

   state loop

   {

       state\_entry()

       {

           llSetTimerEvent(1.2);

       }

       timer()

       {

           redToWhite();

           state loop2;

        }

   }

   state loop2

   {

       state\_entry()

       {

           llSetTimerEvent(1.2);

       }

       timer()

       {

           whiteToBlue();

           iteration = iteration + 1;

           if (iteration == 3)

               {

                   state default;

               }

           state loop;

        }

   }

*(Since the rotation of the tank is using LLTargetOmega the movement is actually*

*rendered on the clients side. That means that as far as the server is concerned the tank has not moved. We use this to reset the tanks orientation buy forcing the server to update the client on the tanks real location.)*

**Reset-**

integer listenHandle;

default

{

   state\_entry()

   {

      listenHandle = llListen(-123, "", NULL\_KEY, "reset");

   }

   listen(integer channel, string name, key id, string message)

   {

       llSetText("1", ZERO\_VECTOR, 1.0);

       llSetText("", ZERO\_VECTOR, 0.0);

   }

}

*(This uses LLTargetOmega to rotate the tank body anit-clockwise for 1 second)*

**RotationL-**

integer gap = 1;

integer counter = 0;

integer listenHandle;

remove\_listen\_handle()

{

   llListenRemove(listenHandle);

}

default

{

   state\_entry()

   {

      // llSetTimerEvent(gap);

       listenHandle = llListen(-691337, "", NULL\_KEY, "lturn");

   }

   listen(integer channel, string name, key id, string message)

   {

       llTargetOmega(<0.0,0.0,1.0> \* llGetRot(), .5, 1.0);

       llSleep(1.0);

       llTargetOmega(ZERO\_VECTOR, 0, 0);

   }

}

*(This uses LLTargetOmega to rotate the tank body clockwise for 1 second)*

**RotationR-**

integer gap = 1;

integer counter = 0;

integer listenHandle;

remove\_listen\_handle()

{

   llListenRemove(listenHandle);

}

default

{

   state\_entry()

   {

      // llSetTimerEvent(gap);

       listenHandle = llListen(-691337, "", NULL\_KEY, "rturn");

   }

   listen(integer channel, string name, key id, string message)

   {

       llTargetOmega(<0.0,0.0,-1.0> \* llGetRot(), .5, 1.0);

       llSleep(1.0);

       llTargetOmega(ZERO\_VECTOR, 0, 0);

   }

}

*(Spawns the projectile in relation to the root prim out of the cannon, Plays a firing sound and a reload sound)*

**SpawnShell-**

integer listenHandle;

default

{

   state\_entry()

   {

       listenHandle = llListen(-6942069, "", NULL\_KEY, "boom boom");

   }

   listen(integer channel, string name, key id, string message)

   {

       llRezAtRoot("TankShell", llGetPos()+ <0, 5.8, .8> , <0 ,20, 0>, llGetRot(), 0);

       llPlaySound("CannonFire",1.0);

       llSleep(1.5);

       llPlaySound("ReloadCannon",1.0);

   }

}

*(Makes a smoke particle effect to shroud the tank in smoke, It is emmited from a small box under the tank)*

**CannonSmoke-**

integer listenHandle;

MakeParticles()                //This is the function that actually starts the particle system.

{

   llParticleSystem([                   //KPSv1.0

       PSYS\_PART\_FLAGS , 0 //Comment out any of the following masks to deactivate them

   | PSYS\_PART\_WIND\_MASK             //Particles are moved by wind

   | PSYS\_PART\_INTERP\_COLOR\_MASK       //Colors fade from start to end

   | PSYS\_PART\_INTERP\_SCALE\_MASK       //Scale fades from beginning to end

   | PSYS\_PART\_FOLLOW\_SRC\_MASK         //Particles follow the emitter

   | PSYS\_PART\_FOLLOW\_VELOCITY\_MASK    //Particles are created at the velocity of the emitter

    ,

    PSYS\_SRC\_PATTERN,

    PSYS\_SRC\_PATTERN\_ANGLE\_CONE

   ,PSYS\_PART\_MAX\_AGE,          2.5                //Lifetime, in seconds, that a particle lasts

   ,PSYS\_SRC\_BURST\_RATE,        0.05               //How long, in seconds, between each emission

   ,PSYS\_SRC\_BURST\_PART\_COUNT,  1                  //Number of particles per emission

   ,PSYS\_SRC\_BURST\_RADIUS,      3.0                //Radius of emission

   ,PSYS\_SRC\_BURST\_SPEED\_MIN,   1.5                //Minimum speed of an emitted particle

   ,PSYS\_SRC\_BURST\_SPEED\_MAX,   2.0                //Maximum speed of an emitted particle

   ,PSYS\_SRC\_ACCEL,             <0.0,0.0,0.8>      //Acceleration of particles each second

   ,PSYS\_PART\_START\_COLOR,      <0.8,0.8,0.8>      //Starting RGB color

   ,PSYS\_PART\_END\_COLOR,        <0.2,0.2,0.2>      //Ending RGB color, if INTERP\_COLOR\_MASK is on

   ,PSYS\_PART\_START\_ALPHA,      0.9                //Starting transparency, 1 is opaque, 0 is transparent.

   ,PSYS\_PART\_END\_ALPHA,        0.0                //Ending transparency

   ,PSYS\_PART\_START\_SCALE,      <0.8,0.8,0.0>      //Starting particle size

   ,PSYS\_PART\_END\_SCALE,        <3.3,3.3,0.0>      //Ending particle size, if INTERP\_SCALE\_MASK is on

   ,PSYS\_SRC\_ANGLE\_BEGIN,       -.3                //Inner angle for ANGLE patterns

   ,PSYS\_SRC\_ANGLE\_END,         .3                //Outer angle for ANGLE patterns

   ,PSYS\_SRC\_OMEGA,             <0.0,0.0,0.0>       //Rotation of ANGLE patterns, similar to llTargetOmega()

           ]);

}

default

{

   state\_entry()

   {

       listenHandle = llListen(-6942069, "", NULL\_KEY, "boom boom");

   }

   listen(integer channel, string name, key id, string message)

   {

       MakeParticles();

       llSleep(2.0);

       llParticleSystem([]);

   }

}

*(Plays the animation of the threads turning without particle effects, This is used as the client lags if the effect is emitted from all the wheels)*

**WheelsNoDirt-**

integer gap = 1;

integer counter = 0;

integer listenHandle;

remove\_listen\_handle()

{

   llListenRemove(listenHandle);

}

make\_particles()

{

   llParticleSystem([

       PSYS\_PART\_FLAGS ,

           PSYS\_PART\_WIND\_MASK |          //Particles are moved by wind

           PSYS\_PART\_INTERP\_COLOR\_MASK |  //Colors fade from start to end

           PSYS\_PART\_INTERP\_SCALE\_MASK |  //Scale fades from beginning to end

           PSYS\_PART\_FOLLOW\_SRC\_MASK   |  //Particles follow the emitter

           PSYS\_PART\_FOLLOW\_VELOCITY\_MASK |//Particles are created at the velocity of the emitter

           PSYS\_PART\_EMISSIVE\_MASK,       //Particles are self-lit (glow)

       PSYS\_SRC\_PATTERN,           PSYS\_SRC\_PATTERN\_ANGLE\_CONE,

       PSYS\_SRC\_TEXTURE,           "DirtTexture",        //UUID of the desired particle texture, or inventory name

       PSYS\_SRC\_MAX\_AGE,           15,            //Time, in seconds, for particles to be emitted. 0 = forever

       PSYS\_PART\_MAX\_AGE,          5.0,            //Lifetime, in seconds, that a particle lasts

       PSYS\_SRC\_BURST\_RATE,        0.1,            //How long, in seconds, between each emission

       PSYS\_SRC\_BURST\_PART\_COUNT,  25,              //Number of particles per emission

       PSYS\_SRC\_BURST\_RADIUS,      -10.0,           //Radius of emission

       PSYS\_SRC\_BURST\_SPEED\_MIN,   .4,             //Minimum speed of an emitted particle

       PSYS\_SRC\_BURST\_SPEED\_MAX,   .5,             //Maximum speed of an emitted particle

       PSYS\_SRC\_ACCEL,             <0.0,-1,0>,    //Acceleration of particles each second

       PSYS\_PART\_START\_COLOR,      <1.0,1.0,1.0>,  //Starting RGB color

       PSYS\_PART\_END\_COLOR,        <1.0,1.0,1.0>,  //Ending RGB color, if INTERP\_COLOR\_MASK is on

       PSYS\_PART\_START\_ALPHA,      0.9,            //Starting transparency, 1 is opaque, 0 is transparent.

       PSYS\_PART\_END\_ALPHA,        0.0,            //Ending transparency

       PSYS\_PART\_START\_SCALE,      <.15,.15,.15>,  //Starting particle size

       PSYS\_PART\_END\_SCALE,        <.2,.2,.2>,  //Ending particle size, if INTERP\_SCALE\_MASK is on

       PSYS\_SRC\_ANGLE\_BEGIN,       50 \* DEG\_TO\_RAD, //Inner angle for ANGLE patterns

       PSYS\_SRC\_ANGLE\_END,         50 \* DEG\_TO\_RAD, //Outer angle for ANGLE patterns

       PSYS\_SRC\_OMEGA,             <0,0,180> //Rotation of ANGLE patterns, similar to llTargetOmega()

   ]);

}

default

{

   state\_entry()

   {

       llSetTimerEvent(gap);

       listenHandle = llListen(-420, "", NULL\_KEY, "vroom vroom");

   }

   listen(integer channel, string name, key id, string message)

   {

       llSetTimerEvent(gap);

       llPlaySound("Treads", 1.0);

       counter = 0;

      // make\_particles();

       llTargetOmega(<0.0,0.0,-1.0> \* llGetRot(), 3.0, 1.0);

   }

   timer()

   {

       counter = counter + gap;

       if (counter == 5)  // Rotates for 10 secs, then stops

           {

               llSetTimerEvent(0.0);

               llTargetOmega(ZERO\_VECTOR, 0, 0);

              // llParticleSystem([]);

           }

   }

}

*(This is the wheel script called when the tank rotates)*

**WheelsRot-**

integer gap = 1;

integer counter = 0;

integer listenHandle;

integer listenHandle2;

default

{

   state\_entry()

   {

       llSetTimerEvent(gap);

       listenHandle = llListen(-691337, "", NULL\_KEY, "rturn");

       listenHandle2 = llListen(-691337, "", NULL\_KEY, "lturn");

   }

   listen(integer channel, string name, key id, string message)

   {

       llSetTimerEvent(gap);

       counter = 0;

       llTargetOmega(<0.0,0.0,-1.0> \* llGetRot(), 3.0, 1.0);

   }

   timer()

   {

       counter = counter + gap;

       if (counter == 1)  // Rotates for 1 sec, then stops

           {

               llSetTimerEvent(0.0);

               llTargetOmega(ZERO\_VECTOR, 0, 0);

           }

   }

}

*(Same as above but with the particle effects functioning, this is only placed into the rear wheels)*

**Wheels-**

integer gap = 1;

integer counter = 0;

integer listenHandle;

remove\_listen\_handle()

{

   llListenRemove(listenHandle);

}

make\_particles()

{

   llParticleSystem([

       PSYS\_PART\_FLAGS ,

           PSYS\_PART\_WIND\_MASK |          //Particles are moved by wind

           PSYS\_PART\_INTERP\_COLOR\_MASK |  //Colors fade from start to end

           PSYS\_PART\_INTERP\_SCALE\_MASK |  //Scale fades from beginning to end

           PSYS\_PART\_FOLLOW\_SRC\_MASK   |  //Particles follow the emitter

           PSYS\_PART\_FOLLOW\_VELOCITY\_MASK |//Particles are created at the velocity of the emitter

           PSYS\_PART\_EMISSIVE\_MASK,       //Particles are self-lit (glow)

       PSYS\_SRC\_PATTERN,           PSYS\_SRC\_PATTERN\_ANGLE\_CONE,

       PSYS\_SRC\_TEXTURE,           "DirtTexture",        //UUID of the desired particle texture, or inventory name

       PSYS\_SRC\_MAX\_AGE,           15,            //Time, in seconds, for particles to be emitted. 0 = forever

       PSYS\_PART\_MAX\_AGE,          5.0,            //Lifetime, in seconds, that a particle lasts

       PSYS\_SRC\_BURST\_RATE,        0.1,            //How long, in seconds, between each emission

       PSYS\_SRC\_BURST\_PART\_COUNT,  25,              //Number of particles per emission

       PSYS\_SRC\_BURST\_RADIUS,      -10.0,           //Radius of emission

       PSYS\_SRC\_BURST\_SPEED\_MIN,   .4,             //Minimum speed of an emitted particle

       PSYS\_SRC\_BURST\_SPEED\_MAX,   .5,             //Maximum speed of an emitted particle

       PSYS\_SRC\_ACCEL,             <0.0,-1,0>,    //Acceleration of particles each second

       PSYS\_PART\_START\_COLOR,      <1.0,1.0,1.0>,  //Starting RGB color

       PSYS\_PART\_END\_COLOR,        <1.0,1.0,1.0>,  //Ending RGB color, if INTERP\_COLOR\_MASK is on

       PSYS\_PART\_START\_ALPHA,      0.9,            //Starting transparency, 1 is opaque, 0 is transparent.

       PSYS\_PART\_END\_ALPHA,        0.0,            //Ending transparency

       PSYS\_PART\_START\_SCALE,      <.15,.15,.15>,  //Starting particle size

       PSYS\_PART\_END\_SCALE,        <.2,.2,.2>,  //Ending particle size, if INTERP\_SCALE\_MASK is on

       PSYS\_SRC\_ANGLE\_BEGIN,       50 \* DEG\_TO\_RAD, //Inner angle for ANGLE patterns

       PSYS\_SRC\_ANGLE\_END,         50 \* DEG\_TO\_RAD, //Outer angle for ANGLE patterns

       PSYS\_SRC\_OMEGA,             <0,0,180> //Rotation of ANGLE patterns, similar to llTargetOmega()

   ]);

}

default

{

   state\_entry()

   {

       llSetTimerEvent(gap);

       listenHandle = llListen(-420, "", NULL\_KEY, "vroom vroom");

   }

   listen(integer channel, string name, key id, string message)

   {

       llSetTimerEvent(gap);

       counter = 0;

       make\_particles();

       llTargetOmega(<0.0,0.0,-1.0> \* llGetRot(), 3.0, 1.0);

   }

   timer()

   {

       counter = counter + gap;

       if (counter == 5)  // Rotates for 10 secs, then stops

           {

               llSetTimerEvent(0.0);

               llTargetOmega(ZERO\_VECTOR, 0, 0);

               llParticleSystem([]);

           }

   }

}

*(This is called when the tank rotates it is shortened to one second to match the rotation of pushing the button once)*

**EngineForRotation-**

integer listenHandle;

integer listenHandle2;

default

{

   state\_entry()

   {

       listenHandle = llListen(-691337, "", NULL\_KEY, "rturn");

       listenHandle2 = llListen(-691337, "", NULL\_KEY, "lturn");

   }

   listen(integer channel, string name, key id, string message)

   {

       llPlaySound("Treads", 1.0);

       llSleep(1.0);

       llPlaySound("Treads", 0.0);

   }

}

*(This emits the smoke for when the tank rotates, it is placed in the exhaust)*

**ExhaustSmoke-**

integer listenHandle;

integer listenHandle2;

default

{

   state\_entry()

   {

       listenHandle = llListen(-691337, "", NULL\_KEY, "rturn");

       listenHandle2 = llListen(-691337, "", NULL\_KEY, "lturn");

   }

   listen(integer channel, string name, key id, string message)

   {

       llPlaySound("Treads", 1.0);

       llSleep(1.0);

       llPlaySound("Treads", 0.0);

   }

}

State-transition diagram: Tank Hatch.

Control Panel

Default State

(Closed)

State Open

Touch Listen

Loop 2 - White to Blue Fireworks

Loop 1 – Red to White Fireworks

Moves from Loop 1 to 2, every time it enters

2, iteration++. If Iteration == 3, state Default.

Description of Tank Hatch script.

The function of this script will open the hatch of the tank. When this happens, an American flag pops and a sound file will play. Red, White and Blue firework particles will then erupt from the hatch until the sound file ends and the hatch will then close.

When the script enters state default, this means that it has heard the call from the script in the appropriate button from the control panel. If the hatch is already open, state default rotates the Tank Hatch by -135 degrees along a hinge to close the hatch and will then move to state open. In state open, the hatch moves back 135 degrees on the z axis to open and a sound file will play, it then moves in to state loop.

In state loop a timer is initialised at 1.2 seconds, in this timer the Red to White firework particle system is called with just enough time for 1 burst. Then loop moves into state loop2. In loop2 another timer is set for 1.2 seconds. In this timer the second particle system, White to Blue is called and the global integer, iteration is incremented. The script will move back and forth between the loops until iteration == 3. This is just enough time for the whole sound file to play. When iteration reaches 3 the script moves back into state default where the tank hatch will be closed and iteration is reset to 0. This script will then wait until the button is pressed on the control panel to reinitialise.

Source script for tank hatch. **Please Note:** Comments have been removed from here to save space but remain in the actual script.

integer iteration = 0;

integer listenHandle;

redToWhite(){

llParticleSystem([

PSYS\_SRC\_PATTERN, PSYS\_SRC\_PATTERN\_ANGLE\_CONE,

PSYS\_SRC\_ANGLE\_BEGIN, 2, PSYS\_SRC\_ANGLE\_END, 225,

PSYS\_PART\_START\_COLOR, <1.0, 0, 0>, PSYS\_PART\_END\_COLOR, <1.0,1.0,1.0>,

PSYS\_SRC\_MAX\_AGE, 2.0, PSYS\_PART\_MAX\_AGE, 3,

PSYS\_SRC\_BURST\_RATE, 2, PSYS\_SRC\_BURST\_PART\_COUNT, 300,

PSYS\_SRC\_ACCEL, <0.0, 0.0,-1.0>, PSYS\_SRC\_BURST\_RADIUS, 0.04,

PSYS\_SRC\_BURST\_SPEED\_MIN, 1.5, PSYS\_SRC\_BURST\_SPEED\_MAX, 5.0,

PSYS\_PART\_START\_SCALE, <0.5,0.5,0.5>, PSYS\_PART\_END\_SCALE, <0.2,0.2,0.2>,

PSYS\_PART\_FLAGS, PSYS\_PART\_INTERP\_COLOR\_MASK

]);

}

whiteToBlue(){

llParticleSystem([

PSYS\_SRC\_PATTERN, PSYS\_SRC\_PATTERN\_ANGLE\_CONE,

PSYS\_SRC\_ANGLE\_BEGIN, 2, PSYS\_SRC\_ANGLE\_END,225,

PSYS\_PART\_START\_COLOR, <1.0,1.0, 1.0>, PSYS\_PART\_END\_COLOR, <0,0,1.0>,

PSYS\_SRC\_MAX\_AGE, 2.0, PSYS\_PART\_MAX\_AGE, 3,

PSYS\_SRC\_BURST\_RATE, 2, PSYS\_SRC\_BURST\_PART\_COUNT, 300,

PSYS\_SRC\_ACCEL, <0.0, 0.0,-1.0>, PSYS\_SRC\_BURST\_RADIUS, 0.04,

PSYS\_SRC\_BURST\_SPEED\_MIN, 1.5, PSYS\_SRC\_BURST\_SPEED\_MAX, 5.0,

PSYS\_PART\_START\_SCALE, <0.5,0.5,0.5>, PSYS\_PART\_END\_SCALE, <0.2,0.2,0.2>,

PSYS\_PART\_FLAGS, PSYS\_PART\_INTERP\_COLOR\_MASK

]);

}

default //closed state{

state\_entry() {

vector v1 = <0.0,0.0,-135.0> \* DEG\_TO\_RAD;

rotation hatch = llEuler2Rot(v1);

llSetRot(hatch \* llGetRot());

llParticleSystem([]);

iteration = 0;

listenHandle = llListen(-1337, "", NULL\_KEY, "hatch");

}

listen(integer channel, string name, key id, string message){

state open;

}

}

state open {

state\_entry(){

vector v1 = <0.0,0.0,135.0> \* DEG\_TO\_RAD;

rotation hatch = llEuler2Rot(v1);

llSetRot(hatch \* llGetRot());

llPlaySound("Trumpets", 1.0);

state loop;

}

}

state loop {

state\_entry() {

llSetTimerEvent(1.2);

}

timer(){

redToWhite();

state loop2;

}

}

state loop2 {

state\_entry(){

llSetTimerEvent(1.2);

}

timer() {

whiteToBlue();

iteration = iteration + 1;

if (iteration == 3)

state default;

state loop;

}

}

**Sources for Scripts/Resources**

All scripts not covered in lectures/labs we found on the Second life wiki (<http://wiki.secondlife.com/wiki/Category:LSL_Functions>). Sound files are all sourced from the Battlefield 4 game files. Having extracted them from there and converting them to the correct format on (http://audio.online-convert.com) . The textures are all sourced from google images apart from the Treads which were made by Artem. Particle effects were made through trial and error from particles shown in the labs.