Plugin Tutorial

What are plugins: Plugins extend bzfs (bzflag server). You can make the server do extra things with a plugin. For example, the "____ is on a rampage," etc. messages are created by a plugin.

How to make your own plugins:

First, you need to download the BZFlag source code. Go to http://my.bzflag.org/w/Download and download the most recent version of the source. At the time of writing, http://downloads.sourceforge.net/bzflag/bzflag-2.0.10.zip?download is the most recent version. Once downloaded, go look at the source code for SAMPLE_PLUGIN. /bzflagsource/plugins/SAMPLE_PLUGIN/SAMPLE_PLUGIN.cpp.

```
// SAMPLE_PLUGIN.cpp : Defines the entry point for the DLL application.
//

#include "bzfsAPI.h"
#include "plugin_utils.h"

BZ_GET_PLUGIN_VERSION

BZF_PLUGIN_CALL int bz_Load ( const char* /*commandLine*/ )
{
  bz_debugMessage(4,"SAMPLE_PLUGIN plugin loaded");
  return 0;
}

BZF_PLUGIN_CALL int bz_Unload ( void )
{
  bz_debugMessage(4,"SAMPLE_PLUGIN plugin unloaded");
  return 0;
}

// Local Variables: ***
// mode:C++ ***
// tab-width: 8 ***
// c-basic-offset: 2 ***
// indent-tabs-mode: t ***
// End: ***
// ex: shiftwidth=2 tabstop=8
```

This plugin doesn't actually do anything, it is just a base to start your plugins with. *#include* "*bzfsAPI.h*", and *BZ_GET_PLUGIN_VERSION* are required for your plugins to work. bz_Load and bz_Unload are called when the plugin is loaded or unloaded. The debug messages are displayed by bzfs when in debug mode. If the first parameter for bz_debugMessage is 4, that message will only display if you start bzfs with -dddd, but will not be displayed if you start bzfs with -d, -dd, or -ddd.

Your First Plugin (maybe)

Now let's make a copy of SAMPLE_PLUGIN to work with. You can copy the entire SAMPLE_PLUGIN folder and replace every occurrence of SAMPLE_PLUGIN with the name of your plugin. Or, you can use the newplug.sh script in the plugin folder to make a copy for you. Open up your terminal, and cd to the plugins folder. Now execute "*./newplug.sh my_first_plugin*" . (You will probably need to give the script permission to be executed first, by running "*chmod +x newplug.sh*"). There will now be a folder in plugins called *my_first_plugin*. Go into that folder and open up *my_first_plugin.cpp* in a text editor (TextEdit, Notepad, etc.) It should look the same as SAMPLE_PLUGIN.

Plugins work (mainly) by defining events. When a certain event happens, a function in your plugin is called. A list of available events is located at http://my.bzflag.org/w/Event%28API%29 . Let's start with the player death event. It is called bz_ePlayerDieEvent .

Put this code in your plugin. This plugin will now send a message from the server saying "You got killed" whenever you die, and "You killed someone" when you make a kill.

```
#include "bzfsAPI.h"

BZ_GET_PLUGIN_VERSION

class my_first_plugin_events : public bz_EventHandler
{
  virtual void process ( bz_EventData *eventData )
  {
    if(eventData->eventType==bz_ePlayerDieEvent)
    {
      bz_PlayerDieEventData* data = (bz_PlayerDieEventData*)eventData;
      bz_sendTextMessage(BZ_SERVER,data->playerID,"You got killed!");
      bz_sendTextMessage(BZ_SERVER,data->killerID,"You killed someone!");
    }
  }
};

my_first_plugin_events my_first_plugin_events;

BZF_PLUGIN_CALL int bz_Load ( const char* /*commandLine*/ )
{
  bz_debugMessage(4,"my_first_plugin plugin loaded");
  bz_registerEvent(bz_ePlayerDieEvent,&my_first_plugin_events);
  return 0;
}

BZF_PLUGIN_CALL int bz_Unload ( void )
{
  bz_debugMessage(4,"my_first_plugin plugin unloaded");
  bz_removeEvent(bz_ePlayerDieEvent,&my_first_plugin_events);
  return 0;
}
```

First we define a class, *my_first_plugin_events*. This class will handle the player death event. This class inherits from the *bz_EventHandler* class. We override the *process* function with our own. This function processes the events (only the player death event, currently). Then we check the event data sent to the function to see what kind of event it is. If it is the bz_ePlayerDieEvent, then we execute the code for that event.

```
if(eventData->eventType==bz_ePlayerDieEvent)
{
  bz_PlayerDieEventData* data = (bz_PlayerDieEventData*)eventData;
  bz_sendTextMessage(BZ_SERVER,data->playerID,"You got killed!");
  bz_sendTextMessage(BZ_SERVER,data->killerID,"You killed someone!");
}
```

Inside the event code, we cast the eventData into the type bz_PlayerDieEventData so we can use the data. To see all the data included in the player death event data, go here: http://my.bzflag.org/w/Bz_ePlayerDieEvent , or you can look in the bzfsAPI.h header file (/bzflagsource/include/bzfsAPI.h). Now we use the bz_sendTextMessage() function. This function sends a chat message. There is a list of available functions at http://my.bzflag.org/w/Functions%28API%29 . You can find more info about bz_sendTextMessage() at http://my.bzflag.org/w/Bz_sendTextMessage .
We will be using this version of the function:

```
BZF_API bool bz_sendTextMessage (int from, int to, const char* message);
```

You can use the BZ_SERVER constant for the server. So, we are sending a message *from* the server, *to* the player that got killed (the playerID variable in the event data), and we are *sending the message* "You got killed."

```
bz_sendTextMessage(BZ_SERVER,data->playerID,"You got killed!");
```

The message sent to the killer is similar, except we use killerID instead of playerID.
Then we create an instance of this class. This instance will be used to handle the death event.
Next, in the load and unload functions, we register the player death event. We pass it a reference to the instance of the my_first_plugin_events class. In the unload function, we remove the event. So the *process* function of the *my_first_plugin_events* class will get called every time a player dies.

Now we need to compile the plugin and test it out! Open your terminal, and cd to the folder /bzflagsource/plugins/my_first_plugin/.
Then execute "make install" as root. (Compilation process may be different for you).
Then run "bzfs -loadplugin /usr/local/lib/my_first_plugin.so" . (/usr/local/lib/ is the default directory for the plugins, but it may be different for you. If you use Windows, you will have a .dll instead.)
Open up bzflag with a few bots ("bzflag -solo 5") and join localhost:5154.
Whenever you kill or get killed, you should get sent a message by the server.

Player Records

Player records let you get information about a player from their playerID. If you wanted something like the IP address of a player from their playerID, you'd need to get a player record. All the information available from a player record is listed here: http://my.bzflag.org/w/Bz_BasePlayerRecord .
You can create a player record like this:

```
bz_PlayerRecord   *playerdata;
//make a variable "playerdata" to hold a playerrecord

playerdata  = bz_getPlayerByIndex(data->playerID);
//get the playerrecord data from a playerID with this function

if(playerdata)
{
  /*it is important to do if(playerrecord) before using the player record. For
example, if the playerID you got a player record from doesn't exist, you will get
a segfault when trying to use the player record. Doing if(playerrecord) will
prevent this.*/
  /*INSERT CODE HERE USING THE PLAYER RECORD. example: the player's callsign would
be "playerdata->callsign".*/
}

bz_freePlayerRecord(playerdata);
//This gets rid of the player record when you are done with it.
//important: forgetting to free the player record will result in a memory leak
```

The messages that it sends you when you kill or get killed are kind of boring. Let's make them a little more exciting using player records to display their callsigns... (new code is blue).

```
class my_first_plugin_events : public bz_EventHandler
{
  virtual void process ( bz_EventData *eventData )
  {
    if(eventData->eventType==bz_ePlayerDieEvent)
    {
      bz_PlayerDieEventData* data = (bz_PlayerDieEventData*)eventData;

      bz_PlayerRecord   *playerdata; //create a playerrecord for the dead player.
      playerdata  = bz_getPlayerByIndex(data->playerID); //get data

      bz_PlayerRecord   *killerdata; //another playerrecord for the  killer
      killerdata  = bz_getPlayerByIndex(data->killerID); //get data

      if(killerdata&&playerdata)
      {//make sure that both playerrecords exist before doing anything with them

          std::string killermessage = std::string("You killed ") +
          playerdata->callsign.c_str() +
          std::string(" with ") +
          killerdata->currentFlag.c_str();
          //now we create a message to send to the killer
          std::string playermessage=playerdata->callsign.c_str() +
          std::string(" got killed by ") +
          killerdata->callsign.c_str() +
          std::string(" whose email string is ") +
          killerdata->email.c_str();
          //...and, another message to send to the player

          bz_sendTextMessage(BZ_SERVER,BZ_ALLUSERS,playermessage.c_str());
          bz_sendTextMessage(data->killerID,data->killerID,killermessage.c_str());
          //send the messages we created

        }

        bz_freePlayerRecord(playerdata);//we're done with the playerrecords now,
        bz_freePlayerRecord(killerdata);//so we free them.
      }
    }
};
```

We create a player record for both the player and the killer. Then we use this to generate messages to send to each player.

```
std::string killermessage = std::string("You killed ") +
playerdata->callsign.c_str() +
std::string(" with ") +
killerdata->currentFlag.c_str();
```

First we create a string called killermessage. Then we concatenate several strings into the full message. We put std::string("") around each string and .c_str() after every variable to convert it into type of string we can store in killermessage. std::string("") will convert text into a string, and .c_str() is a function in bz_ApiString (the type of the strings in the player records) into a type that can be stored in a string. There is similar code for creating playermessage.

```
bz_sendTextMessage(BZ_SERVER,BZ_ALLUSERS,playermessage.c_str());
```

Then we send the playermessage to all players. BZ_ALLUSERS is a constant you can use to send a message to all players. Also, c_str is used again to convert playermessage into a string that can be used in the bz_sendTextMessage function.

```
bz_sendTextMessage(data->killerID,data->killerID,killermessage.c_str());
```

You can see here that the message is sent from and to the same player. When you do this, the player will see the message as white text on their screen (with no SERVER: or anything at the beginning).
Then the player records are freed. Also you can put a team type in the "to" parameter, to send a team message. (like, eBlueTeam or eRogueTeam)

Now our plugin will display "You killed (callsign) with (flag)" to the killer, and "(player) was killed by (killer) whose email string is (email)" to all players. Compile it and try it out.

Modification Events

A modification event is one where you can modify the data. The *bz_eGetPlayerSpawnPosEvent* is a modification event. It is called on spawn and lets you change the spawn position of the player.

At the moment, the plugin really doesn't do anything useful. Let's make it a little more useful. We will set up the plugin so that if you are killed by a tank, you will spawn inside a box where you cannot play until after someone captures the flag. So once you get killed, you're "out" until the next game. (note: if you're doing this in a real map, you probably want to make something to do inside this "box" so that players don't get bored and leave.)

First, we'll create a variable, that stores where each player will spawn.

```
bool spawnpos[256]={0};
```

Now we have an array, with one element for each player. If their variable is 0, they will spawn normally, but if their variable is 1, they will spawn in the box. Instead of having 256 variables for each player, we could just create a new variable for each player when they join, and remove it when they leave. But bools are small (1 bit), so it really doesn't matter.

Then we register three more events: *bz_eGetPlayerSpawnPosEvent*, *bz_eCaptureEvent*, and b*z_ePlayerPartEvent.*

In bz_ePlayerDieEvent, we'll set spawnpos[playerID] to 0 or 1 depending on who killed them (a player, or the server -- a world weapon shock wave after cap)

In bz_eGetPlayerSpawnPosEvent (called on spawn), we change their spawn position depending on who they got killed by (spawnpos[playerID]).

In bz_eCaptureEvent (called when a flag is captured), we'll reset all the spawnpos[]'s and fire a giant SW to kill everyone else

And, in bz_ePlayerPartEvent (called when a player leaves), we reset spawnpos[playerID] so that the next player joining with that ID will not have old data from another player.

```
#include "bzfsAPI.h"

BZ_GET_PLUGIN_VERSION

bool spawnpos[256]={0}; //0 means they spawn normally, 1 means they spawn in a box

class my_first_plugin_events : public bz_EventHandler
{
  virtual void process ( bz_EventData *eventData )
  {



    if(eventData->eventType==bz_ePlayerDieEvent)
    {
      bz_PlayerDieEventData* data = (bz_PlayerDieEventData*)eventData;

      if(data->killerTeam==eNoTeam){spawnpos[data->playerID]=0;}
      else{spawnpos[data->playerID]=1;}
      //If the team that killed the player is eNoTeam, then it's the server. This
      //must be the shockwave fired on flag capture. Make them spawn normally.
      //Otherwise, a player must have killed them. Make them spawn in the box.
    }


    else if(eventData->eventType==bz_eCaptureEvent)
    {
      //flag capture event. (no data needed, we won't be needing it)
      for(int i=0; i<256; i++)
      {spawnpos[i]=0;}
      //if a flag gets captured, then we loop through all of spawnpos and reset
      //each element.

      float firepos[3]={0};
      bz_fireWorldWep("SW",10,BZ_SERVER,firepos,0,0,0,0.0f);
      //now we fire a SW. We set the firing position (firepos) to (0,0,0).
      //Then we fire a shockwave at that position, to kill all players on cap.
    }


    else if(eventData->eventType==bz_eGetPlayerSpawnPosEvent)
    {
      bz_GetPlayerSpawnPosEventData* data =
(bz_GetPlayerSpawnPosEventData*)eventData;
      //Player spawn event. This is a modification event, so we can change the
      //spawn position.

      if(spawnpos[data->playerID])
      {
        data->pos[0]=0;data->pos[1]=0;data->pos[2]=50;
      }
      //we check what the spawnpos for that player is. If it is 1 (spawn in the
      //box), then we change the spawn position to (0,0,50) which is where the box
      //will be.
    }


    else if(eventData->eventType==bz_ePlayerPartEvent)
    {
      bz_PlayerJoinPartEventData* data = (bz_PlayerJoinPartEventData*)eventData;
      spawnpos[data->playerID]=0;
      //And, if a player leaves, we reset their spawnpos. That way a new player
      //joining with the same playerID won't get old data from the old player.
    }

  }
};

my_first_plugin_events my_first_plugin_events;

BZF_PLUGIN_CALL int bz_Load ( const char* /*commandLine*/ )
{
  bz_debugMessage(4,"my_first_plugin plugin loaded");
  bz_registerEvent(bz_ePlayerDieEvent,&my_first_plugin_events);
  bz_registerEvent(bz_eCaptureEvent,&my_first_plugin_events);
  bz_registerEvent(bz_eGetPlayerSpawnPosEvent,&my_first_plugin_events);
  bz_registerEvent(bz_ePlayerPartEvent,&my_first_plugin_events);
  //make sure to register all the events
  return 0;
}

BZF_PLUGIN_CALL int bz_Unload ( void )
{
  bz_debugMessage(4,"my_first_plugin plugin unloaded");
  bz_removeEvent(bz_ePlayerDieEvent,&my_first_plugin_events);
  bz_removeEvent(bz_eCaptureEvent,&my_first_plugin_events);
  bz_removeEvent(bz_eGetPlayerSpawnPosEvent,&my_first_plugin_events);
  //...and remove them on unload.
  bz_removeEvent(bz_ePlayerPartEvent,&my_first_plugin_events);
  return 0;
}
```

Now try the plugin out. It is recommended that you test it on a 2 team CTF map, with an inescapable box at position (0,0,50), a very large _shockOutRadius, and spawnzones on top of each team's base.

Loading Parameters

Plugins can be loaded with parameters. If you load your plugin with "-loadplugin /usr/local/lib/my_first_plugin.so,12345" , you'd be passing the parameter string "12345" to the plugin.

```
BZF_PLUGIN_CALL int bz_Load ( const char* /*commandLine*/ )
```

If you uncomment commandLine in this code, you will be able to access the parameter string via commandLine.

Let's change the plugin to have a customizable features. Servers using the plugin can specify whether or not they want shield flags given out on spawn (To give players "2 lives"). Users will be able to load it these three ways:

```
-loadplugin /path/to/my_first_plugin.so       #No parameter. it will default to 0.
-loadplugin /path/to/my_first_plugin.so,0     #Players will not get a SH on spawn.
-loadplugin /path/to/my_first_plugin.so,1     #Players will get a SH on spawn.
```

Now we'll set up a bool to remember this parameter. New code is marked in blue.

```
BZ_GET_PLUGIN_VERSION

bool spawnpos[256]={0};
bool shield=0; //whether or not players get shields on spawn. Defaults to no.
```

Then we register a new event – bz_ePlayerSpawnEvent, which is called on spawn. Then, in the loading function, we check the parameter string. If the first character of the parameter string exists (If we got passed a parameter), then we assign the value to the "shield" variable. (atoi() converts strings to numbers).

```
BZF_PLUGIN_CALL int bz_Load ( const char* commandLine )
{
  bz_debugMessage(4,"my_first_plugin plugin loaded");
  bz_registerEvent(bz_ePlayerDieEvent,&my_first_plugin_events);
  bz_registerEvent(bz_eCaptureEvent,&my_first_plugin_events);
  bz_registerEvent(bz_eGetPlayerSpawnPosEvent,&my_first_plugin_events);
  bz_registerEvent(bz_ePlayerSpawnEvent,&my_first_plugin_events);
  //register the spawn event.
  bz_registerEvent(bz_ePlayerPartEvent,&my_first_plugin_events);
  if(&commandLine[0]) shield=atoi(&commandLine[0]);
  //check what the parameter is – if it is a "1" then we will give SHs on spawn.
  return 0;
}

BZF_PLUGIN_CALL int bz_Unload ( void )
{
  bz_debugMessage(4,"my_first_plugin plugin unloaded");
  bz_removeEvent(bz_ePlayerDieEvent,&my_first_plugin_events);
  bz_removeEvent(bz_eCaptureEvent,&my_first_plugin_events);
  bz_removeEvent(bz_eGetPlayerSpawnPosEvent,&my_first_plugin_events);
  bz_removeEvent(bz_ePlayerSpawnEvent,&my_first_plugin_events);
  //make sure you remember to remove the event on unload – if you don't, you'll
  //get a segfault on unload
  bz_removeEvent(bz_ePlayerPartEvent,&my_first_plugin_events);
  return 0;
}
```

Then, in that event, we give a SH to the player if(shield).

```
else if(eventData->eventType==bz_ePlayerSpawnEvent)
{
  bz_PlayerSpawnEventData* data = (bz_PlayerSpawnEventData*)eventData;
  if(shield)bz_givePlayerFlag(data->playerID,"SH",0);
}
```

Now recompile and test it out. (Make sure the map you are testing on has some shield flags, preferably in an inaccessible location). Try loading it each of these ways and make sure it works as expected.

```
-loadplugin /path/to/my_first_plugin.so       #No parameter. it will default to 0.
-loadplugin /path/to/my_first_plugin.so,0     #Players will not get a SH on spawn.
-loadplugin /path/to/my_first_plugin.so,1     #Players will get a SH on spawn.
```

Slash Commands

The loading parameter is useful, but what if we want the users of this plugin to be able to toggle shields on spawn ingame? Or what if the server owner wants to restart the game? (kill everyone with a SW). To do this you need to register custom slash commands. Let's register /shield and /restartspawn.

First we have to set up a class to handle our custom slash commands. The code is explained in the comments.

```
class my_first_plugin_slashcommands : public bz_CustomSlashCommandHandler
{//create the class, have it inherit from bz_CustomSlashCommandHandler
public:
  virtual bool handle ( int playerID, bzApiString command, bzApiString
/*message*/, bzAPIStringList *params )
  {//the handle function – gets called when someone runs a custom slash command.

    bz_PlayerRecord     *playerdata;
    playerdata = bz_getPlayerByIndex(playerID);
    if(!playerdata||!playerdata->admin){bz_freePlayerRecord(playerdata); return
0;}
    bz_freePlayerRecord(playerdata);
    //create a player record to make sure they are an admin before letting them
    //run this command.

    if(command=="shield")
    {//if they did the /shield command
      if(params->get(0)=="off" || params->get(0)=="0")
      {shield=0;}//if the parameter (/shield off or /shield 0) is "off", turn
                 //shield off.
      else if(params->get(0)=="on" || params->get(0)=="1")
      {shield=1;}//same, except it turns shield on instead of off.
      else { bz_sendTextMessage(BZ_SERVER,playerID,"Usage: /shield <on|off>"); }
      return 1;//if they did not pass the right parameters, send a message telling
               //how to use it. Then return. (return 1 means successful, return 0
               //will show up as "unknown command".)
    }
    else if(command=="restartspawn")
    {// "restartspawn" is basically the same as a flag capture – resets all spawn
     //positions and fires a SW. See the capture event code for more details
      for(int i=0; i<256; i++)
      {
        spawnpos[i]=0;
      }
      float firepos[3]={0};
      bz_fireWorldWep("SW",10,BZ_SERVER,firepos,0,0,0,0.0f);
      return 1;
    }
  }
};

my_first_plugin_slashcommands my_first_plugin_slashcommands;
//then create an instance of this class to handle the custom slash commands.
```

Now we need to actually register the commands. Put

```
bz_registerCustomSlashCommand("shield",&my_first_plugin_slashcommands);
bz_registerCustomSlashCommand("restartspawn",&my_first_plugin_slashcommands);
```

In the load event, and

```
bz_removeCustomSlashCommand("shield");
bz_removeCustomSlashCommand("restartspawn");
```

in the unload event.

Compile the plugin and try it out. Start bzfs with -passwd 1 and then you can type /password 1 ingame to become an admin. Try out both commands, when you are admin and when you are not. Try each parameter for the /shield command and make sure that everything works as expected.

Custom Map Objects

What if the user wants the "out" tanks to spawn somewhere other than (0,0,50), or spawn in different positions for different teams? You could set these up as loading parameters, but there is a better way – custom map objects. They are objects that go in the map file (like a box or pyramid), but they are handled by your plugin.

We'll register the "outspawn" object. Then users of the plugin will be able to put something like:

```
outspawn
position 0 0 50
size 10 10 0
team 1
end
```

in their map, and then all red tanks that are out will spawn within that zone.

Like event handlers and custom slash command handlers, we will need to create a class to handle the custom map object.

```
class my_first_plugin_mapobjects : public bz_CustomMapObjectHandler
{
//Create a class to handle the custom map object(s).
  public:
    virtual bool handle ( bzApiString object, bz_CustomMapObjectInfo *data )
    {
      //The "handle" function of this object is called whenever
      //the server encounters a custom map object while reading the map.
      //So right when the server starts, this function will be called for each
      //custom map object. We will put code here that stores the outspawn object
      //data in variables, so we can use it later.
    }
};

my_first_plugin_mapobjects my_first_plugin_mapobjects;
//Then we create an instance of it to handle the outspawn objects.


/////////////////////////////////////////////////////////////////////////////////


//load event
bz_registerCustomMapObject("outspawn",&my_first_plugin_mapobjects);
//...and, register it in the plugin load event.

//unload event
bz_registerCustomMapObject("outspawn");
//...then unload in the unload event.
```

And we have a class all set up to handle the "outspawn" object. But, now, we need to write the actual code to handle it. First let's make an array to store the data from the outspawn object(s). (New code is marked in blue)

```
BZ_GET_PLUGIN_VERSION

bool spawnpos[256]={0};
bool shield=0;
float outspawns[5][5]={0};
//Now we have an array "outspawns." outspawns[1] refers to the outspawn zone for
//the red team, outspawns[3] refers to the outspawn zone for the blue team, etc.
//The  6 element array inside stores the xmin, xmax, ymin, ymax, and z for the
//outspawn zone. We will leave out a random Z spawn position for simplicity. (they
//will always spawn at the same Z position.
```

Now there's an array to store the outspawn object data. We need to write code that parses the outspawn code and stores it in the array.

```
virtual bool handle ( bzApiString object, bz_CustomMapObjectInfo *data )
     {
       if(object!="OUTSPAWN")return 0;
       //if the custom object is not "outspawn", it fails (returns 0).

       float x=0,y=0,z=0,xsize=0,ysize=0;
       int team=1;
       //init some temporary variables to hold the outspawn data.

       for ( unsigned int i = 0; i < data->data.size(); i++ )
       {
            //This function gets passed a list of strings. Each string is one line
            //from the outspawn object. We'll loop through the lines and parse
            //each one.


            std::string line = data->data.get(i).c_str();
            //get the current line and store it in a string.


            bzAPIStringList *nubs = bz_newStringList();
            nubs->tokenize(line.c_str()," ",0,true);
            //now we split up the line at each space and store it in a list of
            //strings named "nubs".

            if ( nubs->size() > 0)
            {
                  std::string key = bz_toupper(nubs->get(0).c_str());
                  //now we make the first string (the key, like position or size)
                  //all uppercase (case-insensitive) and store it in a string
                  //named "key".

                  if ( key == "POSITION" && nubs->size() > 3)
                  {
                        x = (float)atof(nubs->get(1).c_str());
                        y = (float)atof(nubs->get(2).c_str());
                        z = (float)atof(nubs->get(3).c_str());
                       //if the key is position, get the x y and z values.
                  }
               else if ( key == "SIZE" && nubs->size() > 3)
                  {
                        xsize = (float)atof(nubs->get(1).c_str());
                        ysize = (float)atof(nubs->get(2).c_str());
```

```
                    //same thing for size, except we don't keep the Z value.
                }
                else if ( key == "TEAM" && nubs->size() > 1)
                {
                    //and then we get the team color.
                   team=(int)atoi(nubs->get(1).c_str());
                }

            }
            bz_deleteStringList(nubs);
            //Make sure to delete the list when you're done with it.
        }
        outspawns[team][0]=x-xsize; //convert x and xsize into xmin
        outspawns[team][1]=x+xsize; //convert x and xsize into xmax
        outspawns[team][2]=y-ysize; //same.
        outspawns[team][3]=y+ysize;
        outspawns[team][4]=z;//we'll always spawn at the same z position.
        //Now we convert x/y/xsize/ysize into xmin, xmax etc variables. Then we
        //store it in the correct array element. Then [0] [1]
        //[2] [3] and [4] refer to xmin xmax ymin ymax and z.

        return 1;//Return 1 (successful)
      }
```

Now our plugin reads the outspawn objects and stores them in an array. But the plugin doesn't actually use the data yet... So we'll update the spawn event to use the outspawn zones. (New code is marked in blue)

```
else if(eventData->eventType==bz_eGetPlayerSpawnPosEvent)
    {
      bz_GetPlayerSpawnPosEventData* data =
(bz_GetPlayerSpawnPosEventData*)eventData;
      if(spawnpos[data->playerID])
      {
        int teamnumber=1; //init a variable to figure out what team they are on.
        switch(data->team)
        {
          case eRogueTeam: teamnumber=0; break;
          case eRedTeam: teamnumber=1; break;
          case eGreenTeam: teamnumber=2; break;
          case eBlueTeam: teamnumber=3; break;
          case ePurpleTeam: teamnumber=4; break;
          default: break;
          //check the data, and assign an int value to teamnumber depending
          //on their team color.
        }
        data->pos[0]=(int)((rand()%((int)(outspawns[teamnumber][1]-
outspawns[teamnumber][0])))+outspawns[teamnumber][0]);
        data->pos[1]=(int)((rand()%((int)(outspawns[teamnumber][3]-
outspawns[teamnumber][2])))+outspawns[teamnumber][2]);
        //now we choose random x and y positions depending on the x/y min/max
        values. This is explained in more detail later.
        data->pos[2]=outspawns[teamnumber][4];//put them at the correct Z position
      }
    }
```

Now, when a tank is "out," it will spawn inside the outspawn zone specified in the map file.

This code is a little confusing. It picks a random number between two numbers.

```
data->pos[0]=(int)((rand()%((int)(outspawns[teamnumber][1]-
outspawns[teamnumber][0])))+outspawns[teamnumber][0]);
```

Expanding this code, it looks like this.

```
data->pos[0] =
(int)
(
  (
    rand() %
    (
      (int)
      (
         outspawns[teamnumber][1] – outspawns[teamnumber][0]
      )
    )
  )
  + outspawns[teamnumber][0]
);
```

(Not all the parenthesis and (int)s are required, it just makes it simpler to understand).

The rand() function returns a random number (between 0 and a very large number). You can use the % operator (modulo) to set the range – for example, the number rand() % x is between 0 and x. Now we need the number to be between xmin and xmax. (outspawns[teamnumber][0] and outspawns[teamnumber][1]). So first we get a random number that is between 0 and the difference between xmin and xmax (rand() % xmax-min). The (int) is required because the % operator only works with ints. Since we now have a random number between 0 and the difference, we can add xmin and get a value between xmin and xmax. Similar code is used to get a random Y position.

Now compile the plugin and try it out again. You will need a map file to test it out this time. Here is a simple map file you could test it with:

```
options
-loadplugin /usr/local/lib/my_first_plugin.so -ms 10 -c -mp 0,10,10,0,0,15 -set
_shockOutRadius 700 +f SH{20}
end

base
position 390 390 0
size 10 10 0
color 1
end

base
position -390 -390 0
```

```
size 10 10 0
color 2
end

outspawn
position 0 0 50
size 10 10 0
team 1
end

outspawn
position 300 300 100
size 50 50 0
team 2
end
```

Now when a red tank is out, they will spawn around (0,0,50). When a green tank is out, they'll spawn somewhere around (300,300,100). (In a real map, you'd need to put some inescapable boxes at those positions – right now, you just spawn in the air when you're out.)

Entire Plugin Source

```
// my_first_plugin.cpp : Defines the entry point for the DLL application.
//

#include "bzfsAPI.h"

BZ_GET_PLUGIN_VERSION

bool spawnpos[256]={0};
bool shield=0;
float outspawns[5][5]={{0}};

class my_first_plugin_events : public bz_EventHandler
{
  virtual void process ( bz_EventData *eventData )
  {



    if(eventData->eventType==bz_ePlayerDieEvent)
    {
      bz_PlayerDieEventData* data = (bz_PlayerDieEventData*)eventData;

      if(data->killerTeam==eNoTeam){spawnpos[data->playerID]=0;}
      else{spawnpos[data->playerID]=1;}
    }


    else if(eventData->eventType==bz_eCaptureEvent)
    {
      for(int i=0; i<256; i++)
      {
        spawnpos[i]=0;
      }
      float firepos[3]={0};
      bz_fireWorldWep("SW",10,BZ_SERVER,firepos,0,0,0,0.0f);
    }


    else if(eventData->eventType==bz_eGetPlayerSpawnPosEvent)
    {
      bz_GetPlayerSpawnPosEventData* data =
(bz_GetPlayerSpawnPosEventData*)eventData;
      if(spawnpos[data->playerID])
      {
        int teamnumber=1;
        switch(data->team)
        {
          case eRogueTeam: teamnumber=0; break;
          case eRedTeam: teamnumber=1; break;
          case eGreenTeam: teamnumber=2; break;
          case eBlueTeam: teamnumber=3; break;
          case ePurpleTeam: teamnumber=4; break;
          default: break;
        }
        data->pos[0]=(int)((rand()%((int)(outspawns[teamnumber][1]-
outspawns[teamnumber][0])))+outspawns[teamnumber][0]);
```

```
        data->pos[1]=(int)((rand()%((int)(outspawns[teamnumber][3]-
outspawns[teamnumber][2])))+outspawns[teamnumber][2]);
        data->pos[2]=outspawns[teamnumber][4];
      }
    }


    else if(eventData->eventType==bz_ePlayerPartEvent)
    {
      bz_PlayerJoinPartEventData* data = (bz_PlayerJoinPartEventData*)eventData;
      spawnpos[data->playerID]=0;
    }

    else if(eventData->eventType==bz_ePlayerSpawnEvent)
    {
      bz_PlayerSpawnEventData* data = (bz_PlayerSpawnEventData*)eventData;
      if(shield)bz_givePlayerFlag(data->playerID,"SH",0);
    }


  }
};

my_first_plugin_events my_first_plugin_events;

class my_first_plugin_slashcommands : public bz_CustomSlashCommandHandler
{
public:
  virtual bool handle ( int playerID, bzApiString command, bzApiString
/*message*/, bzAPIStringList *params )
  {

    bz_PlayerRecord     *playerdata;
    playerdata = bz_getPlayerByIndex(playerID);
    if(!playerdata||!playerdata->admin){bz_freePlayerRecord(playerdata); return
0;}
    bz_freePlayerRecord(playerdata);

    if(command=="shield")
    {
      if(params->get(0)=="off" || params->get(0)=="0")
      {shield=0;}
      else if(params->get(0)=="on" || params->get(0)=="1")
      {shield=1;}
      else { bz_sendTextMessage(BZ_SERVER,playerID,"Usage: /shield <on|off>"); }
      return 1;
    }
    else if(command=="restartspawn")
    {
      for(int i=0; i<256; i++)
      {
        spawnpos[i]=0;
      }
      float firepos[3]={0};
      bz_fireWorldWep("SW",10,BZ_SERVER,firepos,0,0,0,0.0f);
      return 1;
    }
    return 0;
  }
};

my_first_plugin_slashcommands my_first_plugin_slashcommands;

class my_first_plugin_mapobjects : public bz_CustomMapObjectHandler
{
public:
      virtual bool handle ( bzApiString object, bz_CustomMapObjectInfo *data )
     {
       if(object!="OUTSPAWN")return 0;
       float x=0,y=0,z=0,xsize=0,ysize=0;
       int team=1;
       for ( unsigned int i = 0; i < data->data.size(); i++ )
        {
            std::string line = data->data.get(i).c_str();

            bzAPIStringList *nubs = bz_newStringList();
            nubs->tokenize(line.c_str()," ",0,true);

            if ( nubs->size() > 0)
            {
                  std::string key = bz_toupper(nubs->get(0).c_str());
                  if ( key == "POSITION" && nubs->size() > 3)
                  {
                        x = (float)atof(nubs->get(1).c_str());
                        y = (float)atof(nubs->get(2).c_str());
                        z = (float)atof(nubs->get(3).c_str());
                  }
               else if ( key == "SIZE" && nubs->size() > 3)
                  {
                        xsize = (float)atof(nubs->get(1).c_str());
                        ysize = (float)atof(nubs->get(2).c_str());
                  }
                  else if ( key == "TEAM" && nubs->size() > 1)
                  {
                        team=(int)atoi(nubs->get(1).c_str());
                  }

            }
            bz_deleteStringList(nubs);
        }
       outspawns[team][0]=x-xsize;
       outspawns[team][1]=x+xsize;
       outspawns[team][2]=y-ysize;
       outspawns[team][3]=y+ysize;
       outspawns[team][4]=z;
       return 1;
     }
};

my_first_plugin_mapobjects my_first_plugin_mapobjects;

BZF_PLUGIN_CALL int bz_Load ( const char* commandLine )
{
  bz_debugMessage(4,"my_first_plugin plugin loaded");
  bz_registerEvent(bz_ePlayerDieEvent,&my_first_plugin_events);
  bz_registerEvent(bz_eCaptureEvent,&my_first_plugin_events);
  bz_registerEvent(bz_eGetPlayerSpawnPosEvent,&my_first_plugin_events);
  bz_registerEvent(bz_ePlayerSpawnEvent,&my_first_plugin_events);
  bz_registerEvent(bz_ePlayerPartEvent,&my_first_plugin_events);
  bz_registerCustomSlashCommand("shield",&my_first_plugin_slashcommands);
  bz_registerCustomSlashCommand("restartspawn",&my_first_plugin_slashcommands);
  bz_registerCustomMapObject("outspawn",&my_first_plugin_mapobjects);
  if(&commandLine[0]) shield=atoi(&commandLine[0]);
  return 0;
}

BZF_PLUGIN_CALL int bz_Unload ( void )
{
  bz_debugMessage(4,"my_first_plugin plugin unloaded");
  bz_removeEvent(bz_ePlayerDieEvent,&my_first_plugin_events);
  bz_removeEvent(bz_eCaptureEvent,&my_first_plugin_events);
  bz_removeEvent(bz_eGetPlayerSpawnPosEvent,&my_first_plugin_events);
  bz_removeEvent(bz_ePlayerSpawnEvent,&my_first_plugin_events);
  bz_removeEvent(bz_ePlayerPartEvent,&my_first_plugin_events);
  bz_removeCustomSlashCommand("shield");
  bz_removeCustomSlashCommand("restartspawn");
  bz_removeCustomMapObject("outspawn");
  return 0;
}

// Local Variables: ***
// mode:C++ ***
// tab-width: 8 ***
// c-basic-offset: 2 ***
// indent-tabs-mode: t ***
// End: ***
// ex: shiftwidth=2 tabstop=8
```

Suggested improvements

These are a few ideas on how to improve the plugin. Try and code them in yourself. =)

Add/remove variables (spawnpos) for each player instead of having tons array elements.

Use a switch instead of else/if for the event handler, etc.

Make it so there can be more than one outspawn object per team, and players would spawn in a random one. Right now, there can only be one, rectangular outspawn zone.

...and anything else you can think up.