

Underlord's Tutorial

Contributed by D1st0rt
Saturday, 20 November 2004

It might be just me, but Underlord's extremely useful MERVBot Tutorial keeps relocating. So, I decided to put a copy up here that won't move around.

MERVBot Tutorial

(Written by Underlord)

Setup a MERVBot bot and project in visual c++

0) Setting up a MERVBot bot

Command.cpp

- 1) Player commands
(!play, !squadA vs squadB)

Spawn.cpp

- 2) Event descriptions
(describe events in spawn.cpp)
- 2) Messaging
(*arena hi", ":player:*scorerereset)
- 3) MervBot Timer
(do this in 10 seconds)(countdown[n])
- 4) Writing Functions
(bool IsInCenter(Player *p))

Useful operations

- 6) Cycling players
- 7) Check if pilot is in safe zone
- 8) Random numbers
- 9) Time without using countdown[n]
- 10) Storing data for pilots
- 11) Output data in messages

12) Input/Output to files

13) Programming commands

14) Useful player data

15) Bot built in functions

Example Code

- a) No antiwarp in center of map
- b) Setting freq size depending on how many pilots in game
- c) Tracking kills and announcing when pilot gets 10 kills in a row without dying
- d) Warp pilot to coord when they are in a certain region
- e) Structures within structures
- f) Tracking flag data
- g) Example way to do simple `!/spam` feature
- h) Example of implementing a simple stack to do "next in line for several 'boxes' at once"
- i) Example of reading any text from a `.txt` and printing it to pilot line by line
- j) Example of printing player stats grid
- k) Example of checking if any pilots are within a region
- l) Example of functions to get a pilot's struct id info from a name or `*player` info
- m) Example of creating a logfile name using date and squad names
- n) Sending messages to playing freqs or public and logging depending on status
- o) Example of reading in all player/freqs to struct data
- p) Example of finding MVP from struct data
- q) Print time stamp of event

- r) Simple way to track player bomb/bullet damage stats
- s) Simple way to print those stats
- t) Make bot spectate specific coordinates

[0] Setting up a MERVBot bot (plugin)

MERVBot download site: <http://catid.sscentral.com>

MERVBot Forum: <http://www.ssforum.net>
(Development - MervBot forum)

1) Download MERVBot Build 37 (DLL 6.5)

Unzip MERVBot.zip into a new folder. (example c:\program files\continuum\mervbot)

Unzip src.zip into "/src" subfolder of that new folder (example c:\program files\continuum\mervbot\src)

2) Download DLL-plugin Tutorial

Unzip Tutorial.zip into "/tutorial" subfolder of that new folder. (example c:\program files\continuum\mervbot\src\tutorial)

(note spawn.h, spawn.cpp, and command.cpp are here)

File descriptions:

spawn.h = declare/initialize globals

command.cpp = code for commands coming into bot (ie !/help, !/play, etc)

spawn.cpp = code for main part of bot

Note: if you only want to execute someone's premade bot (.dll), skip to step 4 (Run your bot .dll), otherwise continue to learn how to make your own bot

3) Microsoft Visual c++

1) Start Visual Studios 6.0

2) Click the Drop Down Menu labbled "File" at the top left of your screen

3) Click "New"

4) On the next screen that comes up, choose from the Project tab, then Win32 Dynamic-Link Library

5) Select the "/src" folder as the base folder (example c:\program files\continuum\mervbot\src)

6) Name your project "mybot". This will make a "mybot" subfolder in your "src" folder. Click OK
(example creates c:\program files\continuum\mervbot\src\mybot)

7) Choose to create an "Empty DLL project"

8) Click "Finish"

9) Click the Drop Down Menu labbled "Project"

10) Click "Add To Project Files"

11) Copy only spawn.h, spawn.cpp, and command.cpp from the "tutorial" folder into the this new folder.

(example from c:\program files\continuum\mervbot\src\tutorial to c:\program files\continuum\mervbot\src\mybot)

12) Click the Drop Down Menu labelled "Build"

13) Click "Build (dll name)" - where the (dll name) is "mybot"

14) Go into your "mybot" folder and look for a folder named "Debug"

(example c:\program files\continuum\mervbot\src\mybot\debug)

15) Your new DLL will be in that folder. (example mybot.dll)

16) Copy mybot.dll to your base folder that has mervbot.exe in it (example c:\program files\continuum\mervbot)

4) Run your bot dll:

To run your bot you need your .dll (mybot.dll), Commands.txt, MERVBot.exe, MERVBot.ini, Operators.txt, Spawns.txt, subspace.bin, and zlib.dll all in one folder (example c:\program files\continuum\mervbot)

Edit spawns.txt (only one line of text in file needed)

```
bot_name : pw_for_bot_name : arena: dll_plugin : optional_staff_password
```

Example: 2v2-Bot-League : botpw: 2v2a: 2v2league : staffpw

- You should create your bot name and pw using your Continuum client as you would create any new name.

- There is no * in front of staffpw.

- Bot needs to be on vip.txt or have moderator+ access to enter a zone.

Edit MERVBot.ini

[Login]

Zone=216.33.98.254:21000 // make that your zone IP:PORT available from zone.dat in Continuum dir

Edit operators.txt

access_level : name :

Example:

4:my_name:

4:another_sysop:

3:other_person:

Run MERVBot.exe

Double click MERVBot.exe, the bot should now enter the zone

Modify your MERVBot code

You can open your MERVBot project by opening "mybot.dsw" file with visual c++

(example c:\program files\continuum\mervbot\src\mybot\mybot.dsw)

Edit the spawn.h, spawn.cpp, and command.cpp to create your bot, then build, copy your updated dll to your MERVBot.exe folder and then execute the bot.

Use the tutorial to get ideas on how to implement certain types of features into the bot.

[1] Player Commands - (command.cpp)

This section describes how to implement player commands into MERVBot

(example: if you want `!/test` to make the bot respond with "hi")

In `command.cpp`

```
void botInfo::gotCommand(Player *p, Command *c) {  
    switch (p->access) {  
        case OP_Player:  
        {  
            if (c->check("test"))  
            {  
                sendPrivate(p,"hi");  
            }  
        }  
    }  
}
```

You just put your player command where 'test' is, then what you want that command to do.

If you want the command to be for moderators you put it in the `OP_Moderator` section instead

```
        case OP_Moderator:  
{ // Moderator-level commands  
}  
  
// Note: Moderators  
can do Player commands, SuperModerator can do Moderator and Player commands,  
etc
```

How to have commands with numerical input - (ie `!test 1`)

```
        if (c->check("test")) { // reads in test #, default to 1 if invalid number input  
  
            int temp = 1;  
  
            if (isNumeric(c->final))  
                temp = atoi(c->final);
```

How to have player name as input - (ie `!rank player`)

```
        if (c->check("rank")) {
```

```
String player_name = c->final;
```

```
if (player_name.IsEmpty()) // default name to self if invalid name
```

```
    player_name = p->name;
```

How to have complicated input - (ie !squads squadA vs squadB _OR_ !squads teamA:squadA:teamB:squadB)

you parse the c->final message
looking for identifiers, then divide the string to get the values

```

else if (c->check("squads"))
{
    strncpy(squadA, "", 20);
    strncpy(squadB, "", 20);

    char s[256];
    strncpy(s,c->final,255);

    int l= strlen(s) - 1;

    for (int i=0; i<l; i++)
    {
        // team1 vs team2

        if ((s[i]==' ') && (s[i+1]=='v') && (s[i+2]=='s') &&
(s[i+3]==' '))
        {
            s[i]=0;
            strncpy(squadA, s,20);
            s[i]=' ';

            strncpy(squadB, &(s[i+4]),20);
            break;
        }
    }

```

```
// freq1:team1:freq2:team2
if (s[i] == ':')
{
    char s2[256];

teamA = atoi(s);

strncpy(s2,&(s[i+1]),255);

for (int j=0; j < strlen(s2); j++)

    if (s2[j] == ':')

    {

        char s3[256];

s2[j]=0;

strncpy(squadA,s2,20);

s2[j]=': ';

strncpy(s3,&(s2[j+1]),255);

for (int k=0;
k<strlen(s3); k++)

if (s3[k]!=':')

{

teamB = atoi(s3);
```


just worry about events that are relevant to the tasks your bot is doing

// Timer

case EVENT_Tick: - called 1x/sec, decrementing each countdown[n] by 1, use for time triggered events

// Arena Events

case EVENT_ArenaEnter: - called when bot enters arena

case EVENT_ArenaSettings: - called when bot gets arena settings

case EVENT_ArenaLeave: - called when bot exits arena

case EVENT_ArenaListEntry: - called when bot gets an entry from the arena list (esc-a)

case EVENT_ArenaListEnd: - called when bot reads last entry of arena list (esc-a)

// Flag Events

case EVENT_FlagGrab: - called when a pilot picks up a loose flag

case EVENT_FlagDrop: - called when a pilot drops a flag (timer runs out)

case EVENT_FlagMove: - called when a flag moves

case EVENT_FlagVictory: - called when a team wins flag game (gets all flags)

case EVENT_FlagReward: - called when theres a periodic flag reward given

// Timed Game

case EVENT_TimedGameOver: - called when a timed game is over (like speed zone game)

// Soccer

case EVENT_SoccerGoal: - called when a soccer goal is scored

case EVENT_BallMove: - called when a soccer ball moves

// Receive file

case EVENT_File: - called when a file is received (*getfile)

// Player events

case EVENT_PlayerEntering: - called when a pilot enters the arena

case EVENT_PlayerMove: - called when a pilot's ship moves

case EVENT_PlayerWeapon: - called when a pilot fires a weapon

case EVENT_WatchDamage: - called when
a pilot takes damage (bot must have /*watchdamage on for them)

case EVENT_PlayerDeath: - called when a pilot dies

case EVENT_PlayerScore: - called when a pilot changes their score (?scorerest)

case EVENT_PlayerPrize: - called when a pilot gets a prize (green)

case EVENT_PlayerShip: - called when a pilot ship changes

case EVENT_PlayerSpec: - called when a pilot spectates

case EVENT_PlayerTeam: - called when a pilot changes freqs

case EVENT_PlayerLeaving: - called when a pilot exits arena (esc-q, ?go <arena>, or lagout)

// Bot events

case EVENT_SelfShipReset: - called when bot's ship gets reset

case EVENT_SelfPrize: - called when bot gets a prize (green)

case EVENT_SelfUFO: - called when bot toggles UFO mode

case EVENT_PositionHook: - called when "Core is requesting the DLL to send a position packet"

// Brick

case EVENT_BrickDropped: - called when a pilot uses a brick

// Objects

case EVENT_ObjectToggled: - called when a .lvz object is toggled on/off

// Turrets

case EVENT_CreateTurret: - called when a pilot attaches to another pilot (f7)

case EVENT_DeleteTurret: - called when a pilot detaches from a turret

// Messages

case EVENT_Chat: - called when bot gets a chat message

case EVENT_LocalCommand: - called when bot gets a player command from arena

case EVENT_LocalHelp: - called when bot gets a player !help command

case EVENT_RemoteCommand: - called when bot gets a command from outside arena

case EVENT_RemoteHelp: - called when bot gets a player !help command from outside arena

case EVENT_Init: - called when DLL is loaded

case EVENT_Term: - called when DLL is unloaded

[3] Messaging - How to use the messaging system

Private message - void sendPrivate(Player *player, char *msg);

Examples:

```
a) sendPrivate(p,"hi");
b) String s="test"; sendPrivate(p,s);
c) String s="test"; s += "ing"; sendPrivate(p,s);

d) char captain1[20]; char captain2[20];
strncpy(captain1,"",20); strncpy(captain2,"",20);
sendPrivate(p,(String) captain1 + " and " + (String) captain2 + " are the captains.");
```

Team message - void sendTeamPrivate(Uint16 team, char *msg);

Examples:

```
a) sendTeamPrivate(8025,"hi spec freq");
b) Uint16 test=0; sendTeamPrivate(test,"hi freq 0");
```

Public message - void sendPublic(char *msg);

Example: sendPublic("*arena " + (String) p->name + " is now a captain");

Chat channel message - void sendChannel(char *msg);

Example: sendChannel("hi chat channel");

Remote private message - void sendRemotePrivate(char *name, char *msg);

Example: sendRemotePrivate("Player01", "hi");

Note: to have bot print several lines of text fast it needs sysop in the arena (sysop in arena bot first spawns to also) otherwise it'll print slow to avoid being kicked for spam

[4] Timer - How to use the timing function

Setup number of timers and initialize:

// spawn.h - specify how many timers

```
class botInfo
```

```
{
```

```
int countdown[10]; // this gives you 10 timers
// spawn.h - initialize timers
```

```
public:
botInfo(CALL_HANDLE given)
{
countdown[0] = 0;
countdown[1] = 60; // 60 seconds
//
// initialize values
//
countdown[9] = 5*60; // 5 minutes
```

Using timer functions:

```
// in spawn.cpp
```

```
case EVENT_Tick:
{
for (int i = 0; i < 10; ++i)
--countdown[i];

if (countdown[1] == 2) // when timer #1 hits two seconds
{
// do stuff here when timer #1 hits 2 seconds
// example: sendPublic("two seconds left, setting timer to 1 minute");
// example: countdown[1] = 60; // change timer #1 value
}
}
```

Notes: - In the "for" loop, you want "i < 10" to be the number of timers from spawn.h

- The timer is in seconds. EVENT_Tick is called once per second.
- The timers decrease by one each second down, not stopping at 0.

- You can change the value of the countdown[n]'s in other parts of the program

(ie. case EVENT_PlayerDeath: { countdown[1] = 45; // set timer #1 to 45 sec

everytime someone dies)

[5] Writing Functions

Write your function in spawn.cpp at top of file in the `//////// DLL "import" //////////` section
`//////// DLL "import" //////////`

```
bool closeto(Player *p, int x, int y, int tolerance) // note uses abs() function declared elsewhere
{
    // return if player
    p is in area of square with center x,y and radius = tolerance

    return (abs((p->tile.x) - x) <
tolerance) && (abs((p->tile.y) - y) < tolerance);

}
```

If
you want your function to have access to data from the spawn.h botInfo section

```
public:
botInfo(CALL_HANDLE given)
{
```

// then you need to make your spawn.cpp function in the format:

```
bool botInfo::closeto(Player *p, int x, int y, int tolerance)
{
    // function here
}
```

Include function in spawn.h at bottom of file with the other functions

```
public:
botInfo(CALL_HANDLE given)
```

```
{
}
```

```
bool closeto(Player *p, int x, int y, int tolerance);
// your new function. note: w/o botInfo keyword and with a ;
```

```
void clear_objects(); // functions already there
```

```
void object_target(Player *p); // functions already there
```

Pass values by reference

```
bool botInfo::closeto(Player *p, int& x, int y, int tolerance) // note the
```

&

```
bool is_close = closeto(p,x,y,tolerance); // if x is changed in closeto() its saved here
```

Pass array as parameter

```
int freqs[5];
```

```
my_function(freqs); // call function - notice freqs and not freqs[5] or freqs[]
```

```
void my_function(int freqs[]) {} // function - notice freqs[] and not freqs[5] or freqs
```

[6] Cycling players- How to search through the players in the arena

```
_listnode <Player> *parse = playerlist->head;
```

```
while (parse)
```

```
{
```

```
Player *p = parse->item;
```

```
// do functionality here
```

```
// Example 1: sendPrivate(p,"*watchdamage"); // turns on all pilot's watchdamage
```

```
// Example 2: if (p->safety != 0)
sendPrivate(p,"*spec"); // spec all pilots in safe zone
```

```
    parse = parse->next
```

```
}
```

[7] Checking if pilot is in a safe zone

```
        if (p->safety != 0) // pilot is in a safe zone
```

```
if (p->safety == 0) // pilot is NOT in a safe zone
```

To detect a pilot entering a safe zone do:

```
case EVENT_PlayerMove:
```

```
{
```

```
    if (p->safety != 0)
```

[8] Random number

Method 1 (completely random)

```
#include "time.h"
```

```
srand(time(NULL));
```

```
rand();
```

```
int temp = (int) (51 * ((float)rand()/RAND_MAX)); // returns a random integer between 0 and 51
```

Method 2 (random pilot in arena)

```
#include <stdlib.h>
```

```
int temp = GetTickCount() % getIngame(); // getIngame() = how many pilots in arena
```

```
_listnode <Player> *parse = playerlist->head;
```

```
while (parse)
```

```

{
    Player *p = parse->item;

    if (p->ship != SHIP_Spectator){ // if not in spec
    if (!(--temp) { // decrement temp, if its 0, make this pilot rabbit
        rabbit = p;
        break;
    }}
    parse = parse->next;
}

```

[9] Tracking time not using countdown[n]

```
#include <stdlib.h>
```

```

int temp = GetTickCount(); // get time stamp
// later in program
temp = (GetTickCount() - temp)/1000; // how many seconds have passed

```

[10] Storing data for pilots

There are several ways to store data for pilots (ie tracking flagtime or kills in a period of time)

1) get/setTag - use if you only want to track data until pilot leaves arena then its erased

built in tags track by an ID that is reset when pilot leaves/enters arena, so loses track of data once they leave arena

2) modified perm get/setTag - use if you want to track all pilots even if they leave (advantage - easier to sort by player)

also can track near unlimited amount of pilots

3) structs - use to track all pilots even if they leave, (advantage - easier to sort by freqs), have to specify bound of players

note: 2 and 3 are similar in effect, mostly the difference is in how you are able to search through data

you need to decide which method of storing data is best for each bot depending on what it does

beware using modified perm get/setTag if bot is in an arena for long periods of time, data is not reset so the linkedlist could get huge

// initialize values in spawn.h at very top

```
1) Built in get/setTag method #define DMG_DEALT 0
```

```
#define DMG_TAKEN 1
```

// in spawn.cpp initialize the values on arena-enter and player-enter

```
case EVENT_ArenaEnter: {
```

```
    _listnode <Player> *parse = playerlist->head;
```

```
    while (parse) // do for all pilots in arena when bot enters
```

```
    {
```

```
        Player *p = parse->item; // get pilot
```

```
        set_tag(p, DMG_DEALT, 0); // initialize to 0
```

```
        set_tag(p, DMG_TAKEN, 0);
```

```
        sendPrivate(p, "*watchdamage");
```

```
// optionally turn on player *watchdamage
```

```
        parse = parse->next; // get next pilot
```

```
    }
```

```
}
```

```
case EVENT_PlayerEntering: {
```

```
    set_tag(p, DMG_DEALT, 0); // initialize to 0
```

```

set_tag(p, DMG_TAKEN, 0);

sendPrivate(p,"*watchdamage");

// then somewhere edit the tag values

case EVENT_WatchDamage: {

    // sets tag for k (shooter) to be old value plus damage dealt currently
    set_tag(k, DMG_BOMB_DEALT, get_tag(k, DMG_BOMB_DEALT) + damage);

// how to retrieve the tag values

// as a command in spawn.h

else if (c->check("showstats")) {

    int temp = get_tag(p, DMG_TOTAL_DEALT);

    String s = "You've done ";
    s += temp;
    s += " damage so far!";

    sendPrivate(p,s);

// kill tags when player leaves arena

case EVENT_PlayerLeaving: {

    killTags(p);

```

2) Modified permanent get/setTag method

tag code, then can use tags exactly as above // same as get/setTag with some modifications to the

```
// spawn.h, add char name[20]; into struct PlayerTag
```

```
struct PlayerTag
```

```
{  
    Player *p;  
    char name[20];  
    int index;  
    int data;  
};
```

```
// spawn.cpp, modifications
```

```
case EVENT_PlayerLeaving:
```

```
{  
    Player *p = (Player*)event.p[0];  
  
    // killTags(p); // remove so tag not deleted on arena exit
```

```
int botInfo::get_tag(Player *p, int index)
```

```
{  
    _listnode <PlayerTag> *parse = taglist.head;  
    PlayerTag *tag;  
  
    while (parse)  
    {  
        tag = parse->item;  
  
        // if (tag->p == p)  
        if (strcmp(tag->name,p->name)==0) // now tracking by player name, not ID  
        if (tag->index == index)  
            return tag->data;  
  
        parse = parse->next;
```

```
}

return 0;

}

void botInfo::set_tag(Player *p, int index, int data)
{
    _listnode <PlayerTag> *parse = taglist.head;
    PlayerTag *tag;

    while (parse)
    {
        tag = parse->item;

        //if (tag->p == p)
        if (strcmp(tag->name,p->name)==0) // now tracking by player name, not ID
            if (tag->index == index)
            {
                tag->data = data;
                return;
            }
        parse = parse->next;
    }

    tag = new PlayerTag;
    // tag->p = p; // not tracking by ID anymore
    strncpy(tag->name, p->name, 20); // tracking by player name
    tag->index = index;
    tag->data = data;
    taglist.append(tag);
}
```

3) Using Struct's

struct name

```
{

// variables

};

// Example: (in spawn.h)

                                class botInfo

{

struct freqdata {

int kills;

int deaths;

};

freqdata freqs[100]; // 100 of those structs

// then access it in spawn.cpp using

freqs[56].kills = 1;
```

[11] Output of data/messages

using normal Strings

```
// does *arena X pilots left in the game.
```

```
String s = "*arena ";
s += temp; // some variable (int)
s += " pilots left in the game.";
sendPublic(s)
```

OR

```
sendPublic("**arena " + (String) temp + " pilots left in the game.");
```

example using sprintf to align/space data

```
// output data
will be in this approximate format (not lined up perfectly because of html)
```

```

//
-----

// Squad: squadname
PTS  FPTS  K  D  DMG DEALT
TAKEN  F  FK  FLT

// -----

//
PlayerA
    10000    500
116 101
9999 99999 10 150 980:55

// PlayerB
    500
200 7 5
    9999 99999 5
    3 0:04

char str[255];

    sendPublic("**arena-----");

    sprintf(str, "**arena
Squad: %-20s PTS  FPTS  K
D  DMG DEALT TAKEN  F  FK  FLT",freqs[freq].freqname);

    sendPublic(str);

    sendPublic("**arena-----");

// assuming existing freqs struct with data
for (pilot=freqs[freq].playercount-1; pilot>=0; pilot--)
{

```

```

// on freq squad so print stats

char outString[255];

    sprintf(outString,
"*arena %-20s %12d %8d %3d %3d %10d %6d %2d %3d %3d:%02d",freqs[freq].pilots[pilot].name,

    freqs[freq].pilots[pilot].points,
freqs[freq].pilots[pilot].flagpoints, freqs[freq].pilots[pilot].kills,

    freqs[freq].pilots[pilot].deaths,
freqs[freq].pilots[pilot].dmgdealt, freqs[freq].pilots[pilot].dmgtaken,freqs[freq].pilots[pilot].flags,

freqs[freq].pilots[pilot].flagkills, freqs[freq].pilots[pilot].flagtime
/60, freqs[freq].pilots[pilot].flagtime %60);

    sendPublic(outString);
}

```

```

// Notes: sprintf
format = sprintf(output char string, spacing, variables)

// Notes: s = chars, d = integer, - = left align, right align default

// Notes: doing
%02d = put 0 in front if not 2 digits, %3d:%02d makes 0:04 format

```

[12] Input/Output to files

Input to file

```

// example reading from duel.ini looking for line that starts with
MaxBoxes= then taking the next char as value to store as

// MAX_BOXES (ie duel.ini = MaxBoxes=5)

#include <fstream>

using namespace std;

ifstream file("duel.ini");

    char line[256];

// read in MaxBoxes=X

while (file.getline(line, 256))

```

```
{
    if (CMPSTART("MaxBoxes=", line))
    {
        MAX_BOXES = atoi(&(line[9]));
        break;
    }
}
```

Output to file

```
// normal char output
```

```
#include <fstream>
```

```
using namespace std;
```

```
ofstream file("duelleaguestat.inc", ios::app); // app = put all data at end of file
```

```
file << squad1<< endl; // squad1 = char[20]
```

```
file << " vs "<< endl;
```

```
file << squad2<< endl; // squad2 = char[20]
```

```
                // how to output String's
to file (key is converting String to (char*) to file write)
```

```
String str = freqs[freq].slotname[slot];
```

```
str += ", Repels: " + (String)(int) t->repel;
```

```
outf << endl;
```

```
outf << (char*) str;
```

```
// date and time stamp
```

```
        #include "time.h"
```

```
char u[100];
```

```
time_t t=time(NULL);
```

```
tm *tmp = localtime(&t);
```

```
strftime(u,99,"%c",tmp);
```

```
sendPublic("Date and time: " + (String) u);
```

Example reading input from file using "GetPrivateProfileString" (from rampage plugin)

format of rampage.ini

7=is on a killing spree! (6:0)

10=is opening a can of whoop-ass! (9:0)

read input

rampageini.h

#pragma once

```
#ifndef RAMPAGEINI_H
```

```
#define RAMPAGEINI_H
```

```
#define NUM_RANKS 10
```

```
#define BUFFER_LEN 256
```

```
struct RampageSettings
```

```
{
```

```
char quotes[NUM_RANKS][BUFFER_LEN];

};

void LoadSettings(RampageSettings &setts);

#endif // RAMPAGEINI_H

rampageini.cpp

#include "rampageini.h"

static char buffer[BUFFER_LEN];

static char path[BUFFER_LEN];

#include "../algorithms.h"

#define WIN32_LEAN_AND_MEAN

#include <windows.h>

char *rank_type[10] = {

    "7",

    "10",

};

void LoadSettings(RampageSettings &setts)

{

    GetCurrentDirectory(BUFFER_LEN - 64, path);

    strcat(path, "\\rampage.ini");

    for (int i = 0; i < NUM_RANKS; ++i)

    {

        GetPrivateProfileString("Comments",

rank_type[i], "-ERROR-", setts.quotes[i], BUFFER_LEN, path);

    }

}
```

[13] Simple programming commands

simple commands

```
declare/initialize variables // example: int
temp; int temp2 = 1; temp2 = 3;
```

```
// bool check = true; check = false;
```

```
if (condition) {}
// example: if (a > b) { a++; }
```

```
if (condition) {} else {}
// example: if (b <= 0) { b--; } else { a++;
}
```

```
while (condition) {}
// example: while (b > 0) { b--; }
```

```
for (initialize; condition; increment){}
// example: for (a=1; a < 10; a++) { a = a + b; }
```

array

single dimension

```
int teams[100]; // create 100 hundred teams 0-99
```

```
teams[50] = 1;
```

multi-dimensional

```
int teams[100][50]; // multidimensional array
```

```
teams[99][49] = 2;
```

variable size

```
String *list = new String[amount+1]; // string array with size amount (variable) + 1;
```

```
list[amount-1] = "hi";
```

struct

struct name

```
{
```

```
// variables
```

```
};
```

// example:

```
struct freqdata {  
int kills; int deaths; };
```

```
freqdata freqs[100]; // 100 of those structs
```

```
freqs[56].kills =  
1; // access struct
```

switch

switch (variable)

```
{  
  
    case n:  
  
        {}  
  
        break;
```

```
    case m:
```

```
    { }
```

```
    break;
```

```
    default:
```

```
    break;
```

```
}
```

```
    // Example:
```

```
switch (p->ship) {
```

```
    case SHIP_Warbird: {
```

```
        sendPrivate(p,"You're in a warbird");
```

```
} break;
```

default:

```
break;
```

[14] Useful Player data

// useful data stored
by MervBot about each player (player.h)

p->name = player name stored as char[20]

p->squad = player squad stored as char[20]

p->ship = ship (0-7) enumerated as SHIP_Warbird, SHIP_Spectator, etc..

p->safety = if ship is in safety zone (boolean)

p->bounty = player bounty

p->energy = player energy (have bot with *energy on to get accurate readings)

p->flagCount = how many flags player is holding

p->team = player frequency

p->(burst, repel, thor, brick, decoy, rocket, portal) = how many items of that type player has

p->(stealth, cloak, xradar, awarp, ufo, flash, safety, shields, supers) = if player has that item on (boolean)

p->score.killPoints = player kill points

p->score.flagPoints = player flag points

p->score.wins = player kills from f2

p->score.losses = player deaths from f2

[15] Bot built in functions

```
// useful MervBot commands to control what the bot is doing
```

```
// player.cpp
```

```
Player::move(Sint32 x, Sint32 y) // example me->move(512,512) - bot moves to coord 512 512
```

```
Player::clone(Player *p) // example me->clone(p)
```

```
// dllcore.cpp (descriptions of functions in dllcore.h)
```

```
BotEvent makeEcho      (char *m);
```

```
BotEvent makeSay      (int t, int s, int i, char *m);
```

```
BotEvent makeShip     (int s);
```

```
BotEvent makeTeam     (int t);
```

```
BotEvent makeGrabFlag (int f);
```

```
BotEvent makeSendPosition (bool reliable);
```

```
BotEvent makeDropFlags ();
```

```
BotEvent makeDeath    (Player *p);
```

```
BotEvent makeAttach   (Player *p);
```

```
BotEvent makeDetach   ();
```

```
BotEvent makeFollowing (bool f);
```

```
BotEvent makeFlying   (bool f);
```

```
BotEvent makeBanner   (BYTE *b);
```

```
BotEvent makeDropBrick ();
```

```
BotEvent makeFireWeapon (void *weapon_info);
```

```
BotEvent makeToggleObjects (Uint16 player, Uint16 *objects, int num_objects);
```

```
BotEvent makeSpawnBot  
(char *name, char *password, char *staff, char *arena);
```

```
BotEvent makeChangeArena    (char *name);
BotEvent makeChangeSettings (_linkedlist <String> *settings);
```

// example how to use them:

```
tell(makeFollowing(false));
```

// look in Commands.txt , command.cpp (core), or /!help to bot to see all bot external commands (example /!go <arena>)

Example Code

a) No antiwarp in center of map (take green away and warn)

```
// using function closeto() and abs()
bool closeto(Player *p, int x, int y, int tolerance) {
    return (abs((p->tile.x) - x) < tolerance) && (abs((p->tile.y) - y) < tolerance); }

inline int abs(int n) {
    if (n < 0)    return -n;
    else        return n; }

radius = 35; // global variable
case EVENT_PlayerMove: {
    Player *p = (Player*)event.p[0];

    // no anti in center
    if ((p->ship != SHIP_Spectator) && (p->awarp)) {
        if (closeto(p, 512, 512, radius)){
            sendPrivate(p, "**prize #-20");

            sendPrivate(p, "**warn Antiwarp is not allowed in center.");
        }
    }
}
```

```
}

```

b) Setting freq size depending on how many pilots in game

```
case EVENT_Tick: {
if (countdown[0] == 0) { // assuming countdown[0] initialized to > 0 in spawn.h, freqchange=0;

```

```
_listnode <Player> *parse = playerlist->head;

```

```
int count = 0;

```

```
while (parse)

```

```
{

```

```
Player *p = parse->item;

```

```
if (p->ship != SHIP_Spectator)

```

```
++count;

```

```
parse = parse->next;

```

```
}

```

```
if ((count > 24) && (freqchange != 4))

```

```
{

```

```
sendPublic("?set team:maxperteam:4");

```

```
String s;

```

```
s = "Max freq size 4 (";

```

```
s += count;

```

```
s += " pilots in game)";

```

```
sendPublic(s);

```

```
freqchange = 4;

```

```
}

```

```
if ((count < 25) && (count > 14) && (freqchange !=
3))

```

```
{

```

```

sendPublic("?set team:maxperteam:3");

    String s;

    s = "Max freq size 3 (";

    s += count;

    s += " pilots in game)";

    sendPublic(s);

    freqchange = 3;

}
countdown[0] = 120;
}

```

c) Tracking kills and announcing when pilot gets 10 kills in a row without dying

```

                case EVENT_PlayerDeath: {
// assuming tags are setup (see storing data section)

set_tag(p,KILLS, 0); // pilot died, reset to 0 kills in a row

set_tag(k, KILLS, get_tag(k, KILLS) + 1); // pilot killed someone, increment kills in a row by 1

if (get_tag(k,KILLS) == 10) sendPublic("*arena (String) k->name + " has gotten 10 kills.");

```

d) Warp pilot to coord when they are in a certain region

```

// using functions closeto() and abs()

bool closeto(Player *p, int x, int y, int tolerance) {

    return (abs((p->tile.x) - x) < tolerance) && (abs((p->tile.y) - y) < tolerance); }

inline int abs(int n) {

    if (n < 0)    return -n;

    else        return n; }

case EVENT_PlayerMove:  {

```

```

    if (closeto(p, 509, 509, 2)) { // if pilot within 2 of map coord 509,509

```

```
sendPrivate(p, "*warpto 509 504"); // warp to coord 509,504
```

e) Structures within structures (spawn.h botinfo)

```
    // Declare in spawn.h  
struct playerstats
```

```
{  
    char name[20];  
  
    int kills;  
    int deaths;  
    Uint16 points;  
    Uint16 flagpoints;  
    int flagtime;  
    int cflagtime;  
    int flags;  
    int flagkills;  
  
    int dmgdealt;  
    int dmgtaken;
```

```
};
```

```
struct freqdata
```

```
{  
    playerstats pilots[100];  
  
    int freqpoints;  
    char freqname[20];  
    int freqflagpoints;  
    Uint16 freqteam;  
    int freqflagtime;
```

```
int flags;

int kills;

int deaths;

int flagkills;

int dmgdealt;

int dmgtaken;

int playercount;

};

freqdata freqs[100];

// Initialize in spawn.cpp
void botInfo::Clear()

{

// initialize/clear struct data

for (int n=99; n>=0; n--)

{

freqs[n].freqteam=-1;

freqs[n].freqpoints=0;

freqs[n].freqflagpoints=0;

freqs[n].playercount=0;

freqs[n].flags=0;

freqs[n].kills=0;

freqs[n].deaths=0;

freqs[n].freqflagtime=0;

freqs[n].flagkills=0;

freqs[n].dmgdealt=0;

freqs[n].dmgtaken=0;

for (int m = 99; m>=0; m--)

{

freqs[n].pilots[m].deaths=0;

freqs[n].pilots[m].kills=0;

freqs[n].pilots[m].points=0;
```

```

    freqs[n].pilots[m].flagpoints=0;
    freqs[n].pilots[m].flagtime=0;
    freqs[n].pilots[m].cflagtime=0;
    freqs[n].pilots[m].flags=0;
    freqs[n].pilots[m].flagkills=0;
    freqs[n].pilots[m].dmgdealt=0;
    freqs[n].pilots[m].dmgtaken=0;
}
}
}
// Access data in spawn.cpp
int freq = p->team;

```

```
freqs[1].pilots[2].kills++;
```

OR

```
freqs[freq].deaths++;
```

f) Tracking flag data

Example GetPilot() function (using structs from example e)

```

bool botInfo::GetPilot(Player *p)
{
    // get a pilots freq/pilot id from struct
    for (freq=freqcount-1; freq>=0; freq--)
        if (p->team == freqs[freq].freqteam)
            for (pilot = freqs[freq].playercount-1; pilot>=0; pilot--)
                if (strcmp(p->name,freqs[freq].pilots[pilot].name)==0)
                    return true;

    return false;
}

```

Example way to track flag data using above struct/functions
case EVENT_FlagGrab:

```
{
```

```

if (GetPilot(p)) // function
{
    freqs[freq].pilots[pilot].flags++;
    freqs[freq].flags++;
}

```

```

if (freqs[freq].pilots[pilot].flags < 2) // didnt have a flag before,
first flag

```

```

    freqs[freq].pilots[pilot].cflagtime = GetTickCount();
// time stamp when picked up flag
}

```

Example way to track flag data using built in get/set tag (from catid flagbot)

```

case EVENT_FlagGrab:

```

```

{

    set_tag(p, TAG_STAT_FS, get_tag(p, TAG_STAT_FS) + 1);

    set_tag(p, TAG_FLAGTIMER, GetTickCount());

}

```

Get current flag times using struct format

```

void botInfo::SetFlagTimes()
{
    // set current flagtime for pilots/freqs
    _listnode <Player> *parse = playerlist->head;

    while (parse)
    {
        Player *p = parse->item;
    }
}

```

```
if (GetPilot(p))
    if (freqs[freq].pilots[pilot].flags > 0)
    {
        if (PilotOnSquad(p))

freqs[freq].freqflagtime += (GetTickCount() - freqs[freq].pilots[pilot].cflagtime)/1000;

freqs[freq].pilots[pilot].flagtime += (GetTickCount() - freqs[freq].pilots[pilot].cflagtime)/1000;

freqs[freq].pilots[pilot].cflagtime = GetTickCount();
    }

    parse = parse->next;
}
}

// side note: case EVENT_FlagDrop: {} gets called anytime theres a teamkill
```

g) Example way to do simple !spam feature (allowed 1x/60s)

declare and initialize variables in spawn.h

```
class botInfo
```

```
{
bool spamready;
```

```
int SPAM_TIME;
```

```
public:
```

```
    botInfo(CALL_HANDLE given)
```

```
    {
```

```
        spamready = true;
```

```
        SPAM_TIME = 60;
```

spawn.cpp - mark as spamready=true when 60 seconds up

```
        case EVENT_Tick:
```

```
    {
```

```
        if (countdown[0] == 1) {
```

```
            spamready = true; // ready to spam again
```

```
        }
```

command.cpp - handle !spam command

```
        case OP_Player:
```

```
    { // Player-level commands
```

```
    else if (c->check("spam"))
```

```
    {
```

```
    // zone announcement "Need pilots to duel in ?go arena -pilotname"
```

```
        if (spamready == true)
```

```
        {
```

```
            String s;
```

```
s += "*zone Need pilots to duel in ?go ";

    s += arena;

    s += " - ";

    s += p->name;

    sendPublic(s);

    spamready=false;

countdown[0] = SPAM_TIME * 60; // next spam time limit
    }
    else if (countdown[0] < 0)
    {

sendPrivate(p,"Spam ability disabled.");
    }
    else
    {
        String s;
        s += SPAM_TIME;

s += " Minute timer between announcements. ";

        s += countdown[0] / 60;

        s += ":";

        if (countdown[0] % 60 < 10)
            s += "0";

        s += countdown[0] % 60;

s += " minutes left before next spam allowed.";

        sendPrivate(p, s);

    }

}
```

h) Example of implementing a simple stack to do "next in line for several 'boxes' at once"

```
//spawn.h declare variables
```

```
class botInfo
```

```
{  
Player *next[99][99];  
int MAX_NEXT;  
int nextcount[99];
```

```
public:
```

```
    botInfo(CALL_HANDLE given)
```

```
{  
    MAX_NEXT = 8;
```

```
// spawn.cpp MoveUp function
```

```
void botInfo::MoveUp(int pos, int box)
```

```
{  
    // moves up the next line for that box and decrement box's nextcount  
    if (nextcount[box] > 0)  
        nextcount[box]--;
```

```
for (pos = pos; pos < MAX_NEXT - 1; pos++)
```

```
{  
    next[box][pos] = next[box][pos + 1];  
}
```

```
next[box][MAX_NEXT] = 0;
```

```
}
```

i) Example of reading any text from a .txt and printing it to pilot line by line

```
                                #include <fstream>

using namespace std;

case OP_Player:
{
    if (c->check("schedule"))
    {
        // read in schedule from schedule.txt
        ifstream file("schedule.txt");
        char line[256];

        while (file.getline(line, 256))
        {
            sendPrivate(p, line);
        }
        file.close();
    }
}
```

j) Example of printing player stats grid

```
freqcount = # of freqs) // spawn.cpp (see "structures within structures" example for variable declarations, varibale
```

```

void botInfo::DisplayPlayers()
{
    // Display Match player/freq stats in this format (not aligned b/c of html but aligned in bot)
    // -----

    // Squad: squad_name_1
    K D TK DMG DEALT TAKEN

    // -----

        // Player_1
    0 0 0    0
    0
    // Player_2
    0 0 0    0
    0
    // TOTAL:
    0 0 0    0
    0

    // -----

    // Squad: squad_name_2
    K D TK DMG DEALT TAKEN

    // -----

        // Player_3
    0 0 0    0
    0
    // Player_4
    0 0 0    0
    0
    // Player_5
    0 0 0    0
    0
    // TOTAL:
    0 0 0    0
    0

    // -----

    for (freq=freqcount-1; freq>=0; freq--)
    {
        char str[255];

        sendFreqs("-----");
    }
}

```

```
    sprintf(str, "Squad: %-20s K D TK DMG DEALT TAKEN", freqs[freq].freqname);
    sendFreqs(str);

    sendFreqs("-----");

    for (pilot=freqs[freq].playercount-1; pilot >= 0; pilot--)
    {
        sprintf(str, "%-20s
        %8d %2d %2d %9d %5d", freqs[freq].pilots[pilot].name, freqs[freq].pilots[pilot].kills,
        freqs[freq].pilots[pilot].deaths, freqs[freq].pilots[pilot].teamkills,
        freqs[freq].pilots[pilot].dmgdealt,
        freqs[freq].pilots[pilot].dmgtaken);
        sendFreqs(str);
    }

    sprintf(str, "TOTAL:
    %2d %2d %2d %9d %5d", freqs[freq].kills, freqs[freq].deaths,
        freqs[freq].teamkills, freqs[freq].dmgdealt, freqs[freq].dmgtaken);
    sendFreqs(str);
}

sendFreqs("-----");
}
```

k) Example of checking if any pilots are within a region

```

// see GetPilotName(name) function in other example, returns *player as TempPlayer from name
// closeto() function from several previous examples

```

```

bool botInfo::FreqInBox()
{
    // return true if teamA has a pilot in the box, otherwise false
    for (int tempplayercount = freqs[0].playercount-1; tempplayercount >= 0; tempplayercount--)
        if (GetPilotName(freqs[0].pilots[tempplayercount].name))
            if (closeto(TempPlayer,
                coordX, coordY, 73) && (TempPlayer->ship != SHIP_Spectator))
                return true;

    return false;
}

```

I) Example of functions to get a pilot's struct id info from a name or *player info

```

// see struct examples for variable info

```

```

// return struct freq/pilot id from *player info
bool botInfo::GetPilot(Player *p)
{
    // return freq, pilot of a player p
    for (freq=freqcount-1; freq>=0; freq--)
        if (p->team == freqs[freq].freqteam)

```

```
    for (pilot = freqs[freq].playercount-1; pilot>=0; pilot--)

    if (strcmp(p->name,freqs[freq].pilots[pilot].name)==0)

        return true;

    return false;

}
```

```
// return *player as TempPlayer info from p->name info
bool botInfo::GetPilotName(char *name)
```

```
{

    // get pilot from a name, return as TempPlayer
    _listnode <Player> *parse = playerlist->head;
```

```
while (parse)
```

```
{
    Player *p = parse->item;
```

```
    // convert both to lowercase to compare
```

```
    char pname[20];
```

```
    strncpy(pname,p->name,20);
```

```
    char nname[20];
```

```
    strncpy(nname,name,20);
```

```
tolower(pname);
tolower(nname);

if (strcmp(pname,nname)==0)
{
    TempPlayer = p;
    return true;
}

    parse = parse->next;
}
return false;
}

// note: better to implement these functions as passing values by reference instead of using global variables
// just easier to not have to be declaring different int freq, int pilot all the time
```

m) Example of creating a logfile name using date and squad names

```
                                // create log file name (squadA and squadB external char[20] variables)

char u[100];
time_t t=time(NULL);
tm *tmp = localtime(&t);
strftime(u,99,"%y",tmp);
logname = "c:\\Program Files\\Continuum\\logs\\";

logname += u;
```

```
logname += "y";
strftime(u,99,"%m",tmp);
logname += u;
logname += "m";
strftime(u,99,"%d",tmp);
logname += u;
logname += "d";
logname += squadA;
logname += " vs ";
logname += squadB;
strftime(u,99,"%l",tmp);
logname += u;
logname += "h";
strftime(u,99,"%M",tmp);
logname += u;
logname += "m";
logname += ".txt";
```

```
// example name created: 03y01m27dBLACKDRaGON vs Integral05h08m.txt
```

```
// format year, month, day, squadA vs squadB, hour, minute
```

n) Example of sending messages to playing freqs or public and logging depending on status

```
// teamA, teamB, logname global variables
void botInfo::sendFreqs(char *msg)
{
    char *mmsg = "*arena";
    String s = msg;
```

```
if (teammgs == false)
{
    s.prepend("*arena ",7);
    sendPublic(s);
}
else
{
    sendTeamPrivate(8025,msg);
    sendTeamPrivate(teamA,msg);
    sendTeamPrivate(teamB,msg);
}
if (gameon == true)
{
    ofstream outf(logname, ios::app);
    outf << msg << endl;
    outf.close();
}
}
```

o) Example of reading in all player/freqs to struct data

// see structures within structures example for freqs[] declaration

```
// to get freqs in a game where there are several freqs
void botInfo::GetFreqs()
{
    // read pilots into freq struct data from ingame and on playing freqs
```

```
_listnode <Player> *parse = playerlist->head;

while (parse)
{
    Player *p = parse->item;

    if (p->ship != SHIP_Spectator)
        if (closeto(p, coordX, coordY, 73))
        {
            // look for freq in struct
            bool foundfreq=false;

            freq=freqcount-1;

            while ((freq>=0) && (foundfreq==false))
            {
                if (p->team == freqs[freq].freqteam)
                {
                    foundfreq=true;

                    strncpy(freqs[freq].pilots[freqs[freq].playercount].name,
                    p->name, 20);

                    freqs[freq].playercount++;

                }

                freq--;
            }
        }
    }
}
```

```
// didnt find freq in struct so add new freq
if (foundfreq == false)
{
    if (manuelsquads == false)
    {

        strncpy(freqs[freqcount].freqname,
p->squad, 20);

    }

    if (freqcount == 0)
    {

        teamA = p->team;

        strncpy(squadA,
p->squad, 20);
    }
    else
    {

        teamB = p->team;

        strncpy(squadB,
p->squad, 20);
    }
}
else
{

    if (p->team == teamA)

        strncpy(freqs[freqcount].freqname,squadA,20);
```

```
else if (p->team == teamB)

    strncpy(freqs[freqcount].freqname,squadB,20);
    }

freqs[freqcount].freqteam = p->team;

strncpy(freqs[freqcount].pilots[0].name, p->name, 20);

    freqs[freqcount].playercount++;
    freqcount++;
    }
}
parse = parse->next;
}
}

// to get freqs in a game where there are only two teams
void botInfo::GetFreqs()
{
    // read pilots into freq struct data from ingame and on playing freqs
    _listnode <Player> *parse = playerlist->head;

    while (parse)
    {
        Player *p = parse->item;
```

```
if ((p->ship != SHIP_Spectator)
&& ((p->team == teamA) || (p->team == teamB)))
{
    // freq 100, team A
    // set freq
    freq = 0;

    if (p->team == teamB)
        freq = 1;

    // number of pilots on freq counted so far, starts 0
    pilot = freqs[freq].playercount;

    // pilot name
    strncpy(freqs[freq].pilots[pilot].name, p->name, 20);
    // time stamp for playing time
    freqs[freq].pilots[pilot].cplaying_time = GetTickCount();

    // slot name
    if (freqs[freq].playercount < NUMBER_PILOTS)
        strncpy(freqs[freq].slotname[pilot], p->name, 20);

    // increment freq player count
    freqs[freq].playercount++;
```

```

// if freq not already have name, give it player squad name
if ((manualsquads == false) && (strlen(p->squad) > 0))
    strncpy(freqs[freq].freqname, p->squad, 20);

// set player ship
freqs[freq].pilots[pilot].ship = p->ship + 1;
}
parse = parse->next;
}
}

```

p) Example of finding MVP from struct data ($2 \cdot \text{kills} - \text{deaths}$ formula)

```

int highest=-20;
int mvp=0;

for (pilot = freqs[mvpteam].playercount-1; pilot >=0; pilot--)
{
    if (((freqs[mvpteam].pilots[pilot].kills
* 2) - freqs[mvpteam].pilots[pilot].deaths) > highest)
    {
        mvp = pilot;
        highest = (freqs[mvpteam].pilots[pilot].kills
* 2) - freqs[mvpteam].pilots[pilot].deaths;
    }
}
}

```

q) Print time stamp of event

```
#include "time.h"
```

```
char u[100];  
time_t t=time(NULL);  
tm *tmp = localtime(&t);  
strftime(u,99,"%c",tmp);  
sendPublic("Current date and time: " + (String) u);
```

r) Simple way to track player bomb/bullet damage stats

```
// spawn.cpp (see data section for how to setup set_tag)
```

```
// see clientprot.h for weapon information
```

```
case EVENT_WatchDamage:
```

```
{  
    if (PLAYING) // if tracking stats  
    {
```

```
if ((wi.type == PROJ_PBomb) && (p->name != k->name))
```

```
{
```

```
    set_tag(k, DMG_BOMB_DEALT, get_tag(k, DMG_BOMB_DEALT)  
+ damage);
```

```
    set_tag(k, DMG_TOTAL_DEALT, get_tag(k, DMG_TOTAL_DEALT)  
+ damage);
```

```
    set_tag(p, DMG_BOMB_TAKEN, get_tag(p, DMG_BOMB_TAKEN)
+ damage);

    set_tag(p, DMG_TOTAL_TAKEN, get_tag(p, DMG_TOTAL_TAKEN)
+ damage);
}

    else if (wi.type == PROJ_BBullet)
    {

        set_tag(k, DMG_BULLET_DEALT, get_tag(k, DMG_BULLET_DEALT)
+ damage);

        set_tag(k, DMG_TOTAL_DEALT, get_tag(k, DMG_TOTAL_DEALT)
+ damage);

        set_tag(p, DMG_BULLET_TAKEN, get_tag(k, DMG_BULLET_TAKEN)
+ damage);

        set_tag(p, DMG_TOTAL_TAKEN, get_tag(k, DMG_TOTAL_TAKEN)
+ damage);
    }
}
```

s) Simple way to print those stats

```
                case OP_Moderator:

{

    if (c->check("showstats"))
    {

        sendPublic("Showing stats:");
    }
}
```

```
_listnode <Player> *parse = playerlist->head;
```

```
while (parse)
{
    Player *p = parse->item;

    if (get_tag(p, DMG_TOTAL_DEALT) > 0)
    {
        char str[256];

        sprintf(str, "%-20s Dmg Dealt: Total
%0004d, Bomb %0004d, Bullet %0004d Dmg TAKEN: Total %0004d, Bomb %0004d,
Bullet %0004d)",

            p->name, get_tag(p,DMG_TOTAL_DEALT),
            get_tag(p,DMG_BOMB_DEALT), get_tag(p,DMG_BULLET_DEALT),

            get_tag(p,DMG_TOTAL_TAKEN), get_tag(p,DMG_BOMB_TAKEN),
            get_tag(p,DMG_BULLET_TAKEN));

        sendPublic(str);
    }

    parse = parse->next;
}
}
```

t) Make bot spectate specific coordinates

```
// make bot spectate the coord 512,600
```

```
// possible use - capturing weapon packets in a specific region
```

```
tell(makeFollowing(false));
```

```
tell(makeFlying(true));
```

```
me->move(512 * 16, 600 * 16);
```

```
tell(makeSendPosition(true));
```

