

How WoG was made: Curses & Blessings

by Jakub Grzana

Introduction:

This document covers general idea of how you can add new “structures” to the game and store data in savefile. As example I will use very simple part of WoG: curses and blessings, because that feature was first to be revamped by me. Without further ado, let’s start.

What are Curses & Blessings?

If you don’t know what those are, I wouldn’t really blame you. Blessings & Curses are effect that can be granted to heroes using ERM commands or by answering Sphinx riddle. You can access them via leftclick on “Curses and Blessings” icon in hero screen.



Curses might have various effects, from forbidding entrance to various object (Cartograph, Arena) or subtracting some number of resource daily, to decreasing scouting range. Blessing are technically same thing, except their effect is positive. All curses used to be hardcoded, until I’ve revamped them to be more usable in modding – you can now setup custom curses with CR receiver, as well as edit existing ones. List of standard blessings and curses can be found in ERM Help.

Most sources mentioned here can be found in [Curse.cpp](#) and [Curse.h](#).

So, how does it work?

At the beginning it's very important to understand how curses are stored in memory. By looking at the picture, you may think they are stored in hero data, right? Wrong! This feature – like many others – is stored in completely different structure, and adding curse to any hero means adding new entry to this structure.

```
extern struct _GodCurse_{
    int Type; // тип проклятия, 0=свободно, "Type of curse, 0=free"
    int HeroInd; // герой-хозяин, "hero-host"
    int StartDay; // день, когда появился у героя, "the day curse was applied"
    int Length; // длительность действия, "Duration of curse"
    int CurseVal; // Значение параметра наказания, "Power of curse"
    int Backup[2];
} CurseInfo[1000];
```

Every entry in this structure store its Type, Hero index, Value, StartDay and Length. If you want to check curses granted to particular hero, you have to search this entire structure for entries where Hero Index match. Because it's static array, there might be 1000 curses granted to all heroes combined. You can try to add more – you will definitely get “internal error”. And that's okay, because in normal circumstances reaching 100 curses in-game would be a challenge.

However structure above has nothing to do with possibilities granted by CR receiver, it wasn't touched, because it's used only to store data about curses granted to heroes. Pictures, descriptions and effects of curses are managed in different, and in my opinion... messy way. Let's start with pics and descs.

```
#define CURSETYPE_NUM (100+1) // Maximum amount of CurseTypes (Type 0 is dummy. so effectively it's indexed from 1)
extern struct _CurseType_ {
    char PicName[64];
    char Desc[256];
} CurseType[CURSETYPE_NUM]; // index in table = Type
```

This structure defines pictures and descriptions for 101 cursetypes, indexed from 0 to 100. Curse type 0 is dummy (in CurseInfo type=0 means “free entry”) so effectively we have 100 cursetypes, indexed from 1 to 100. Note there is no hardcoded connection between description/picture and effect curse grants. Effects are coded totally separately.

Using CR receiver you can change data in CurseType structure. It is stored in savefile, but note that it exist “outside” of particular scenario – those are global variables, thus besides saving and loading feature we need “reset data” function. I will come back to this in a moment.

There is another structure to store data for Curses: DHVC_Table. I will show 3.58f version of that, as it's better as example. New version is same, except it isn't constant

```
// Curses prohibit objects on map table: o_DHVC_Table -> DHVC_Tabke
// Struct: [0] = cursetype, [1] = object type, [2] = object subtype (-1 for any)
// Last entry in structure must be {0,0,0} otherwise segfault
const short int o_DHVC_Table[CURSE_BLOCKS][3]={
    // curse_id, ob_type, ob_subtype
    {22,109,-1}, // Water Wheel
    {23,17,-1}, // Dwelling
    {23,20,-1}, // Dwelling
    {23,216,-1}, // Dwelling
    {23,217,-1}, // Dwelling
    {23,218,-1}, // Dwelling
    {24,4,-1}, // Arena
```

This structure stores information about objects blocked by Curses. There might be up to 200 entries.

There are also structs for Curse/Bless pool. Again, I'm showing 3.58f version as example, new version is the same, except it isn't constant.

```
// Sphinx pools:
// o_AS_CGood -> AS_CGood
// o_AS_CBad -> AS_CBad
// Note: o_AS_CGood[0][0] = number of blessings/curses in pool, need to be increased/decreased when changing!
// same goes for o_AS_CBad
// Struct: [0] = cursetype, [1] = curse minimal value, [2] = curse maximal value

// Sphinx blessings pool
const short int o_AS_CGood[CURSETYPE_NUM][3]={ {12,0,0},
{5,1,3},{7,1,3},{9,50,200},{15,1,6},{16,1,6},
{17,1,2},{18,1,2},{19,1,2},{20,1,2},{21,100,500},
{64,1,4},{65,100,500}};

// Sphinx curses pool
const short int o_AS_CBad[CURSETYPE_NUM][3]={ {48,0,0},
{1,0,0},{2,-1,-1},{3,0,0},{4,100,500},{6,1,3},
{8,0,0},{10,100,300},{22,0,0},{23,0,0},{24,0,0},
{25,0,0},{26,0,0},{27,0,0},{28,0,0},{29,0,0},
{30,0,0},{31,0,0},{32,0,0},{33,0,0},{34,0,0},
{35,0,0},{36,0,0},{37,0,0},{38,0,0},{39,0,0},
{40,0,0},{41,0,0},{42,0,0},{43,0,0},{44,0,0},
{45,0,0},{46,0,0},{47,0,0},{48,0,0},{49,0,0},
{50,0,0},{51,0,0},{52,0,0},{53,0,0},{54,0,0},
{55,0,0},{56,0,0},{57,0,0},{58,0,0},{59,0,0},
{60,0,0},{61,0,0},{62,0,0}};
```

Curse effects:

It's time to discuss hard-coded effects. Let's start with "visit curse" (prohibited entrance)

```
int DoesHeroHasVisitCurse(int hn, int type,int stype)
{
    STARTNA(__LINE__, 0)
    int cn;
    if((type==63)&&(stype>0)) RETURN(-1) // ERM object cannot be prohibited
    for(int i=0;i < CURSE_BLOCKS;i++){
        cn=DHVC_Table[i][0];
        if(cn==0) break;
        if(DHVC_Table[i][1]==type){
            if((DHVC_Table[i][2]==stype)|| (DHVC_Table[i][2]==-1))
            {
                int output = DoesHeroHas(hn,cn);
                if(output != -1) { RETURN(output); }
            }
        }
    }
    RETURN(-1) // не нашли такого типа
}
```

This function has hero index, object type and subtype as parameters. Returns -1 if entrance SHOULDN'T be prohibited, and index of entry in CurseType if it should be prohibited. It's called from hook managing hero-visiting-any-object – this function is far too complex to bring it here.

Now let's look at code responsible for daily curses/blessings. It's basically hardcoded timer. Note the same timer is used to removed curses when they expire.

```
// Daily effect of curse
void DailyCurse(int Owner)
{
    STARTINA(__LINE__, 0)
    int i, day, cr, val, hn, v;
    _Hero_ *hr;
    day=GetCurDate();
    for(i=0; i<CURSENUM; i++){
        cr=CurseInfo[i].Type;
        if(cr==0) continue;
        val=CurseInfo[i].CurseVal;
        hn=CurseInfo[i].HeroInd;
        hr=GetHeroStr(hn);
        if(hr->Owner!=Owner) continue;
        if((CurseInfo[i].StartDay+CurseInfo[i].Length)<day){ // закончилось
            if(_DelCurse(hn, cr, i)){ Error(); RETURNV } // почему-то не удалилось
            continue;
        }
        switch(cr){
            case CURSE_NMONY : AddRes(Owner, 6, -val); break;
            case CURSE_NMANA : v=(int)hr->SpPoints-val; if(v<0) v=0; hr->SpPoints=(Word)v; break;
            case CURSE_PMANA : v=(int)hr->SpPoints+val; if(v>900) v=900; hr->SpPoints=(Word)v; break;
            case CURSE_PEXP : hr->Exp+=val; AddExp(hr); break;
            case CURSE_SLOW : hr->Movement-=val; break;
            case CURSE_SPEED : hr->Movement+=val; break;
            case CURSE_PRI345 : AddRes(Owner, 1, val); AddRes(Owner, 3, val); AddRes(Owner, 4, val); AddRes(Owner, 5, val); break;
            case CURSE_PRO2 : AddRes(Owner, 0, val); AddRes(Owner, 2, val); break;
            case CURSE_PRO : AddRes(Owner, 0, val); break; // дерево
            case CURSE_PR2 : AddRes(Owner, 2, val); break; // руда
            case CURSE_PR5 : AddRes(Owner, 5, val); break; // Самоцветы
            case CURSE_PR1 : AddRes(Owner, 1, val); break; // Ртуть
            case CURSE_PR3 : AddRes(Owner, 3, val); break; // Сера
            case CURSE_PR4 : AddRes(Owner, 4, val); break; // Кристаллы
            case CURSE_PR6 : AddRes(Owner, 6, val); break; // Золото
        }
    }
    // бонусы суместв
    for(i=0; i<HERNUM; i++){
        hr=GetHeroStr(i);
        if(hr->Owner!=Owner) continue;
        if(CheckForCreature(hr, 151)==1){ // алмазный дракон
            AddRes(Owner, 5, 1);
        }
    }
    MagicWonder(hr);
}
RETURNV
}
```

As addition, part of code for scouting/blinding curse. You can find it in [herospec.cpp](#) file.

```
| void __stdcall HeroCheck(int NewX, int NewY, int Level, int Owner, int Radius, int Flag)|
{
    int hn;
    Dword basepo, po;
    _Hero_ *Hp;
    // void pascal (*OrFun)(int, int, int, int, int, int);

    _ECX(basepo);
    _ESI(Hp);
    // OrFun=(void pascal (*)(int, int, int, int, int, int))Callers[4].forig;
    // OrFun(NewX, NewY, Level, Owner, Radius, Flag);
    // ää` "inēīdēīīīīīī" āīdī` āūōīā çāīñū.
    STARTINA(__LINE__, 0)
    //if(WoG){
        if(DoesHeroHas(Hp->Number, CURSE_BLIND)!=-1) RETURNV
        int wl_cscout_index = DoesHeroHas(Hp->Number, CURSE_NSCUT);
        if(wl_cscout_index != -1){
            if(CurseInfo[wl_cscout_index].CurseVal <= 0){ Radius = 2; } // for backcompability
            else { Radius = CurseInfo[wl_cscout_index].CurseVal; } // new
        }
    }
```

You can see which global variables are accessed here, which pointers and which functions are used. It isn't time to discuss them, way too many of them, so let's just go to next station.

Showing curse dialog

```
static char _GC_Length[100][50];
static char *GC_Pics[64]; // Pointers to pictures (paths)
static char *GC_Descr[256]; // Pointers to descriptions
static char *GC_Length[100]; // String containing "length", like this
// 71 "Will last for another %s turns."

static _CurseShow _CurseShow(GC_Pics, GC_Descr, GC_Length);
_ZPrintf_Descr1;
Void BlessesDescr(_MouseStr_ *ms, _Hero_ *hp)
{
    if (ms->Item != 151 && ms->Item != 152)
        return;

    STARTNA(__LINE__, 0)
    char *str;
    if (ms->Item == 151){
        int gr=DoesHeroGot(hp);
        if(gr==0){
            str=ITxt(53,0,&Strings);
        }else{
            Zprintf2(&Descr1,ITxt(54,0,&Strings),(Dword)ITxt(55+gr-GODMONISTRT,0,&Strings),
                    GetGodBonus(hp->Number,0));
            str=Descr1.Str;
        }
    }
    if (ms->Item == 152){
        int cr=DoesHeroHas(hp->Number,0);
        if(cr==1 || ms->Flags & 512){
            str=ITxt(70,0,&Strings);
        }else{
            FillCurseStruct(hp);
            ShowCurse(&CurseShow);
            RETURNV
        }
    }
    Message(str, ((ms->Flags & 512) ? 4 : 1));
    RETURNV
}

struct _CurseShow{
    char **Pics;
    char **Text1;
    char **Text2;
};

static _ZPrintf_FCS_tmp;
static int FillCurseStruct(_Hero_ *hr)
{
    STARTNA(__LINE__, 0)
    int i,j,cnum;
    if (hr==0) RETURN(-1)
    for (i=0,j=0;i<CURSENUM;i++){
        if (CurseInfo[i].Type==0) continue;
        if (CurseInfo[i].HeroInd!=hr->Number) continue;
        cnum=CurseInfo[i].Type;
        if (j==99){ Error(); RETURN(0) }
        GC_Pics[j] = CurseType[cnum].PicName;
        GC_Descr[j] = CurseType[cnum].Descr;
        if (CurseInfo[i].Length==9999) { // 9999
            StrCopy(_GC_Length[j],50,ITxt(72,0,&Strings));
            GC_Length[j]=_GC_Length[j];
        }else{
            Zprintf2(&FCS_tmp,ITxt(71,0,&Strings),
                    (Dword)CurseInfo[i].CurseVal,
                    (Dword)(CurseInfo[i].StarDay*CurseInfo[i].Length-GetCurDate()));
            StrCopy(_GC_Length[j],50,FCS_tmp.Str);
            GC_Length[j]=_GC_Length[j];
        }
        ++j;
    }
    GC_Pics[j]=0;
    RETURN(0)
}
```

This is whole code responsible for creation of Curse dialog – dialog itself is programmed in [service.cpp](#). Red rectangle is function that shows dialog. `_ZPrintf_` is vessel for formatted string – in my opinion, using buffer and `sprintf_s` is better solution, but won't remake old ones unless they are proven to cause errors. `ITxt` is function that allows you to read data from external txt files – localization support.

Saving, loading and restarting curses

```
int SaveCursesData()
{
    STARTNA(__LINE__, 0)
    if (Saver("CRSE",4)) RETURN(1)
    if (Saver(CurseType,sizeof(CurseType))) RETURN(1)
    if (Saver(DHVC_Table,sizeof(DHVC_Table))) RETURN(1)
    if (Saver(CurseInfo,sizeof(CurseInfo))) RETURN(1)
    if (Saver(AS_CBad,sizeof(AS_CBad))) RETURN(1)
    if (Saver(AS_CGood,sizeof(AS_CGood))) RETURN(1)
    RETURN(0)
}

int LoadCursesData(int)
{
    STARTNA(__LINE__, 0)
    char buf[4]; if (Loader(buf,4)) RETURN(1)
    if (buf[0]!='C' || buf[1]!='R' || buf[2]!='S' || buf[3]!='E')
        {MError("LoadCursesData cannot start loading"); RETURN(1) }
    if (Loader(CurseType,sizeof(CurseType))) RETURN(1)
    if (Loader(DHVC_Table,sizeof(DHVC_Table))) RETURN(1)
    if (Loader(CurseInfo,sizeof(CurseInfo))) RETURN(1)
    if (Loader(AS_CBad,sizeof(AS_CBad))) RETURN(1)
    if (Loader(AS_CGood,sizeof(AS_CGood))) RETURN(1)
    RETURN(0)
}
```

To store data in savefile, we use `Saver()` and `Loader()` functions.

```
int Saver(const void *Po,int Len)
int Loader(void *Po,int Len)
```

`Po` is pointer to data you wish to save, and `Len` is length (in bytes) of data to be saved. Both must be used in Heroes 3 saving/loading functions (obviously both are hooked, all you need to do is add entry) Returns 0 if everything is alright, 1 means error.

Note that you CAN'T save/load dynamic arrays the way that you can see above – only static ones. This is because in following example

```
int* dyn_array = new int[size];|
```

`sizeof(dyn_array)` will return size of `int*` (pointer), not size of array.

Also remember that using some kind of "header" (CRSE) for every part of savefile (curses in our example) is preferable – this way, if something goes wrong while saving/loading, you can see which part caused error.


```

void ResetCursesData()
{
    STARTNA(__LINE__, 0)
    for(int i=0;i<CURSENUM;i++){
        CurseInfo[i].Type=0;
        CurseInfo[i].HeroInd=0;
        CurseInfo[i].StartDay=0;
        CurseInfo[i].CurseVal=0;
        CurseInfo[i].Length=0;
    }
    // Reinitialising object blocked by curses
    bool end = false;
    for(int i = 0; i < CURSE_BLOCKS; ++i)
    {
        if(o_DHVC_Table[i][0] == 0) end = true;
        for(int j = 0; j < 3; ++j)
        {
            if(!end) DHVC_Table[i][j] = o_DHVC_Table[i][j];
            else DHVC_Table[i][j] = 0;
        }
    }

    // Reinitialising curse pictures and description
    end = false;
    for(int i = 0; i < CURSETYPE_NUM; ++i)
    {
        if(!strcmp(o_GC_Pics[i],"")) end = true;
        if(!end)
        {
            sprintf_s(CurseType[i].PicName,sizeof(CurseType[i].PicName), "%s", o_GC_Pics[i]);
            if(i>40){
                sprintf_s(CurseType[i].Desc,sizeof(CurseType[i].Desc), "%s", ITxt(90+i,0,&Strings) );
            }else{
                sprintf_s(CurseType[i].Desc,sizeof(CurseType[i].Desc), "%s", ITxt(80+i,0,&Strings) );
            }
        }
        else
        {
            sprintf_s(CurseType[i].PicName,sizeof(CurseType[i].PicName), "");
            sprintf_s(CurseType[i].Desc,sizeof(CurseType[i].Desc), "");
        }
    }

    // Reinitialising sphinx reward/penalty
    bool end_good=false, end_bad=false;
    for(int i = 0; i < CURSETYPE_NUM; ++i)
    {
        if(i > o_AS_CGood[0][0]) end_good=true;
        if(i > o_AS_CBad[0][0]) end_bad=true;
        for(int j = 0; j < 3; ++j)
        {
            if(!end_good) AS_CGood[i][j] = o_AS_CGood[i][j];
            else AS_CGood[i][j] = 0;
            if(!end_bad) AS_CBad[i][j] = o_AS_CBad[i][j];
            else AS_CBad[i][j] = 0;
        }
    }
    RETURNV
}

```

This function is called anytime game want to reset data – that is, every time you load savefile (first Reset is called, then Load) or at start of new map. It restores default values for all cursetypes (defined in [Curse_Hardcoded.h](#)) and clears CurseInfo structure.