[TuT] Trinity BossScripts

18.01.2012, 16:25 #1

Kent_Brockman
Der einzig wahre Booksize

elite*gold: 0
The Black Market: 0/0/0
Registriert seit: Mar 2011
Beiträge: 525
Erhaltene Thanks: 194

[TuT] Trinity BossScripts

Weil nach einem TuT für BossScripte gefragt wurde und ich schon einmal eines geschrieben hatte, mach ichs mal Guttenberg und paste es mal hier ins Forum rein xD

Ich werde in dem Tut das Wort Script bewusst nehmen, auch wenn ich weiß das es prinzipiell eine Struktur/Klasse ist, aber das Wort Script hat sich nun einmal durchgesetzt, weswegen es nur verwirrend wäre alles wieder anders zu benennen.
Auch werde ich es vermeiden Fachbegriffe zu verwenden und alles etwas einsteigerfreundlicher zu beschreiben, wer also hier ein Vortrag für angehende Informatiker erwartet sollte bitte den Thread wieder zu machen.

Und los gehts:
Eigentlich ist so ein Script bauen recht einfach, hier mal ne kleine "Anleitung" wie man sowas anfangen kann...

Als erstes brauchen wir ein Grundgerüst für den Boss.
Das sieht meist so aus:

Code:

```
    CreatureAI* GetAI(Creature* pCreature) const
    {
        return new boss_meinerAI (pCreature);
    }

    struct boss_meinerAI : public ScriptedAI
    {
        boss_meinerAI (Creature *c) : ScriptedAI(c)
        { }

        void Reset()
        { }

        void EnterCombat(Unit* /*who*/)
        { }

        void JustDied(Unit* /*killer*/)
        { }

        void KilledUnit(Unit *)
        { }

        void UpdateAI(const uint32 diff)
        {
            if (!UpdateVictim())
                return;
        }
    };

};

void AddSC_boss_meiner()
{
    new boss_meiner();
}
```

Nun zur Erläuterung der einzelnen Routinen:

- boss_meiner() : CreatureScript("boss_meiner") { }
Hier definiert man den ScriptNamen, in dem Fall heißt er in der ScriptName für die DB "boss_meiner".

- boss_meinerAI (Creature *c) : ScriptedAI(c)
Das ist die "Initialroutine", sie wird am Anfang angefahren und kann verwendet werden um feste Variablen zu setzen (zum Beispiel um den Zugriff auf unser InstanzScript zu bekommen). Auch kann man hier sagen was für eine Art Script es ist.
In dem Fall ist es ne Scripted-AI, wenn der Boss in einer Instanz stehen soll und nicht irgendwo anders in der Welt verwendet wird, können wir auch eine BossAI nehmen, dazu aber vllt. später mehr.

- void Reset()
Diese Routine wird aufgerufen, wenn der Boss resettet wird, also Quasi nachdem ein Wipe war, oder nachdem der Init für das Script kam.

- void EnterCombat(Unit* /*who*/)
Das ist die Routine die aufgerufen wird, wenn er in den Kampf eintritt, dort kann man Texte, Effekte oder was auch immer beim Kampfeintritt gewünscht ist einbauen

- void JustDied(Unit* /*killer*/)
Hat nix mit DiätCola oder so zu tun, sondern ist die Routine, die gefahren wird, wenn der Boss tot ist.

- void UpdateAI(const uint32 diff)
Sehr wichtiges Ding, die Routine wird alle 50ms aufgerufen, dort legt man alle Timerabfragen für Spells, Damage, was noch so passiert rein.

- void KilledUnit(Unit *)
Eine lustige Routine, hier kann man bestimmen was passieren soll, wenn ein Spieler tot ist.

Nun ist dieses Grundgerüst sehr langweilig, weil nix drinnen steht was passieren könnte...
Also füllen wir den Boss mit Leben.
Als erstes würde ich da mal ein paar Texte einfügen.
Einen Text um die Player zu warnen das der Boss nun infight geht, einen um Player verhöhnen, falls einer von ihnen stirbt und einen falls der Boss stirbt.
Dann mal los.

Um einen Boss etwas sagen lassen zu können haben wir zwei Varianten zur Auswahl:
1. Über die Datenbank
2. Über das Script

Natürlich können wir hier auch differenzieren, ob er schreien, sagen oder emotes machen soll (das sind so die wichtigsten, mehr kann man aus der Object.cpp ersehen).

Zum Yell (schreien) nehmen wir me->MonsterYell("TEXT",0,0);
Zum Say (sagen) nehmen wir me->MonsterSay("TEXT",0,0);

Für unseren Boss können wir vorerst ruhig Hardcoding nehmen, wer später lust hat kanns auch in die DB schreiben, der Unterschied ist man muss hier eine ID für seinen Text nutzen, beim Hardcoding passiert das nicht, weil wir den text im Scripts haben.

Okay, ich würde jetzt mal als Eintritt in den Kampf eine einfache Kampfansage bauen.
Das sehe dann so aus:

Code:

```
        void EnterCombat(Unit* /*who*/)
         {
           me->MonsterYell("Ihr wagt es mich zu stoeren.", 0, 0);
           me->MonsterYell("So spuert meinen Zorn", 0, 0);
        }
```

Leider ist das ganze nicht Zeitgesteuert, somit spricht er die beiden Sätz gleich hintereinander.
(Leider könnt ihr keine Umlaute äöü nutzen, weil sonst einfach kein Text angezeigt wird, das ist aber verschmerzlich.
Das ich den Parameter "who" auskommentiert habe ist ganz einfach zu erklären: Ich benutze ihn in dem Fall nicht und ein guter C++ Compiler sollte bei dem Aufruf der Routine damit dann auch keine Parameter übergeben und uns somit die Routine ein bisschen schneller anfahren, das ist zwar nur marginal, dennoch kann es sich bei recht umfangreichen Routinen ziemlich addieren.)

Nun einen netten Text falls ein Player stirbt.

Code:

```
void KilledUnit(Unit *)
{
   me->MonsterYell("Das war zu einfach!", 0, 0);
}
```

Weil es aber schnell langweilig werden kann immer nur das selbe zu hören können wir das ganze ertwas variieren.
Dazu benörigen wir einen Random-generator und eine IF oder einen Switch.

Code:

```
void KilledUnit(Unit *)
{

   switch (urand(1,3))
   {
          case 1:
                 me->MonsterYell("Das war zu einfach!", 0, 0);
          break;
          case 2:
                me->MonsterYell("Ihr seid zu schwach!", 0, 0);
          break;
          case 3:
                me->MonsterYell("Schon muede?", 0, 0);
          break;
   }
}
```

In dem Falle weisen wir urand an eine Zahl im Bereich von 1 bis 3 zu gerieren und mit der Case fragen wir ab welche es war und lassen ihn den Text reden.

Nun weisen wir unseren Boss noch an, das er am Ende auch noch etwas sagt und schon haben wir leben in der Bude.

Code:

```
void JustDied(Unit* /*killer*/)
{
me->MonsterSay("Ich habe versagt...",0,0);
}
```

Das ganze sollte nun so aussehen:

Code:

```
#include "ScriptPCH.h"


class boss_meiner : public CreatureScript
{
public:
    boss_meiner() : CreatureScript("boss_meiner") { }

    CreatureAI* GetAI(Creature* pCreature) const
    {
        return new boss_meinerAI (pCreature);
    }

    struct boss_meinerAI : public ScriptedAI
    {
        boss_meinerAI (Creature *c) : ScriptedAI(c)
        { }

        void Reset()
        { }

        void EnterCombat(Unit* /*who*/)
        {
           me->MonsterYell("Ihr wagt es mich zu stoeren.", 0, 0);
           me->MonsterYell("So spuert meinen Zorn", 0, 0);
        }

        void JustDied(Unit* /*killer*/)
        {
          me->MonsterSay("Ich habe versagt...",0,0);
        }

        void KilledUnit(Unit *)
        {
             switch (urand(1,3))
             {
```

So ohne Spells ists aber recht langweilig im Kampfgeschehen, also lassen wir ihn mal was nettes Casten.
Für den eintritt in den Kampf nehmen wir mal nen Effekt-Spell, der noch keine Auswirkungen hat, aber schön aussieht.
Ab dann werden wir 2 Spells im Timer casten.

Code:

```
#include "ScriptPCH.h"

enum Spells
{
    SPELL_BALNAZZAR            = 17288,
    SPELL_STOMP                   = 66330,
    SPELL_EXPLOSION            = 69839
};

class boss_meiner : public CreatureScript
{
public:
    boss_meiner() : CreatureScript("boss_meiner") { }

    CreatureAI* GetAI(Creature* pCreature) const
    {
        return new boss_meinerAI (pCreature);
    }

    struct boss_meinerAI : public ScriptedAI
    {
        boss_meinerAI (Creature *c) : ScriptedAI(c)
        { }

        uint32 t_stomp;
        uint32 t_explosion;

        void Reset()
        {
            t_stomp = 3000;
            t_explosion = 10000;
        }

        void EnterCombat(Unit* /*who*/)
        {
           me->MonsterYell("Ihr wagt es mich zu stoeren.", 0, 0);
```

Wie man sehen kann hab ich nun die UpdateAI etwas ausgebaut und oben einen Enum(erator) erstellt.
Der Enum ist dazu da um unsere SPELL_FOO eine SpellID zuzuweisen.
Warum man das macht ist schnell geklärt:
Es ist übersichtlicher, als wenn man beim Cast immer die ID angeben muss, ausserdem kann man schnell mal den Spell wechseln.

Nun zur UpdateAI().
Im wesentlichen hab ich unsere zwei Spells mit Timern eingebaut.
Die TimerWerte werden in unserer Reset-Routine eingestellt, im UpdateAI heruntergezählt und anschliessend wieder neu gesetzt.
Setzt man sie nicht neu, macht er den Spell nur einmal.

Code:

```
            if (t_stomp <= diff)
            {
                DoCast(me->getVictim(), SPELL_STOMP);
                t_stomp = urand(3000,6000);
            } else t_stomp -= diff;
```

Stampfen (stomp) wird hier alle 3 Sekunden ausgeführt, dannach wird es alle 3 bis 6 Sekunden gefahren (urand(3000,6000);).

Bei Explosion nehmen wir hier einen festen Wert von immer 10 Sekunden.

Code:

```
if (t_explosion <= diff)
            {
                DoCast(me->getVictim(), SPELL_EXPLOSION);
                t_explosion = 10000;
            } else t_explosion -= diff;
```

Um nochmal das Runterzählen aufzuschlüsseln:
if (t_explosion <= diff) //Wenn t_explosion kleiner als diff dann
{
DoCast(me->getVictim(), SPELL_EXPLOSION); //Caste meinen Spell
t_explosion = 10000; //und setze den Timer zurück
}else t_explosion -= diff; //Wenn nicht zähle weiter herunter

10000 bedeutet 10Sekunden.

Tjoa und das wars dann auch schon...
Wer alles bis hier hin verstanden hat sollte schonmal kleine BossScripts schreiben können, wer nicht muss nochmal lesen

Bis hier war alles noch recht einfach gehalten, denn unser Boss machte ja nichts weiter als nur ein paar kleine Spells.
Weil aber einfache Spells auf die Dauer wenig herausforderung bieten können wir ihn ja auch mal ein paar kleine Adds spawnen lassen.
Für den Anfang reichen uns dumme Adds ohne Script.

Für diesen Vorgang nutzen wir den Befehl:
me->SummonCreature();
Leider will dieser X,Y,Z Coordinaten, welche wir uns entweder manuell besorgen (mit GPS) oder einfach von Boss nehmen.
Bei der letzteren Variante spawned der NPC aber an der Position des Bosses.
Was aber hier nicht stören sollte, denn es ist ja nur ein dummes Add.

Code:

```
enum Spells
{
    SPELL_BALNAZZAR            = 17288,
    SPELL_STOMP                  = 66330,
    SPELL_EXPLOSION            = 69839
};

enum Adds
{
    ADD_BOESER_WOLF            = 12345 //(Die ID 12345 ist fiktiv, also bitte durch eine reelle
};


class boss_meiner : public CreatureScript
{
public:
    boss_meiner() : CreatureScript("boss_meiner") { }

    CreatureAI* GetAI(Creature* pCreature) const
    {
        return new boss_meinerAI (pCreature);
    }

    struct boss_meinerAI : public ScriptedAI
    {
        boss_meinerAI (Creature *c) : ScriptedAI(c)
        { }

        uint32 t_stomp;
        uint32 t_explosion;
        uint32 t_add;

        void Reset()
        {
            t_stomp = 3000;
```

Wer den oberen Teil sorgsam durchgelesen hat, sich den Teil mit den Timern amgeschaut hat, der sieht was ich hier gemach habe.
Für alle anderen, ich habe einfach einen Enum(erator) für unsere Add-Sammlung erstellt (welcher hier noch aus einer NPC ID besteht) und lasse den Boss nun alle 11Sekunden einen Add spawnen.

Der Spawn geschieht hiermit.
me->SummonCreature (ADD_BOESER_WOLF, me->GetPositionX(), me->GetPositionY(), me->GetPositionZ());
Wie man sehen kann stellt uns der Boss seine aktuelle Position zur Verfügung, somit erspart es uns einen Fixpunkt zu bauen.
Klar kann man auch die GPS-Koordinaten nutzen und die Adds fix spawnen lassen, aber man muss bedenken das wenn das Add ausserhalb der Range ist nicht angreift.

me->SummonCreature (ADD_BOESER_WOLF, 1.987f, 0.898, -1.9866f);
Die XYZ-Coordinaten werden alle als Float(point) angegeben, weswegen unsere Zahlen hinten ein f brauchen.

ACHTUNG WISSEN
Wer sich nun fragt was ein Floatpoint ist, dem erkläre ich es schnell:
Ein Float ist eine Fließkommazahl, also eine Zahl mit einem Komma an einer beliebigen Stelle.
Das Gegenstück ist ein Int(integer) oder ein uint.
Ein Integer hat kein Komma und ist immer eine Ganzzahl (eine Zahl ohne Komma).
Der direkte Integer kann Vorzeichen +/- enthalten, ein uint hat kein Vorzeichen und ist immer positiv.
Wichtig währe auch zu wissen das ein uint in verschiedenen Größen existiert, die währen:
uint8. uint16, uint32, uint64, uint128-Bit.
Die Bit-Werte geben die maximale Größe einer Zahl an, der der uint speichert.
8=255
16= 65.535
32= 4.294.967.295
usw.
Wer mehr darüber lesen möchte: Integer (Datentyp)
Warum macht man das mit den 8,16,32,64 Bit?
Auch schnell geklärt: Wenn man weiss das eine Zahl niemals größer als 255 ist es Speicherverschwendung ist sie in einer 16 oder gar 128 Bit großen Variable abzulegen, denn egal wie groß der Wert in der Variable ist, der zugesicherte Speicher ist immer fest (als Vereinfachung 8-Bit=3 Stellen 64-Bit=20 Stellen).
Also platzsparend proggen, sonst liegen bald die wichtigsten Daten in der Swap/Auslagerungsdatei, ab dann gibts Performance einbußen.

Mal schauen vielleicht werde ich das ganze demnächst weiterführen

LG GiR-Blunti

PS:
Ich denke das wenn man sich die anderen Scripte anschaut auch schnell begreift, wie man diese korrekt im ScriptLoader und in CMAKE einbindet, weswegen ich es nicht erwähne

Alles ohne Gewähr und Pistole.

QUOTE QUOTE + THANKS

Thanks
10 Benutzer

Ammonit (25.04.2013), Arcn (19.12.2012), bumbummen99 (11.08.2013), coho (08.02.2013), Frosttall (18.01.2012), n1_Roxxer (20.01.2012), Psychoduke (23.01.2012), Sarumon (24.01.2012), Tulba (26.09.2014), °"Crazy"° (18.01.2012)

20.01.2012, 17:53 #2

megathorn
Junior Member

elite*gold: 0
The Black Market: 0/0/0
Registriert seit: Aug 2011
Beiträge: 16
Erhaltene Thanks: 0

Sehr guter Guide, wer sich den anschaut und etwas mit anderen Scripts vergleicht, sollte schon recht anständige Scripte schaffen können

QUOTE QUOTE + THANKS

23.01.2012, 12:00 #3

Kent_Brockman
Der einzig wahre Booksize

elite*gold: 0
The Black Market: 0/0/0
Registriert seit: Mar 2011
Beiträge: 525
Erhaltene Thanks: 194

Danke für das Feedback, vllt werd ich, wenn ich etwas mehr Zeit finde, noch erklären wie man AI-Übergreifend arbeiten kann.
Das gebe dann solchen "Scripts" noch die richtige Würze xD

QUOTE QUOTE + THANKS

24.01.2012, 15:14 #4

Sarumon
Learning AutoIT

Habe das Tut nir nal durchgelesen(und getestet) und es hat super geklappt.
Da meine Bosse in SS und GvS noch keine Texte haben, habe ich dies eingefügt.
Nun habe ich eine Frage.
Wo finde ich die IDs für das Sprechen wie z.B Krick in GvS sagt: "Den da, Den da, nimm den da!" ... wo finde ich solche IDs?
Währe super wenn du mir das sagen könntest.
Und dannn würde ich gerne wissen wie ich das in den Script einbaue da ich dazu nichts gefunden habe. Vielleicht auch nur überlesen.

MfG Sarumon

elite*gold: 0
The Black Market: 0/0/0
Registriert seit: Mar 2010
Beiträge: 208
Erhaltene Thanks: 83

Datenschutzinfo | In Linux | Install CD Linux | MySQL Linux | Linux Hosting

_____

Sollte ich geholfen haben bitte THANKS drücken.

Spoiler
Mein S4League Chars:

mysigs.de hat seinen Service eingestellt.
Wir danken allen Nutzern für ihre langjährige Treue.

mysigs.de hat seinen Service eingestellt.
Wir danken allen Nutzern für ihre langjährige Treue.

[Vorstellung] Redlight Wars - Macht, Ruhm, Verbrechen

QUOTE QUOTE + THANKS

11.04.2013, 07:44 #5

~~Shøxxø~~
Banned
ID VERIFIED
ELITE*GOLD TRADER

elite*gold: 5
The Black Market: 10/0/1
Registriert seit: May 2012
Beiträge: 460
Erhaltene Thanks: 44

Ich habe mich mal daran probiert und einen Boss mit 5 Spells gemacht, ist es soweit so richtig ???

Code:

```
#include "ScriptPCH.h"

enum Spells
{
    SPELL_SCHILD                        = 75381,
    SPELL_NATURBOMBE            = 64650,
        SPELL_STRAMPELN                         = 64639,
        SPELL_BERSERKER                         = 41924,
        SPELL_KETTE                                     = 75362
};

enum Adds
{
    ADD_BOESER_WOLF             = 34826
};

class boss_meiner : public CreatureScript
{
public:
    boss_meiner() : CreatureScript("boss_meiner") { }

    CreatureAI* GetAI(Creature* pCreature) const
    {
        return new boss_meinerAI (pCreature);
    }

    struct boss_meinerAI : public ScriptedAI
    {
        boss_meinerAI (Creature *c) : ScriptedAI(c)
        { }

                uint32 t_msal;
                uint32 t_explosion;
                uint32 t_stomp;
        uint32 t_add;
```

QUOTE QUOTE + THANKS

13.09.2014, 19:54 #6

jasdhl
Nabrezzelt

elite*gold: 0
The Black Market: 0/0/0
Registriert seit: Feb 2014
Beiträge: 1
Erhaltene Thanks: 0

Ich weiß nicht ob ich es überleben habe, aber dieses TuT/Script ist schon für Trinity oder?

QUOTE QUOTE + THANKS

14.09.2014, 14:01 #7

Kaev <3
Codemonkey

Im Titel steht, dass es sich um Scripts für TrinityCore handelt.
Aber ist teilweise outdated.

elite*gold: 110
The Black Market: 20/0/0
Registriert seit: Mar 2008
Beiträge: 747
Erhaltene Thanks: 314

QUOTE QUOTE + THANKS

15.09.2014, 20:03 #8

Kent_Brockman
Der einzig wahre Booksize

elite*gold: 0
The Black Market: 0/0/0
Registriert seit: Mar 2011
Beiträge: 525
Erhaltene Thanks: 194

Nach 2 Jahren ist das durchaus möglich
Ich bin auch nicht mit der aktuellen API vertraut, weil ich mehr in den Interna der Core werkel oder andere Dinge programmiere und beruflich ne Menge um die Ohren habe.
_____

QUOTE QUOTE + THANKS

Thanks
2 Benutzer

Bin.Reader (27.04.2015), Kaev <3 (16.09.2014)

NEW REPLY >>

Direkt antworten

Nachricht:

Optionen
Beitrag in Antwort zitieren?

Antworten Erweitert

« Vorheriges Thema | Nächstes Thema »

Ähnliche Themen

GMH Trinity 1.0.9
nvm pls löschen
0 Antworten - WoW Addons

Ähnliche Themen

Trinity DB 3.3.3a
Hi, Ich suche eine deutsche trinity datenbank auf dem patch 3.3.3a. Wenn möglich were eine schon bearbeitete db sehr gut (bugfixxes) sehr gut....
5 Antworten - WoW Private Server

UDB Datenkbank Trinity tauglich machen (2.4.3, aktuelster Trinity Core)
Gibts da nen Converter? Ich arbeite zum ersten mal mit Trinity und häng jetzt an der Datenbank fest :-(bekomm leider beim starten immer Errors und...
2 Antworten - WoW PServer Hosting

trinity
any working hack for trinity...i dont need like speed or others just the godmode working Edit: OMG so many are hacking in trinity and no1 got the...
1 Antworten - GunZ

Forumregeln

Es ist dir **erlaubt**, neue Themen zu verfassen.
Es ist dir **erlaubt**, auf Beiträge zu antworten.
Es ist dir **erlaubt**, Anhänge hochzuladen.
Es ist dir **erlaubt**, deine Beiträge zu bearbeiten.

BB-Code ist **an**.
Smileys sind **an**.
[IMG] Code ist **an**.
HTML-Code ist **aus**.
Trackbacks sind **aus**
Pingbacks sind **aus**
Refbacks sind **aus**

Forenregeln

Gehe zu
WoW PServer Hosting Los

Alle Zeitangaben in WEZ +2. Es ist jetzt 07:15 Uhr.

-- elitepvpers -- German (DE)

elitepvpers - play less, get more - Nach oben