Modificateur de rate d'EXP

Mithrandir

Modificateur de rate d'EXP

Bien le bonjour,

On va voir dans ce petit tutoriel comment créer votre propre modificateur de Rate d'XP. Ce tutoriel est écrit pour fonctionner sur AzerothCore qui permet de créer des modules.

sommaire

1. Création du dossier du module
2. Comprendre le système
3. Création & Modifications des fichiers
4. Création du code
5. Conclusion

Indication

`$AC_ROOT` = Remplacer `$AC_ROOT` par le chemin du dossier racine d'azerothCore

Création du dossier du module

Avant toute chose nous devons créer le dossier qui contiendra le module, pour ce faire placer vous dans votre dossier racine d'azerothCore

```
cd $AC_ROOT/modules
sh create_module.sh
```

```
Enter the name of your future module:  mod_xp
```

Une fois fait, vous pouvez supprimer `MyPlayer.cpp`, supprimez la ligne

```
AC_ADD_SCRIPT("${CMAKE_CURRENT_LIST_DIR}/src/MyPlayer.cpp")
```

Nôtre dossier est prêt !

Passons aux explications !!

Comprendre le système

Le modificateur de rate d'xp se place en tant que "middleware", c'est à dire, qu'il va modifier le montant d'xp que la personne reçevra.

Ce que nous devrons faire, c'est sauvegarder le choix du joueur dans une variable, avec son `GUID` on arrivera à retrouver le rate qu'il souhaite

On peut donc voir le système comme suit :

- Le joueur tue la créature
- Notre algorithme intercepte le gain d'xp
- Après vérification dans une variable, nous retrouvons le choix de modification du rate d'xp

- Après vérification dans une variable, nous retrouvons le choix de modification du taux d'xp
- On applique la modification
- Mission finie !

Création des fichiers

Donc nous allons devoir créer un fichier que l'on nommera `XPPlayer.hpp` et `XPPlayer.cpp` ainsi que `XPCreature.hpp` et `XPCreature.cpp`

Avant de continuer plus loins nous allons modifier le nom de notre scripts dans le fichier `CMakeLists.txt`
Remplacer donc

```
AC_ADD_SCRIPT_LOADER("MyPlayer" "${CMAKE_CURRENT_LIST_DIR}/src/loader.h")
```

par

```
AC_ADD_SCRIPT_LOADER("XPModifier" "${CMAKE_CURRENT_LIST_DIR}/src/loader.h")
```

Restez toujours dans le fichier dans le CMake, nous allons ajouter nos fichier sources et d'en-tête !

Ajouter donc avant la ligne que l'on vient de modifier au dessus ceci :

```
AC_ADD_SCRIPT("${CMAKE_CURRENT_LIST_DIR}/src/XPCreature.hpp")
AC_ADD_SCRIPT("${CMAKE_CURRENT_LIST_DIR}/src/XPPlayer.hpp")
AC_ADD_SCRIPT("${CMAKE_CURRENT_LIST_DIR}/src/XPCreature.cpp")
AC_ADD_SCRIPT("${CMAKE_CURRENT_LIST_DIR}/src/XPPlayer.cpp")
```

Ajoutez dans le fichier `XPPlayer.cpp` cette ligne :

```
void AddSC_XPPlayer()
{
    new XPPlayer();
}
```

et dans le fichier `XPCreature.cpp` cette ligne:

```
void AddSC_XPCreature()
{
    new XPCreature();
}
```

Maintenant dirigeons nous dans le fichier `loader.hpp` ajouter ces lignes :

```
#ifndef XP_LOADER_H

#define XP_LOADER_H

void AddSC_XPPlayer();
void AddSC_XPGroup();

void AddXPModifierScripts() {
    AddSC_XPPlayer();
    AddSC_XPGroup();

}

#endif
```

Création du code

Nous voilà maintenant dans la partie la plus intéressante :)

Partie joueur

Nous devons ajouter un héritage, nous devons hériter de `PlayerScript` et aussi d'ajouter le constructeur par défaut :

```
#ifndef AZEROTHCORE_XPPLAYER_HPP
#define AZEROTHCORE_XPPLAYER_HPP


class XPPlayer : public PlayerScript {
public:
    XPPlayer() : PlayerScript("XPModifierPlayer") {};
};

#endif //AZEROTHCORE_XPPLAYER_HPP
```

maintenant nous pouvons ajouter un attribut static à la classe XPPlayer : on le nommera `modXp` (vous pouvez choisir ce que vous voulez :))

```
#ifndef AZEROTHCORE_XPPLAYER_HPP
#define AZEROTHCORE_XPPLAYER_HPP


class XPPlayer : public PlayerScript {
public:
    XPPlayer() : PlayerScript("XPModifierPlayer") {};


    static std::map<uint64,float> modXp;
};


#endif //AZEROTHCORE_XPPLAYER_HPP
```

Cet attribut statique est une variable partagée, c'est à dire que peut importe l'instance de XPPlayer, vous accéderez toujours aux mêmes données pour `modXp` .
Comme vous pouvez le constater le type de cet attribut est : `std::map<uint64,float>` en effet nous allons stocker la modification du rate d'xp dans un map qui permettra d'avoir un accés clé <-> valeur personnalisé.

Il faudra l'inclure dans le fichier `XPPlayer.cpp` :

```
#include "XPPlayer.hpp"

std::map<uint64,float> XPPlayer::modXp;

void AddSC_XPPlayer()
{
    new XPPlayer();
}
```

Pour pouvoir "attraper" la valeur du montant d'xp que recevra le joueur il faut `override` la méthode :

```
void OnGiveXP(Player* /*player*/, uint32& /*amount*/, Unit* /*victim*/) { }
```

En effet c'est cette méthode qui sera appelé lorsqu'un joueur recevra de l'xp (en fait elle est appelée juste avant qu'il ne reçoit l'xp)

Notre classe ressemblera donc à ça :

```
#ifndef AZEROTHCORE_XPPLAYER_HPP
#define AZEROTHCORE_XPPLAYER_HPP


class XPPlayer : public PlayerScript {
public:
    XPPlayer() : PlayerScript("XPModifierPlayer") {};
    void OnGiveXP(Player* /*player*/, uint32& /*amount*/, Unit* /*victim*/) override;

    static std::map<uint64,float> modXp;
};


#endif //AZEROTHCORE_XPPLAYER_HPP
```

Maintenant il faut créer le corps de notre méthode, la description de l'algorithme ce fait donc dans le fichier `XPPlayer.cpp`

```
#include "XPPlayer.hpp"

std::map<uint64,float> XPPlayer::modXp;

void XPPlayer::OnGiveXP(Player* Player, uint32& amount, Unit* /*victim*/) {

}

void AddSC_XPPlayer()
{
    new XPPlayer();
}
```

Nous allons donc nous atteler à décrire le fonctionnement de notre algorithme.
Tout d'abord regardons si notre joueur existe ! Il faut donc vérifier que le pointeur `Player* player` n'est pas null (`nullptr` pour pointeur null)

```
if(player == nullptr) {
    return;
}
```

On vient donc de vérifier que le joueur existe ! (si le joueur est égal à un pointeur null alors on éxecute pas la fonction on `return`)

Maintenant il faut savoir si notre personnage est inscrit dans le `std::map<uint64,uint32>` : nous allons donc utiliser la méthode `std::find` de la `STL (Standard Template Library)` :

```
if(std::find(XPPlayer::modXp.begin(), XPPlayer::modXp.end(), player->GetGUID()) != XPPlayer::modXp.end()) {

}
```

On regarde donc dans le map si il existe une occurence de notre joueur si oui alors son iterateur est différent de l'itérateur de fin du map, et on peut lui appliquer la modification du rate d'xp

```
if(std::find(XPPlayer::modXp.begin(), XPPlayer::modXp.end(), player->GetGUID()) != XPPlayer::modXp.end()) {
    amount *= XPPlayer::modXp[player->GetGUID()];
```

```
}
```

Notre fonction nous donne donc ceci :

```
void XPPlayer::OnGiveXP(Player * player, uint32 & amount, Unit * /*victim*/) {
    if(player == nullptr) {
        return;
    }

    if(std::find(XPPlayer::modXp.begin(), XPPlayer::modXp.end(), player->GetGUID()) != XPPlayer::modXp.end()) {

        amount *= XPPlayer::modXp[player->GetGUID()];

    }

    return;
}
```

Partie Créature

Nous allons maintenant nous intéresser au script de la créature, ici je ne ferais pas de tutoriel sur `comment créer votre créature`, je pense qu'il y a bien des tutoriels pour cela. Donc je pars du principe que vous avez une créature sous la main configurer en `GOSSIP`

Donc la même chose que pour le joueur, nous devons ajouter un héritage, on fait donc hériter XPCreature de `CreatureScript` ce qui nous donne donc dans le fichier `XPCreature.hpp` :

```
#ifndef AZEROTHCORE_XPCREATURE_HPP
#define AZEROTHCORE_XPCREATURE_HPP


class XPCreature : public CreatureScript {
public:
    XPCreature() : CreatureScript("XPModifierCreature") {};
};


#endif //AZEROTHCORE_XPCREATURE_HPP
```

Nous allons, comme pour la classe pour le joueur, override certaines méthodes :

```
bool OnGossipHello(Player * /*player*/, Creature * /*creature*/) override;
bool OnGossipSelect(Player * /*player*/, Creature * /*creature*/, uint32 /*sender*/, uint32 /*action*/) override;
```

Notre fichier d'en-tête ressemble donc à :

```
#ifndef AZEROTHCORE_XPCREATURE_HPP
#define AZEROTHCORE_XPCREATURE_HPP


class XPCreature : public CreatureScript {
public:
    XPCreature() : CreatureScript("XPModifierCreature") {};
    bool OnGossipHello(Player * /*player*/, Creature * /*creature*/) override;
    bool OnGossipSelect(Player * /*player*/, Creature * /*creature*/, uint32 /*sender*/, uint32 /*action*/) override;
};


#endif //AZEROTHCORE_XPCREATURE_HPP
```

```
#endif //AZEROTHCORE_XPCREATURE_HPP
```

et notre fichier source à :

```
#include "XPCreature.hpp"
void AddSC_XPCreature()
{
    new XPCreature();
}

bool XPCreature::OnGossipHello(Player *, Creature *) {

}

bool XPCreature::OnGossipSelect(Player *, Creature *, uint32, uint32) {

}
```

Il faut maintenant ajouter la variable statique que nous avons déclarer dans le fichier source `XPCreature.cpp`

```
#include "XPCreature.hpp"
#include "XPPlayer.hpp"

std::map<uint64,float> XPPlayer::modXp;

void AddSC_XPCreature()
{
    new XPCreature();
}

bool XPCreature::OnGossipHello(Player * player, Creature * creature) {

    if(player == nullptr || creature == nullptr) {
        return true;
    }

}

bool XPCreature::OnGossipSelect(Player * player, Creature * creature, uint32 /*sender*/, uint32 action) {
    if(player == nullptr || creature == nullptr) {
        return true;
    }
}
```

Vous verrez ici que j'ai déjà préparé le terrain, en effet j'ai ajouter les vérifications de l'existence du joueur & de la créature.

Maintenant il faut dire au serveur que lorsque le joueur clique sur le PNJ il affiche un menu :

Deux fonctions existe pour cela :

```
AddGossipItemFor(...);
SendGossipMenuFor(...);
```

La première fonction nous permettra d'inscrire une ligne dans le menu et la seconde d'envoyé les données au client afin d'afficher le menu :)

Notre code va donc ressembler à ceci :

```
#include "XPCreature.hpp"
#include "XPPlayer.hpp"
```

```
#include "XPPlayer.hpp"

std::map<uint64,float> XPPlayer::modXp;

void AddSC_XPCreature()
{
    new XPCreature();
}

bool XPCreature::OnGossipHello(Player * player, Creature * creature) {

    if(player == nullptr || creature == nullptr) {
        return true;
    }


    AddGossipItemFor(player, GOSSIP_ICON_CHAT, "Rate x1", GOSSIP_SENDER_MAIN, 1);
    AddGossipItemFor(player, GOSSIP_ICON_CHAT, "Rate x2 ",GOSSIP_SENDER_MAIN, 2);
    AddGossipItemFor(player, GOSSIP_ICON_CHAT, "Rate x3 ",GOSSIP_SENDER_MAIN, 2);

    SendGossipMenuFor(player, 68, creature);

}

bool XPCreature::OnGossipSelect(Player * player, Creature * creature, uint32 /*sender*/, uint32 action) {
    if(player == nullptr || creature == nullptr) {
        return true;
    }
}
```

Ici j'ai ajouté trois options, `Rate x1` `Rate x2` `Rate x3` , elles sont liées par un numéro dans l'ordre 1, 2 et 3.
L'astuce est de capturer le numéro (uint32 action) et de faire les manipulations que nous souhaitons ainsi dans la méthode `OnGossipSelect` nous aurons :

```
bool XPCreature::OnGossipSelect(Player *player, Creature *creature, uint32 /*sender*/, uint32 action) {
    if (player == nullptr || creature == nullptr) {
        return true;
    }

    ClearGossipMenuFor(player);

    float mod=1.00;

    switch (action) {
        case 1: {
            mod=1.00;
            break;
        }
        case 2: {
            mod=2.00;
            break;
        }
        case 3: {
            mod=3.00;
            break;
        }
    }
```

```
    if(std::find(XPPlayer::modXp.begin(), XPPlayer::modXp.end(), player->GetGUID()) != XPPlayer::modXp.end())
    {
        XPPlayer::modXp[player->GetGUID()] = mod;
    } else {
        XPPlayer::modXp.insert(std::make_pair(player->GetGUID(), mod));
    }

    CloseGossipMenuFor(player);

    return true;
}
```

En première étape je nettoie le menu du joueur, puis j'initialise une variable mod (qui temporairement sauvegardera la modification d'xp), après je regarde le choix du joueur puis je modifie le modificateur selon le choix. Ensuite je regarde si l'utilisateur est déjà connue dans la map, si c'est le cas je le modifie directement :) par contre si cela n'est pas le cas je l' `insert` grâce à `std::make_pair` une fois cela fait, je ferme le gossip et retourne `true`

Au final les algorithmes doivent ressembler à

XPCreature.cpp

```
#include "XPCreature.hpp"
#include "XPPlayer.hpp"

std::map<uint64, float> XPPlayer::modXp;

void AddSC_XPCreature() {
    new XPCreature();
}

bool XPCreature::OnGossipHello(Player *player, Creature *creature) {

    if (player == nullptr || creature == nullptr) {
        return true;
    }


    AddGossipItemFor(player, GOSSIP_ICON_CHAT, "Rate x1", GOSSIP_SENDER_MAIN, 1);
    AddGossipItemFor(player, GOSSIP_ICON_CHAT, "Rate x2 ", GOSSIP_SENDER_MAIN, 2);
    AddGossipItemFor(player, GOSSIP_ICON_CHAT, "Rate x3 ", GOSSIP_SENDER_MAIN, 2);

    SendGossipMenuFor(player, 68, creature);

}

bool XPCreature::OnGossipSelect(Player *player, Creature *creature, uint32 /*sender*/, uint32 action) {
    if (player == nullptr || creature == nullptr) {
        return true;
    }

    ClearGossipMenuFor(player);

    float mod=1.00;

    switch (action) {
        case 1: {
            mod=1.00;
            break;
        }
        case 2: {
```

```
            mod=2.00;
            break;
        }
        case 3: {
            mod=3.00;
            break;
        }
    }


    if(std::find(XPPlayer::modXp.begin(), XPPlayer::modXp.end(), player->GetGUID()) != XPPlayer::modXp.end())
    {
        XPPlayer::modXp[player->GetGUID()] = mod;
    } else {
        XPPlayer::modXp.insert(std::make_pair(player->GetGUID(), mod));
    }

    CloseGossipMenuFor(player);

    return true;
}
```

XPCreature.hpp

```
#ifndef AZEROTHCORE_XPCREATURE_HPP
#define AZEROTHCORE_XPCREATURE_HPP


class XPCreature : public CreatureScript {
public:
    XPCreature() : CreatureScript("XPModifierCreature") {};
    bool OnGossipHello(Player * /*player*/, Creature * /*creature*/) override;
    bool OnGossipSelect(Player * /*player*/, Creature * /*creature*/, uint32 /*sender*/, uint32 /*action*/) override;
};


#endif //AZEROTHCORE_XPCREATURE_HPP
```

XPPlayer.cpp

```
#include "XPPlayer.hpp"

std::map<uint64,float> XPPlayer::modXp;

void XPPlayer::OnGiveXP(Player * player, uint32 & amount, Unit * /*victim*/) {
    if(player == nullptr) {
        return;
    }

    if(std::find(XPPlayer::modXp.begin(), XPPlayer::modXp.end(), player->GetGUID()) != XPPlayer::modXp.end()) {

        amount *= XPPlayer::modXp[player->GetGUID()];

    }

    return;
}
```

```
void AddSC_XPPlayer()
{
    new XPPlayer();
}
```

XPPlayer.hpp

```
#ifndef AZEROTHCORE_XPPLAYER_HPP
#define AZEROTHCORE_XPPLAYER_HPP


class XPPlayer : public PlayerScript {
public:
    XPPlayer() : PlayerScript("XPModifierPlayer") {};
    void OnGiveXP(Player* /*player*/, uint32& /*amount*/, Unit* /*victim*/) override;

    static std::map<uint64,float> modXp;
};


#endif //AZEROTHCORE_XPPLAYER_HPP
```

Conclusion

J'ai crée ce tutoriel dans l'optique de vous aider à comprendre comment fonctionne un script de modification de script, il reste beaucoups de chose non faites, et c'est voulue, je ne souhaite pas que le scripts soit copier coller sans qu'il ne soit compris, vous pouvez par la suite de ce tutoriel, ajouter des éxecutions dans la base de données pour sauvegarde le choix du joueur.

Si vous avez la moindre question je reste disponible sur le discord d' `Open-Wow`

---

iThorgrim

Super tutoriel, franchement c'est niquel surtout pour les débutant :D
