Christian Semmler, Matthias Rein

VAULT-TEC MULTIPLAYER MOD

Softwareprojekt
Informatik EA
Dezember 2010

Vault-Tec Multiplayer Mod („vaultmp“) ist eine Multiplayer-Modifikation für die PC-Version von Bethesda Softworks Action-Rollenspiel „Fallout 3“ (Spiel des Jahres 2008). Fallout 3 spielt in einem post-apokalyptischen Setting im Jahr 2277 in der US-amerikanischen Hauptstadt Washington D.C. Durch einen weltweiten Atomkrieg im Jahr 2077 bedingt ist der überwiegende Teil der Menschen von der Erde verschwunden, die einzigen Überlebenden halten sich in Atomschutzbunkern auf, den sogenannten „Vaults“. In der „Vault 101“ startet die Geschichte des Spiels mit der Geburt des Protagonisten. Im weiteren Verlauf verlässt der Protagonist die Vault 101, begibt sich in die verstrahlten Areale von Washington D.C. („Ödland der Hauptstadt“) und trifft dort auf konkurrierende Splittergruppen von Menschen, Mutanten und eine allgemein lebensfeindliche Umgebung.

Das Spiel zeichnet sich durch einen hohen Detailgrad in der Story aus und präsentiert sich selbst in sarkastischer Art und Weise; die dargestellte Menschheit scheint in Sachen Kultur und Lebensweise in die 50er bzw. 60er-Jahre des 20. Jahrhunderts zurückgefallen zu sein, während sich der Stand der Technik auf dem Niveau von z.B. tragbaren Atomsprengköpfen, Laserwaffen und intelligenten, fliegenden Robotern befindet. Der atmosphärische Kontrast macht den besonderen Reiz von Fallout 3 aus.

Unsere Überlegung besteht darin, die Welt von Fallout 3 für mehr als nur einen Spieler gleichzeitig betretbar zu machen. Dazu zählt in erster Linie das Synchronisieren der Spielcharaktere sowie deren Aktionen und Animationen. Eine Neukonfiguration der Story und Aufträge, die man während des Spiels erfüllen muss, ist bisher nicht geplant und würde vermutlich auch den Rahmen des uns Möglichen sprengen. Im Folgenden unser derzeitiges Konzept sowie der Implementierungsstatus:

vaultmp bedient sich der Sprache C++. Um eine Synchronisation der Spielerdaten zu ermöglichen, wird ein direkter Zugang zu der bereits vorhandenen Script-Engine von Fallout 3 benötigt (über die man wiederum Zugriff auf die gewünschten Daten haben kann). Das macht es erforderlich, Teile des Maschinencodes des dafür zuständigen Moduls „Fallout3.exe“ neu zu schreiben bzw. ab zu ändern. Um das zu erreichen, wird eine DLL mit dem nötigen Code injiziert bevor der Prozess anläuft.
Es gibt einen Client mit grafischer Oberfläche, über den man eine Liste mit laufenden Servern empfangen kann, die sich bei einem MasterServer angemeldet haben. Der dedizierte Server hat neben seiner Hauptfunktion, die Spieldaten zu empfangen und weiterzugeben, eine abstrakte Maschine implementiert, über die zuvor kompilierte Scripts geschrieben in der Sprache PAWN interpretiert werden können.
Eine zusätzliche grafische Oberfläche (GUI) im Spiel ist geplant, um u.a. einen Chat bereit stellen zu können. Das wurde bisher aber noch nicht umgesetzt.

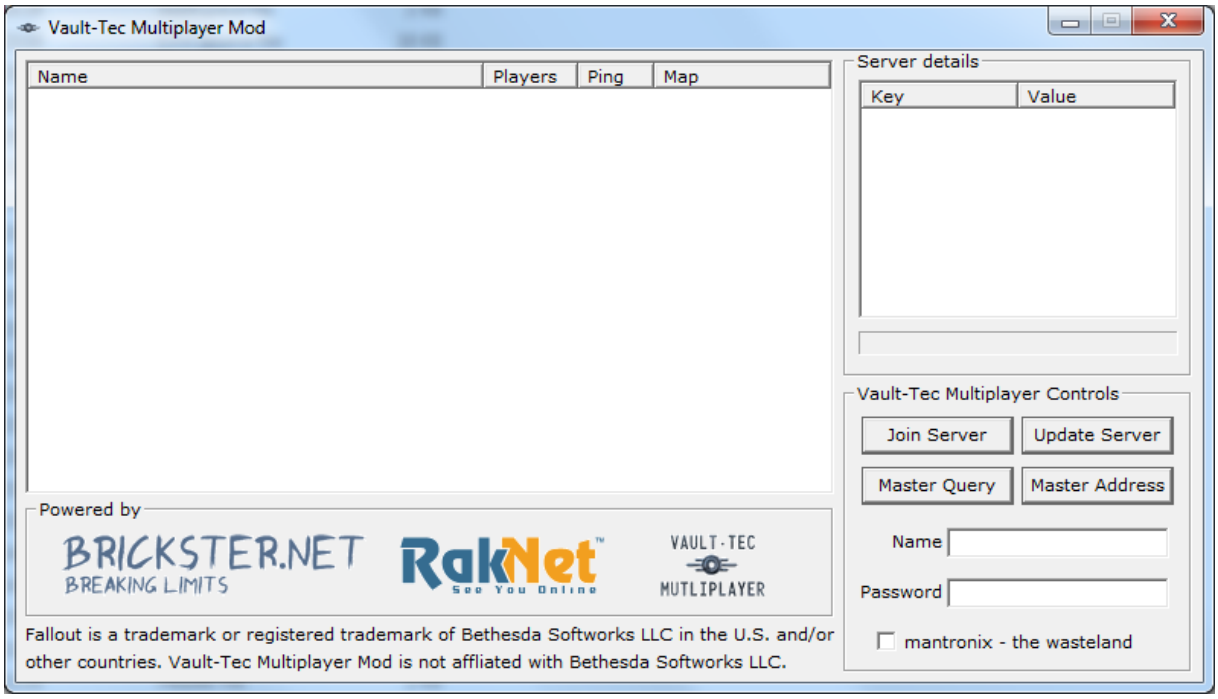
vaultmp benutzt folgende zusätzliche, quelloffene Bibliotheken:

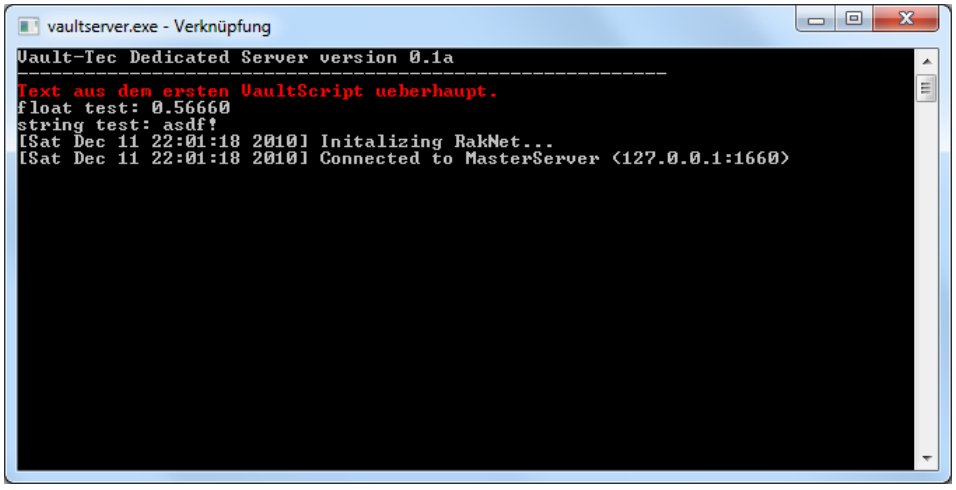
RakNet (http://www.jenkinssoftware.com/raknet/index.html)
CEGUI (http://www.cegui.org.uk/)
PAWN (http://www.compuphase.com/pawn/pawn.htm)
uFMOD (http://ufmod.sourceforge.net/)

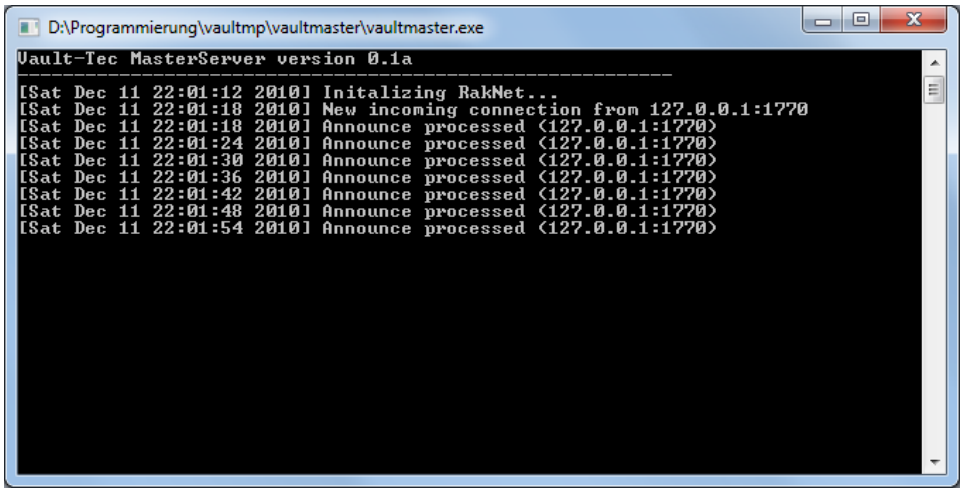
Version 0.1a, Revision 74
http://code.google.com/p/vaultmp/

Christian Semmler, Matthias Rein

VAULT-TEC MULTIPLAYER MOD







VAULT-TEC MULTIPLAYER MOD

Ein mögliches (nicht-kompiliertes) Script in der C-ähnlichen Sprache PAWN für die abstrakte Maschine des dedizierten Servers könnte nach der jetzigen Implementierung so aussehen:

```
1  #include <vaultmp>
2
3  main()
4  {
5          print("Text aus dem ersten VaultScript ueberhaupt.\n", 1, -1, 1);
6          new Float: asdf;
7          asdf = 0.5666;
8          printf("float test: %f\n", asdf);
9          new blub[32] = "asdf";
10         if (strcmp(blub, "asdf") == 0)
11                 print("string test: asdf!\n");
12 }
13
14 public OnClientAuthenticate(clientID, const name[], const pwd[])
15 {
16         printf("OnClientAuthenticate: ID: %d name: %s, pwd: %s\n", clientID, name, pwd);
17         return 1;
18 }
19
20 public OnClientRequestGame(clientID, savegame[], len)
21 {
22         printf("OnClientRequestGame: ID: %d savegame: %s, len: %d\n", clientID, savegame, len);
23         return 1;
24 }
```

Die Callbacks in Zeile 14 und 20 werden vom Server aufgerufen. Der Verfasser des Scripts kann nun z.B. im Falle von *OnClientAuthenticate* entscheiden, ob der neu hinzugestoßene Client, der sich über die Eingabefelder im Client mit Namen und Kennwort identifiziert, zugelassen werden soll oder nicht. Die Rückgabe eines Werts ungleich Null würde den Client als Spieler zulassen und umgekehrt. Ähnlich verhält es sich mit *OnClientRequestGame*. Hier kann zusätzlich ein Fallout 3 Speicherplatz angegeben werden, welches der neue Spieler dann zu laden hat.

Der Teil des Maschinencodes, der dafür zuständig ist, einen Fallout 3 Script-Engine Befehl von der Modifikation in die Engine zu übertragen, sieht so aus:

```
1   50                  PUSH EAX                    # Routine um einen C-String (Befehl) aus vaultmp in die Engine
2   51                  PUSH ECX                    # byteweise zu kopieren; EAX, ECX und ESI brauchen wir, also
3   56                  PUSH ESI                    # sichern wir die Inhalte der Register auf dem Stack
4   8A06                MOV AL,BYTE PTR DS:[ESI]    # ESI beinhaltet den Zeiger zum Speichersegment wo der Befehl
5   3C 00               CMP AL,0                    # steht und von dem aus er geparst wird; wenn das erste Byte dort
6   74 0D               JE SHORT XXXXXXXX           # 0x00 ist, dann können wir zu Zeile 12 springen
7   5E                  POP ESI                     # Ende der Routine; Inhalte wieder zurück in ihre Register
8   59                  POP ECX
9   58                  POP EAX
10  BA FFFEFE7E         MOV EDX,7EFEFEFF            # Überbleibsel aus Fallout3.exe Modul
11  E9 XXXXXXXX         JMP Fallout3.00C075B4       # Rücksprung in das Fallout3.exe Modul
12  B9 XXXXXXXX         MOV ECX,XXXXXXXX            # Zeiger zum Quellstring im Speicher in ECX kopieren
13  8A01                MOV AL,BYTE PTR DS:[ECX]    # Erstes Byte vom Speicher in AL kopieren
14  3C 00               CMP AL,0                    # Wenn AL = 0x00, dann gibt es nichts zu kopieren
15  74 E8               JE SHORT XXXXXXXX           # In diesem Fall, Rücksprung zu Zeile 7 (Ende der Routine)
16  8806                MOV BYTE PTR DS:[ESI],AL    # Gelesenes Byte an die Ziel-Speicherstelle kopieren
17  C601 00             MOV BYTE PTR DS:[ECX],0     # Quell-Speicherstelle auf 0x00 setzen (leeren)
18  83C1 01             ADD ECX,1                   # Ein Byte wurde kopiert; erhöhe die Zeiger um 1 Byte, damit
19  83C6 01             ADD ESI,1                   # das nächste Byte gelesen / geschrieben werden kann
20  8A01                MOV AL,BYTE PTR DS:[ECX]    # Kopiere das nächste Byte von der Quelle in AL
21  3C 00               CMP AL,0                    # Wenn AL = 0x00, dann sind wir fertig mit Kopieren
22  74 02               JE SHORT XXXXXXXX           # In diesem Fall, Sprung zu Zeile 24
23  EB ED               JMP SHORT XXXXXXXX          # Wir sind noch nicht fertig; Rücksprung zu Zeile 16
24  C605 XXXXXXXX 00    MOV BYTE PTR DS:[XXXXXXXX],0 # Mutex von vaultmp auf 0x00 setzen (wir sind mit Kopieren
25  EB CC               JMP SHORT XXXXXXXX          # fertig); Rücksprung zu Zeile 7 (Ende der Routine)
```