Recherchebericht

Konfigurator

Der Konfigurator des MUDs soll einem User erlauben, einen Multi User Dungeon in der Form zu konfigurieren, dass auf Basis der Wünsche und Anforderungen dieses Users, ein Spielablauf nach diesen Aspekten realisierbar ist

In dem Konfigurator werden mehrere Features eines MUDs konfigurierbar sein und sollten auch konfiguriert werden.

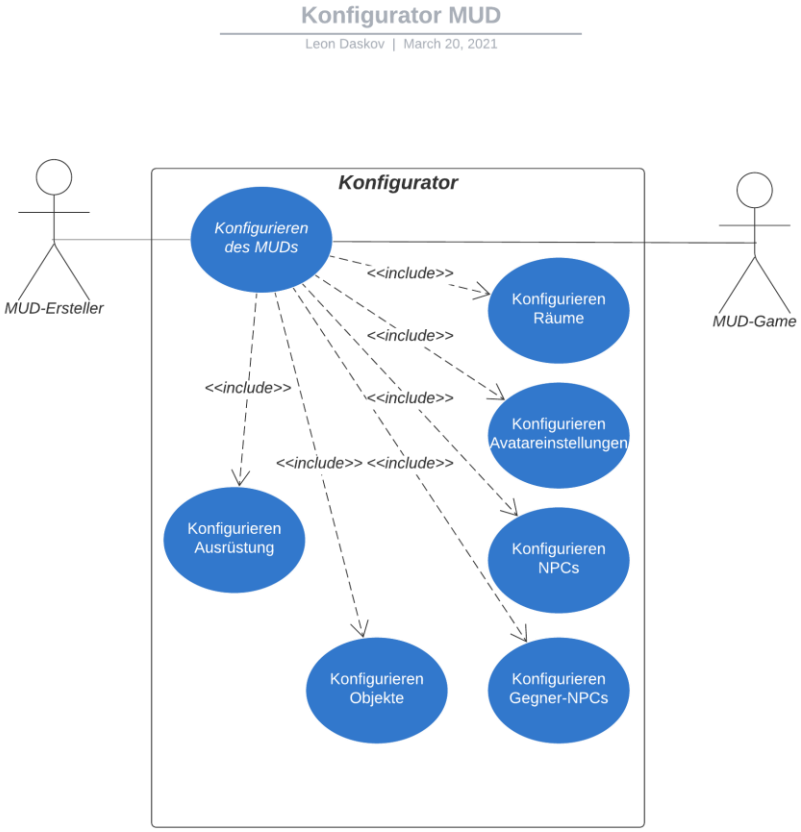
Das Erstellen von Räumen

Jeder Raum innerhalb des MUDs soll Objekte, NPCs und Gegner-NPCs enthalten, also andere konfigurierbare Objekte. Dies soll mit Initialwerten konfigurierbar sein, d.h. der Ersteller des Dungeons kann einstellen, dass zu Beginn des Spieles Raum X mit dem Item Y versehen ist.
Um zu gewährleisten, dass die Spieleserver nicht überlastet werden und dass die Spieler nicht von der Masse an Objekten in einem Raum überwältigt werden, wird eine Maximalanzahl an zulässigen Objekten in jedem Raum zu definieren sein.
Jeder Raum besitzt die Möglichkeit in Zeitintervallen bestimmte Objekte, NPCs und Gegner-NPCs zu generieren, damit die Räume nach dem initialen Besuchen dieser, nicht nur als „nutzloser“ Raum existieren. Dies ermöglicht das Wiederverwenden der Räume und führt zu einem interessanteren Spielfluss.
Damit die Räume einen genutzt werden können, muss in der Konfiguration auch einstellbar sein, mit welchen Räumen sie verbunden sind. Ein Raum X soll zu Raum Y führen und verbunden sein, jedoch ist Raum Z nicht zu erreichen von Raum X, ohne durch einen anderen Raum zu traversieren.
Durch die Erstellung der Räume wird eine Basis für die Spielwelt des MUDs gestellt.

Das Definieren des Spielercharakter-Optionen („Avatar“)

In diesem Abschnitt muss zwischen Charaktererstellung und Charakterkonfiguration differenziert werden. Die Charakterkonfiguration bezieht sich, auf das Aufstellen der Möglichkeiten der Erstellung des MUD-Masters und die Erstellung bezieht sich auf das eigentliche Erstellen der Charaktere als Spielabschnitt/Spielschritt.
Jeder Spielercharakter soll mit Basisinformationen gefüllt werden damit diese am Spiel teilnehmen können. Jeder Charakter soll mit konfigurierbaren Rassen, Klassen und Ausrüstungsarten versehen werden, damit der Spieler „eigene“ Charaktere erstellen kann. Die Rassen, Klassen und verfügbaren Ausrüstungsarten für die jeweiligen Klassen, werden separat konfiguriert und in der Erstellung für die Charaktere werden verfügbare Rassen, Klassen und Ausrüstungsarten dann angeboten.
Die Klassen und Rassen können mit speziellen Events konfiguriert werden um dem Spielverlauf zusätzliche Möglichkeiten auf Basis der Wahl des Spieler anzubieten.
Die Aussehensoptionen können im späteren Entwicklungsverlauf auch erstellt

werden und dann konfiguriert werden, jedoch ist dies von keiner höheren Priorität.
Die Größe des Inventars soll in der Charakterkonfiguration einstellbar sein, aber nicht in der Charaktererstellung.
Einem Spielercharakter können in der Konfiguration initiale Statuswerte wie „HP“ (Health Points) oder später auch Stats gegeben werden. Dies kann auch in Relation mit den Klassen und Rassen geschehen.



Bereits existierende MUD-Creators

http://dead-souls.net/

Dead Souls besteht zum einen aus einem “Driver”, der den eigentlichen Server darstellt. Dieser lädt dann die “mudlib”, eine Anzahl von Dateien, die das Spielverhalten bestimmen. Die mudlib ist in LPC, einer auf C aufbauenden Sprache speziell für MUDs, geschrieben und kann vom Benutzer beliebig erweitert werden. Spieler können sich per Telnet einloggen, wobei das wahrscheinlich erweitert werden kann.

https://www.evennia.com/

Evennia, ein ziemlich moderner und bekannter MUD-Creator ist in Python geschrieben. Der Quellcode ist öffentlich auf GitHub und kann beliebig geforkt werden. Dieser Creator benutzt Django, ein Webframework mit Datenbankunterstützung, Templating für Websiten, Authentifizierung, uvm.. Im Javaumfeld ist es mit Spring vergleichbar. Mit dem Framework Twisted setzt Evennia auf eine Event-gesteuerte Architektur und bietet Unterstützung für verschiedene Protokolle: Websockets für den Webclient, Telnet (+ SSL) und SSH für traditionelle Clients. Der Chat läuft hier ingame über Commands ab. Das Spielverhalten wird auch hier mit Python über den Source Code bestimmt. Das heißt, es können beliebige Erweiterungen und Spielmöglichkeiten umgesetzt werden.

Mögliche Technologien

Kommunikation

Da der Client im Browser laufen soll, bietet es sich hier an, statt Telnet oder SSH Standardtechnologien von Browsern zu nutzen. Am besten eignen sich Websockets, da diese anders als REST eine bidirektionale Verbindung zwischen Client und Server ermöglichen.

Server

Da die meisten aus der Gruppe bereits Erfahrungen mit Java haben, konzentrieren wir uns auf die JVM. Auch wollen wir Kotlin verwenden, da diese Sprache gegenüber Java benutzerfreundlicher ist und bedeutende Features wie Nullsicherheit oder Patternmatching bietet. Es ist möglich beide Sprachen zusammen zu verwenden.

Ktor

Für Kotlin gibt es das in diesem Umfeld ziemlich bekannte Webframework Ktor. Da es nicht auf Java EE setzt ist es sehr leichtgewichtig und performant. Es bietet viele Erweiterungen, wie z.B. Unterstützung für WebSockets oder Authentifizierung. Außerdem kann es mit Hilfe von GraalVM zu einer nativen Anwendung kompiliert werden, die dann keinen Applikationsserver benötigt. Aber es kann auch traditionell mit Tomcat o.ä. deployt werden.

Spring Boot

Spring Boot ist im Javaumfeld der Industriestandard was Webanwendungen angeht. Mit Java EE bietet es alle erdenklichen Möglichkeiten von Haus aus, wodurch es auch viel Legacy Code und unnötige Features mitbringt. Außerdem hat die ganze Java EE Magie eine größere Lernkurve. Auch benötigt Spring einen Applikationserver.

Daten / Persistenz

Da unser MUD-Creator eher niedrige Nutzerzahlen hat, kann der Spielstand wahrscheinlich während einem Spiel im Hauptspeicher gehalten werden. Deswegen halten sich die Lese- und Schreiboperationen in Grenzen. Gerade beim Spielkonfigurator muss nur beim Speichern der Konfiguration und beim Laden derselben (beim Spielstart oder falls diese editiert wird) gelesen bzw. geschrieben werden. Deshalb würden sich hier einfache Dateien anbieten. Außerdem muss, da die Konfiguration sowieso über eine grafische Nutzungsschnitstelle geändert wird, diese nicht unbedingt von Menschen lesbar sein, weshalb eine einzelne große Datei pro Konfiguration reichen sollte. Der Spielstand muss an sich nur beim Starten eines Spiels oder des Servers gelesen werden, jedoch wird hier wahrscheinlich viel öfter geschrieben (damit beispielsweise beim Absturz des Servers keine Spielstände verloren gehen). Deshalb würde sich hier eine Datenbank anbieten. Diese kann auch für allgemeine Daten, wie z.B. Zugangsdaten genutzt werden. Folgende Paradigmen währen hier möglich:

Der ganze Spielstand wird normal in der Datenbank gespeichert. Mit der traditionellen Methode kann der Spielstand auf einmal einfach gelesen werden, jedoch gestalten sich Schreiboperationen schwieriger, da entweder bei jedem Update (möglicherweise mehrmals pro Sekunde) der ganze Spielstand geschrieben werden muss, oder granulare Änderungen herausgefunden und geschrieben werden müssen. Letzteres bringt vermutlich einen deutlich erhöhten Implementierungsaufwand mit sich. Auch sind hier für Aspekte wie Chat, die einen zeitlichen Verlauf haben, falls diese gespeichert werden sollen, extra Tabellen mit Vorgehensweisen ähnlich wie bei 2. notwendig.

Event Sourcing: Es werden nur Events in die Datenbank geschrieben (z.B. “Spieler hebt Gegenstand x auf”). Dadurch sind Schreiboperationen sehr performant. Beim Lesen werden nun alle Events in ihrer originalen Reihenfolge “abgespielt” und der Spielstand somit schrittweise aufgebaut. Damit nicht jedes Mal die ganze Datenbank gelesen werden muss, können regelmäßig sogenannte Snapshots gespeichert werden, die wie bei 1. den ganzen Spielinhalt enthalten. Beim Lesen müssen die Events dann nur nach dem Snapshot abgespielt werden. Nachteil von dieser Methode ist das erschwerte Suchen, das jedoch bei unserer Anwendung eher irrelevant ist. Jedoch erhält man durch die Events automatisch eine Versionierung des Spielstandes und einen Log.

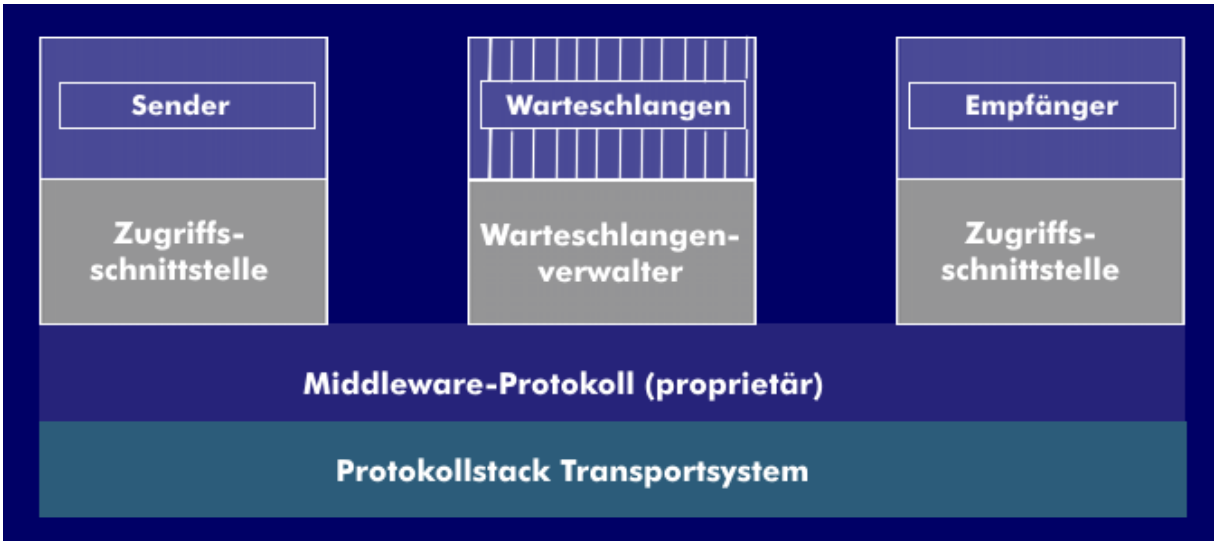
Chat

Technisch gibt es hier zwei Möglichkeiten. Zum einen kann der Chat über eine externe Messaging Software wie RabbitMQ erfolgen, zum anderen kann es über Kommandos wie alle anderen Spielinteraktionen funktionieren. Die erste Methode ist zwar stark skalierbar, bringt aber einen deutlichen Overhead mit sich, da die Software erst gelernt und konfiguriert und später auch gemanaged werden muss. Spieler können im Spiel dann entweder über ein seperates Chatfenster kommunizieren, oder das Ganze läuft über die Hauptkonsole. Ersteres könnte unter der Haube auch Kommandos nutzen, jedoch müsste hier für einen Sitzungsübergreifenden Chatverlauf die Datenbank durchsucht werden, was sich jedoch in beiden Fällen nicht allzu schwer gestalten dürfte.

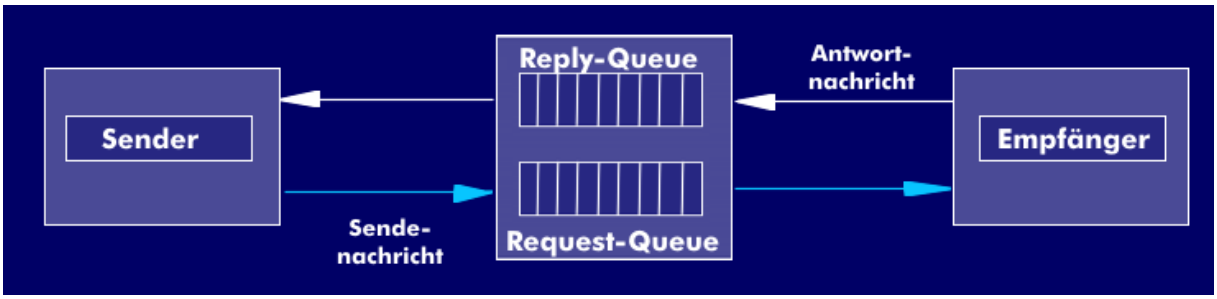
MOM – Message Oriented Middleware

https://www.itwissen.info/MOM-message-oriented-middleware.html

Message Oriented Middleware (MOM) ist die Basis für eine asynchrone Kommunikation zwischen Client und Server in einer verteilten Anwendung. Diese Form der Kommunikation steht im Gegensatz dazu, wenn Client und Server direkt und zeitgleich synchron) miteinander in Verbindung stehen und sich damit ebenso blockieren können. Die Architektur ist wie in der folgenden Grafik dargestellt:



Die Basis ist dabei ein nachrichtenorientiertes Modell, das den asynchronen Austausch von Nachrichten zwischen verschiedenen Prozessen organisiert. In der Regel wird dabei die Technik der Warteschlangen (Queue) eingesetzt. Der Sender stellt eine Nachricht in die Queue des Empfängers, wobei Sender und Empfänger unabhängig voneinander wirken. Der Sender kann weiteragieren, ohne zu wissen, ob die Nachricht angekommen ist, oder nicht. Der Empfänger kann die Nachricht zu einem belieigen Zeitpunkt aus der Queue holen. MOM hat die in der folgenden Abbildung zu sehenden Komponenten:



MOM unterstützt 3 verschiedene Protokolle. Zum einen das Message Passing, was die direkte Kommunikation zwischen Anwendungen bedeutet, zum anderen das Message Queueing (wie zuvor erklärt) und zuletzt das Publish & Subscribe, wobei der Herausgeber dem Abonnenten Nachrichten zur Verfügung stellt. Zu den bekanntesten MOM-Produkten auf dem Markt zählen RabbitMQ, Apache ActiveMQ, Websphere MQ (IBM) und SAP Process Integration.

Client

Hier gibt es folgende zwei Ansätze: Wie bei Evennia den Webclient in der Sprache des Servers schreiben und HTML und JavaScript daraus generieren lassen, oder den Client direkt mit HTML und Java- oder TypeScript (o.ä.) zu schreiben.

Erstere Methode hätte den Vorteil, dass im ganzen Projekt eine Sprache verwendet werden kann. Es gibt Frameworks wie Thymeleaf oder JavaServer Pages, die mit eigenen Templatesprachen arbeiten, jedoch sind diese schon älter und erscheinen unflexibel. Kotlin kann jedoch auch zu JavaScript kompiliert werden und bietet einige Bibliotheken, mit denen man in Kotlin direkt HTML-Seiten definieren kann. Dadurch müsste man hier keine zusätzlichen Templatesprachen erlernen.

Bei der seperaten Programmierung des Clients wird meistens Java- oder TypeScript eingesetzt. Ein Framework, wie Angular oder React erleichtert hier die Arbeit sehr, jedoch muss dieses noch zusätzlich erlernt werden.

Tests

Für Java und Kotlin gibt es eine Vielzahl an Testframeworks, wie z.B. JUnit oder KotlinTest. Für End to End Tests (Testen des Clients) können entweder Clientframeworkspezifische Testframeworks oder das Browser-Testframework Selenium eingesetzt werden. Die meisten Tools sollten mit GitHub CI (oder anderen CI-Tools) funktionieren.

Sonstiges

Java und Kotlin bieten ein großes Ökosystem an Bibliotheken und Ressourcen, die Anforderungen wie Parsen von Kommandos, (De-)Serialisieren von Objekten nach z.B. JSON und die Generierung von Java-/KotlinDocs erleichtern. Als Dependency Management- und Build-Tool kommen Maven oder Gradle in Frage.

Kompetenzen

Die meisten in der Gruppe haben bereits Erfahrung mit Java. Kotlin ist sehr ähnlich und baut auf Java auf, weshalb das Erlernen kein großes Problem darstellen sollte. Was clientseitige Sprachen und Frameworks angeht bestehen nur bei einzelnen Mitgliedern Vorkenntnisse. Auch die Testframeworks und die CI-Tools müssen noch erlernt werden.

Account

Um die Accounts mit bestimmten Spielständen zu verbinden, kann man einen Zusammenhang innerhalb der Datenbank erstellen, welcher einen Account nur die selbst erstellten Charaktere nutzen lässt oder wenn man ein Dungeon Master ist, die selbst erstellten Dungeons bearbeiten lässt.

Wenn sich jedoch für eine Datei für die Spiele entschieden wird, muss sich jedoch für einen komplizierteren Implementationsweg entscheiden. Dies kann mit einer einzigartigen Account-ID gelöst werden, welche dem erstellten Charakter oder Dungeon hinzugefügt wird und beim Beitritt in ein Dungeon überprüft wird, damit der User nur seine eigenen Spielstände nutzen kann.

MUD - Das Spiel

Ein Multi-User-Dungeon (MUD) ist in seiner ursprünglichen Form ein rein textbasiertes Rollenspiel. Heute gibt es auch einige MUDs mit grafischer Oberfläche und vielen anderen Erweiterungen. Im Rahmen der Marktanalyse zum inhaltlichen Themenbereich des Spiels bzw. des Gameplays wurden nur textbasierte Produkte der Konkurrenz miteinbezogen.

Das Thema des Spiels handelt meistens in einer Märchen- oder Fantasy-Welt, so z.B. auch in „Morgengrauen“ (http://www.mud.de/).
Nachdem man dem Spiel beigetreten ist, wird man gebeten, einen Avatar bzw. eine Spielfigur zu wählen. Hier gibt es meist unterschiedliche Rassen und Klassen, die

jeweilige Vor- und Nachteile haben. Im MUD „Morgengrauen“ gibt es als Auswahl z.B. Elfen, Zwerge, Hobbits, Felinen, Orks und Elfen. Als einer dieser Charaktere kann man unterschiedlichen Gilden beitreten. Als Beispiele wäre hier Abenteurer, Wipfelhäufer, Katzenkrieger und Bierschüttler zu nennen.
Im MUD „Ravencroft Academy“ (https://www.ravencroftacademy.com/) gibt es viele Einstellmöglichkeiten bei der Erstellung eines Charakters. Es ist z.B. möglich das Aussehen, die Stimme, Eigenschaften und viele weitere Merkmale einzustellen.
Des Weiteren gibt es unterschiedliche Spielwelten bzw. Karten, die vom Spieler erkundigt werden können.

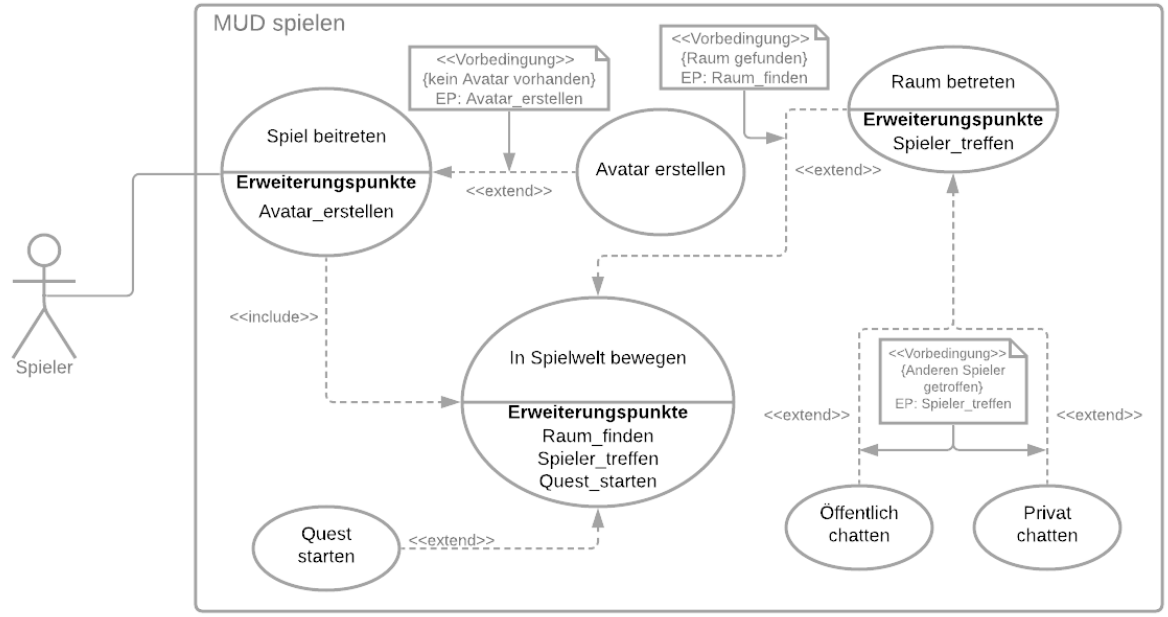
Die Spielfigur besitzt in den untersuchten MUDs eine bestimmte Anzahl an Lebenspunkten, welche schwinden und steigen können. Somit ist ein weiteres, wichtiges Feature mit Objekten aus der Spielwelt zu interagieren. Man kann sich mit bestimmten Objekten also heilen oder anderen Spielern Schaden zufügen.
Objekte werden mit Befehlen, wie z.B. „nimm Apfel“ aufgenommen. Diese werden in einem Inventar gespeichert oder direkt verwendet.
Ein nettes Feature, welches es z.B. in „Morgengrauen“ gibt, ist die reale Uhr. Durch Benachrichtigungen im Spiel wird der Spieler über die aktuelle Uhrzeit informiert, dass lässt das Spiel realer wirken.

Im Verlaufe des Spiels kann der Spieler Quests bzw. Rätsel lösen. Diese haben einen unterschiedlichen Schwierigkeitsgrad und sind somit nicht immer als Einzelspieler lösbar.
Des Weiteren gibt es Minispiele wie Schach, Dame oder Skat, welche der Spieler gegen Computergegner oder andere Spieler spielen kann.

In MUDs gelten verschiedene Spielregeln. Die wichtigsten aus „Morgengrauen“ zusammengefasst: Andere Spieler angreifen oder töten ist untersagt, eine Person darf mehrere Spielercharaktere besitzen, der Spieler darf sich nur eingeschränkt mit Skripten oder Triggern eines Clients unterstützen (Skripte z.B. zum Untersuchen von mehreren Räumen auf Objekte sind untersagt).

Über eine Chatfunktion kann der Spieler sich mit anderen Spielern austauschen.

Im MUD „Ravencroft Academy“ kann beim Erstellen des Charakters noch eingestellt werden, aus welcher Sicht der Spieler gerne spielen möchte. Zur Auswahl gibt es hier first-, second- und third person.



Game-Master

Der Game-Master (oder auch Dungeon Master, Game Operations Director (GOD), Storyteller) ist der Spielorganisator des Spiels.
Er kontrolliert alle wesentlichen Aspekte des Spiels und ist verantwortlich für alle Details und Herausforderungen in Rätseln bzw. Quests. Für die Aktionen anderer Spielercharaktere (PCs) ist er nicht immer verantwortlich. Dies wäre bei großen Spieleranzahlen zu aufwändig.

Das Definieren der Objekte (statisch und aufnehmbar)

Objekte, welche in diesem Fall verwendbare/konsumierbare Objekte behandeln, können in Räumen platziert werden. Ein Objekt kann konfiguriert werden ob es aufgehoben werden kann oder ob es verankert in einem Raum bleibt.
Objekte mit denen interagiert wird, lösen Events aus. Diese Events sind mit den Objekten in der Konfiguration zu erstellen.
Ein Objekt kann einen Preis besitzen, damit dieses mit NPCs gehandelt werden kann. Wenn dieser Preis nicht konfiguriert wird liegt er entweder bei „0“ oder ist nicht handelbar.
Standartobjekte, wie zum Beispiel Heilitems, könnten im Konfigurator Standart-Templates besitzen, da viele MUDs dies benötigen werden. Dies kann dem Ersteller eines MUDs viel Arbeit nehmen und das Erstellen vereinfachen.

Definieren von Ausrüstung

Ausrüstung sind grundsätzlich wie Objekte aufgebaut. Diese können jedoch aus dem Inventar und auch aus der Welt „ausgerüstet“ werden. Jeder Ausrüstung müssen jedoch weitere Stats zugewiesen werden, damit diese in einem möglichen Kampfsystem einen Einfluss haben.
Ausrüstung wie Waffen sollten in Werten wie „Schaden“ konfigurierbar sein, während Rüstungen einen Rüstungs- oder HP-Wert besitzen sollten.
Spezielle Ausrüstung sollte eventuell unter bestimmten Voraussetzungen, wie die normalen Objekte in der Lage sein Events aufzurufen.

Definieren von NPCs

Diese NPCs sollen „freundliche“ NPCs darstellen mit denen man interagieren kann. Diese NPCs sollten konfigurierbar sein in Hinsicht der Namen, NPC-Typ (beispielsweise Händler) und in den Dialogoptionen.
Der NPC-Typ soll bestimmen auf welche Systeme der NPC Zugriff hat. Ein Händler-NPC sollte in der Lage sein, das Handelssystem aufzurufen. Ein Heilungs-NPC sollte in der Lage sein den Spieler zu heilen, also auf das Statsystem zuzugreifen. Dies sollte über ein Typensystem konfiguriert werden.
NPCs haben meist auch Dialog, somit müssen Dialogoptionen und die jeweiligen Antworten konfigurierbar sein. Dies kann später eventuell in Kombination mit Dialogbäumen ausgeweitet werden.
NPCs sollten in der Lage sein, Events aufzurufen, welche in der Konfiguration an Dialogantworten gebunden werden.

Definieren von Gegner NPCs

Diese Gegner NPCs sollen feindliche NPCs darstellen, welche der Spieler im Verlauf des Dungeons in Räumen antreffen kann und bekämpfen kann. Einem Gegner NPC müssen für das Kampfsystem ein „Verhalten“ konfiguriert werden, d.h. der Gegner ist aggressiv und benutzt Angriff X häufig, oder ist defensiv und heilt sich mit Aktion Y häufig.
Dies bedeutet, dass man Gegner NPCs Aktionen in Kampfsituationen definieren muss, beispielsweise die oben genannten beim Verhalten.
Ein Gegner benötigt zur Berechnung im Kampfsystem ähnliche Stats wie der Spielcharakter des Users. Somit sind Werte wie Health Points, Schaden und Ähnliches auch zu konfigurieren.
Ein Gegner in einem typischen Dungeonsetting, ob es ein MUD oder andere Spiele seien, sollte Objekte bei seinem Tod fallen lassen. Diese Objekte werden in der Konfiguration in einem Loottable mit Dropchancen versehen, somit kann beispielsweise beim Tod des „Orkkönigs“ mit einer 50%-igen Chance die „Orkkrone“ als Objekt/Ausrüstung fallen gelassen werden.
Falls ein Levelsystem implementiert werden sollte, sollte jedem Gegner ein XP-Wert

zugewiesen werden, der dem Spieler gutgeschrieben wird, wenn der Gegner besiegt wird.

Definieren der Events

Die Events dienen als Automatisierungsmöglichkeit, damit der MUD-Master nicht jeden Schritt eines jeden Spielers beobachten muss und agieren muss. Die Events werden in der Form konfiguriert, dass ein „Ziel“ ausgewählt wird und eine Aktion am Inhalt, dem Status oder anderen Aspekten des Ziels ausgeführt wird.
Beispiel: Spieler Y trinkt einen Trank der Heilung, Event X wird ausgeführt auf Spieler Y. Dieses Event besagt, dass die HP des Spieler Y um 100 erhöht wird, denn Event X ist ein Heilungsevent. Jedoch hätte auch mit einer 5% Chance dieses Event zum Tod des Spielers führen könne.
Somit werden in Events Chancen, Werte und Aktionen definiert und konfiguriert. Standart-Template-Events sollten vorhanden sein, wie zum Beispiel ein simples Heilevent, damit der Ersteller nicht erstmal ein Event in Form von „Spieler X kriegt seine HP um Z inkrementiert“ erstellt werden muss.

All diese Aspekte akkumuliert führen dazu, dass der Ersteller des MUDs sehr viel einstellen kann und ein spielbares MUD erzeugt wird.
Viele der im Vorfeld konfigurierten Einstellungen, können im Spielverlauf verändert werden (Bsp. HP eines Spielers). Gegnerdefinitionen, Ausrüstungsdefinitionen und ähnliches sind nicht während des Spielflusses zu verändern.

MUD - Spieloberfläche

Betrachtets man die ersten MUDs in ihrer Grundform, so waren dessen Spieloberflächen nur reiner Text ohne jegliche zusätzlichen Dinge wie Bilder. Betrachtet man dessen Design, so könnte man es am besten mit dem einer alten DOS-Konsole vergleichen. Das heißt, um das Spiel zu spielen hat lediglich ein Konsolenfenster und eine Telnet Verbindung ausgereicht. Jegliches Spielrelevante wurde somit in der Konsole selbst angezeigt bzw. dargestellt.

Als Beispiel wäre das „Colossal Cave Adventure“. Erstere Versionen des MUDs liefen beispielsweiße auf einem Osborne 1 Computer mit Grüner Konsolenschrift und dunklem Hintergrund. Mehr gab es dabei nicht. Erst spätere Versionen des Spiels haben dann angefangen auf MS-DOS anstatt nur Text auch noch grafisch Bilder mit auszugeben. Diese Bilder waren zwar recht simpel im Vergleich zu heutigen Möglichkeiten, konnten aber dazu beitragen, dass ein Spieler tiefer in die Spielwelt eintauchen kann und sich gegebenenfalls auch besser in dieser orientieren kann. Auch heute noch werden MUDs auch nur mit Text angeboten. Eine Vielzahl der aktuellen MUDs greift jedoch beim Darstellen der Spieloberfläche auch auf weitere Elemente zurück bzw. nutzen gar einen eigenen Client, welcher der Spieler herunterladen kann. Typischer ist jedoch die Darstellung über eine Website. Einige Anbieter bieten mittlerweile sogar alle Möglichkeiten auf einmal an. So kann dem

Nutzer komplette Entscheidungsfreiheit gegeben werden, wie er das Spiel spielen will. Anstatt nur Text lässt sich nun mittlerweile Dinge wie eine Lebensanzeige, Statistiken, Attribute des eigenen Charakters, einen separaten Chat und einiges mehr darstellen. Um die Relevanz zu unserem Produkt zu behalten beziehen wir uns im Folgenden jedoch hauptsächlich um die Darstellung aktueller MUDs welche über eine Website dargestellt und spielbar sind.

Als Beispiel nutzen wir in diesem Falle das MUD „Morgengrauen“, da dieses die Möglichkeit bietet sowohl rein textbasiert als auch mit einer grafisch komplexeren Darstellung das gleiche Spiel zu spielen. Dieses MUD ist auch heute noch aktiv sowie bekannt in der MUD-Szene. Da wir uns nicht auf Clients, sondern nur auf Web basierte Lösungen interessieren, gehen wir nur auf diese näher ein. Es gibt dazu drei verschiedene Auswahlmöglichkeiten. Diese sind Web Telnet (einfach), Web Telnet (sicher) und Web Telnet (komplex). Da sich einfach und sicher nur in der Übertragungsart unterscheiden, fassen wir diese Spieloberfläche zu einer zusammen und gehen nicht getrennt auf beide ein.

Betrachten wir nun die Web Telnet (einfach/sichere) Spieloberfläche des MUDs genauer. Diese funktioniert wie die, der ersten MUDs. Dort gibt es keine grafischen Inhalte wie Objekte oder Bilder. Alles wird über eine recht neutral aussehende „Konsole“ gesteuert, welche eine Textzeile als Eingabefeld besitzt. Jeglicher Spielinhalt wird anhand von Text dem Spieler über die Konsole angezeigt. So bekommt dieser darüber jegliche Nachrichten, kann darüber navigieren, sich umschauen und vieles mehr. Hierfür muss der Spieler jeweils einen bestimmten Befehl in die Zeile schreiben und diesen abschicken. Daraufhin bekommt er dann dementsprechend eine „Antwort“ vom Server mit den neuen Spielrelevanten Infos bzw. Inhalten. Diese Art ist vor allem für Anfänger anfangs schwerer, da diese jegliche Commands nicht auswendig können und dadurch zumindest anfangs definitiv länger benötigen um sich im Spiel zurechtzufinden.

Betrachten wir nun aber die Web Telnet (komplex) Spieloberfläche, so merkt man direkt, dass diese nicht nur aus einer simplen „Konsole“ besteht. Ergänzend zur Konsole sowie der Eingabezeile gibt es nun dazu auch noch diverse Buttons und extra Fenster. So wird die Möglichkeit geboten anstatt „Enter“ einen extra Senden Button zu nutzen sowie häufig verwendete (prädefinierte) Befehle direkt über mehrere Extra Buttons über der Eingabezeile anhand von nur einem Klick zu betätigen. Um sich in der Spielwelt zu bewegen und allgemein mit dieser zu interagieren, gibt es neben der „Konsole“ zusätzliche Buttons. Diese ermöglichen eine Bewegung in alle Himmelsrichtungen, das Umschauen im Raum sowie zum Beispiel das Verlassen eines Raumes. Zusätzlich gibt es ein extra Fenster für Mitteilungen, in welchem Private Nachrichten angezeigt werden sowie ein Log des „Ebenen Chat“. Dies ist vor allem sehr praktisch, um auch nach längerer Zeit einen Überblick über ältere Nachrichten zu haben, da diese in der Konsole sehr schnell untergehen. Hierbei gibt es auch noch die Möglichkeit über weitere Buttons sich weitere Fenster anzeigen zu lassen, Makros auszuführen oder sich auszuloggen. Unter diesen weiteren Fenstern befinden sich bspw. welche zur Anzeige des Inventars, der Ausrüstung, Info über den Spieler, Ort, Kampf und einiges mehr. Öffnet man mehrere dieser so wirkt das ganze unübersichtlich. Besser wäre es vermutlich gewesen diese fest in die Spieloberfläche zu implementieren, damit das Ganze eine bessere Struktur besitzt.