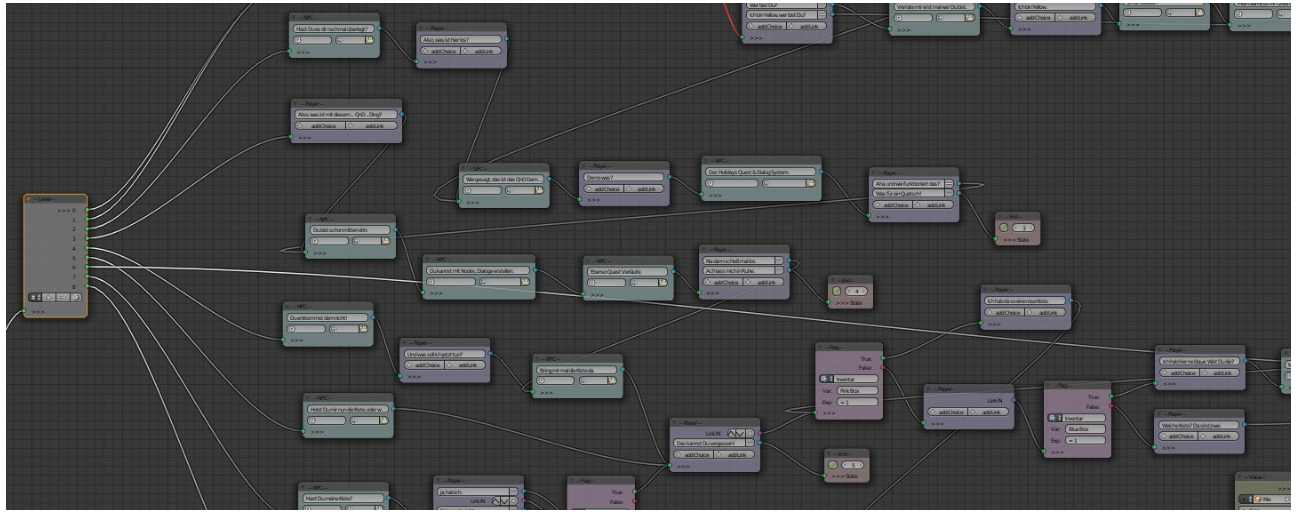
QnD – Quest und Dialog System

Dokumentation der Version 1.0 (Doc Holiday 2015)



QnD ist ein Node basierendes, Quest und Dialog System für Blender GE.

Es ermöglicht sehr einfach dynamische Dialoge und Quest z.B. für RPGs zu erstellen. Durch lesen und schreiben von Objekt-Properties und globalen Variablen kann das System auf Ereignisse reagieren, als auch welche auslösen.

Funktionsweise

Das System besteht zunächst aus 2 Teilen, welche Bestandteile des Paketes sind:

1. Das AddOn selbst, welches ein neues Node System in Blender erzeugt
2. Das „QnD_Mailer.py“-Script, welches die logischen Abläufe abarbeitet

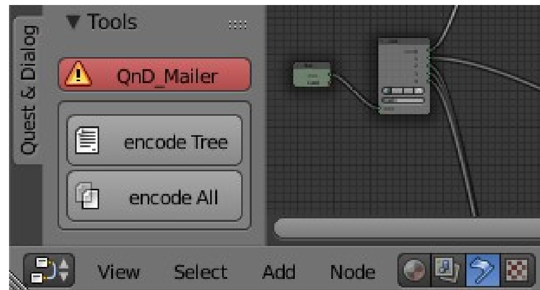
Über das Node System werden die Dialoge erstellt, und in eine Text-Form umgewandelt, die „QnD_DataBase.py“. Einzig diese Database wird vom „QnD_Mailer.py “-Script verarbeitet. Das Node System ist für den Ablauf nicht nötig.

Das „QnD_Mailer.py“-Script wird komplett über Messages angesteuert. Ebenso gibt es Messages aus, die von einem Empfänger verarbeitet werden müssen. z.B. Befehle und Text für den NPC, mögliche Antworten usw. Um letztlich mit dem Spieler zu interagieren ist noch eine GUI oder Entsprechendes nötig.

Dieser Empfänger sowie die GUI sind nicht Teil des QnD. Sie stellen sozusagen die Schnittstellen zum individuellen Projekt dar, und müssen daher vom Entwickler erstellt werden. Das Demo File enthält ein einfaches Beispiel wie es funktionieren kann. Wer will kann es als Vorlage verwenden. Ich werde in dieser Dokumentation auch gelegentlich Bezug auf diese Demo nehmen.

> Um Fehlverhalten zu vermeiden, sollte man sich vergewissern das die Objekte die man in QnD verwendet einzigartig sind, und nicht mehrfach vorhanden sein können.

Installation

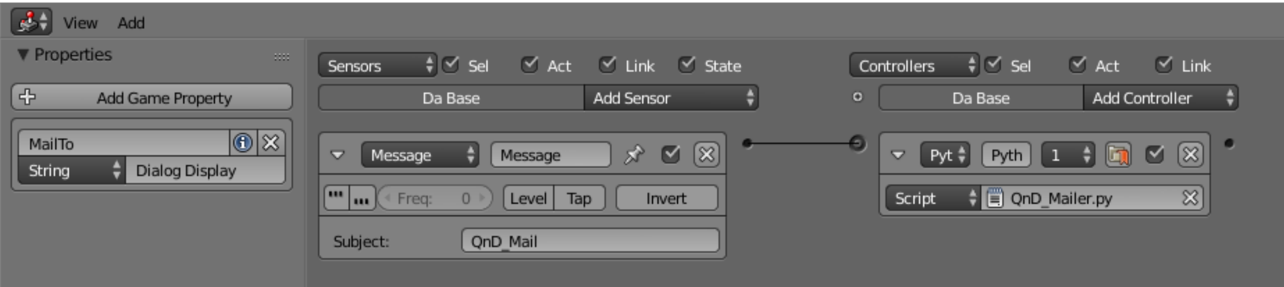


Das AddOn wird als ZIP-Paket wie üblich über die >User Pereferences / Add-ons / Install from File< installiert und aktiviert. Im Node-Editor steht ab sofort ein neues Node System zur Verfügung.

In der Toolbar (T-Key) befinden sich neben den Buttos für die Nodes, noch weitere Funktionen. Der rot unterlegte Button weist auf das fehlende „QnD_Mailer.py“ Script hin. Ein Klick darauf läd es aus dem AddOn Bereich in das blend.

Aufbau

Das „QnD_Mailer.py“ Script ist das Kernstück der Echtzeit-Komponente. Es kann beliebig in der Scene platziert werden. Durch die Messages wird es überall erreicht. In der Demo hängt es z.B. an einem Empty names „Da Base“.



Es benötigt lediglich einen Message-Sensor, der auf das Subject „QnD_Mail“ filtert. Dieses Subject wird von allen Messages verwendet, die in das System eingehen.

Das String-Property „MailTo“ ist optional. Es beinhaltet den Namen des Objekts, welches für die Verarbeitung ausgehender QnD Messages zuständig ist. Dieses Objekt wird so direkt „angefunkt“. Ohne dies gehen die Messages an alle Objekte.

Hat man einen Dialog, oder eine Quests Tree im Node Editor erstellt, muss dieser über die „encode-Butons“ in eine Text Form umgewandelt werden. „Tree“ verarbeitet nur den aktuellen, „All“ alle vorhandenen. Dabei wird die „QnD_DataBase.py“ erzeugt, oder ggf. aktualisiert.

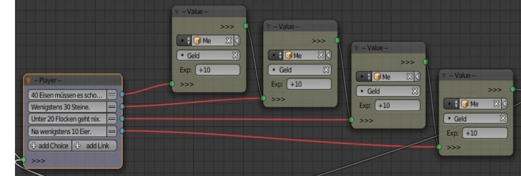
Die Namen der Trees werden sowohl für die DataBase, als auch für den Ablauf im Spiel verwendet. Es ist ratsam eindeutige Bezeichnungen zu verwenden.

Um einen NPC als „ansprechbar“ zu kennzeichnen, erhält dieser einfach ein String-Property mit der Bezeichnung „QnD“, und als Inhalt den Namen des Trees. z.B. MrGeen:

(Ich habe meine Blender Source etwas umgeschrieben um Game Properties übersichtlicher zu gestalten.)

Nodes

Um den weiteren Ablauf zu erklären, kann ich gleich zu den Funktionen der einzelnen Nodes übergehen.

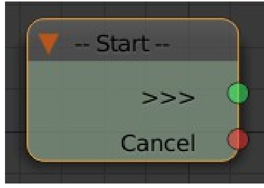


Das neue Node System unterscheidet sich im Handling nicht von anderen Nodes in Blender. Allerdings sind LoopBacks hier kein Fehler, sondern sogar gewünscht. Blender markiert diese Strips rot. Das kann man aber getrost ignorieren.

- Ein Dialog beginnt mit der Start-Node, und endet erst mit einer End-Node.
- Es darf kein Socket leer bleiben. (Ausgenommen Cancel und Link INs.)
- Änderungen am Tree müssen mit encode in die DataBase übernommen werden um im Spiel zu funktionieren.

Selbstverständlich darf man auch hier speziellen Nodes eigene Farben verpassen. z.B. Wenn ein Quest Status angehoben wird, oder andere wichtige Dinge. So findet man sich recht schnell auch in großen Trees wieder zurecht.

** Start Node*



Diese Node markiert den Beginn einen Trees, und darf nur 1x vor kommen. Das bedeutet nur 1 Tree pro Slot ist erlaubt. Ein Dialog wird gestartet indem eine Message an den QnD_Mailer gesendet wird. Diese hat folgendes Format:

Subject:	„QnD_Mail“
Body:	„Start:“ + Name oder ID des NPC + “|“ + Name oder ID des Players
To:	(optional) an alle wenn leer

Body ist ein String. z.B.: „Start:MrGreen|Me_Body“. Das Mailer Script zerlegt diesen in einzelne Anweisungen. Alternativ zum Namen der Objekte kann auch deren ID gesendet werden. Diese ermögliche zwar eine eindeutige Zuordnung, ist aber nur mit Python zu bekommen. Wird der Dialog beendet, erfolgt eine Wiederaufnahme auf die gleiche Weise.

Mit dem Eintreffen einer Start-Message wird der Tree ab der Start-Node in Richtung „>>>“ abgearbeitet.

Abbrechen lässt sich der Dialog mit einer Message im folgenden Format:

Subject:	„QnD_Mail“
Body:	„Cancel:“
To:	(optional) an alle wenn leer

Beim deren Eintreffen wird der Cancel Zweig der Start-Node abgearbeitet. Dieser Slot braucht kein End-Node, es sei denn es sind noch weiteren Nodes daran angeschlossen.

Während eines Dialoges ist am QnD_Mailer ein Bool Property Namens: „IsTalking“ auf True gesetzt. Dieses kann man z.B. verwenden, um den Spieler zwischenzeitlich bewegungsunfähig zu machen, damit der Mauszeiger für die GUI frei wird.

* End Node

Nun, man kann es sich denken, diese Node beendet den Dialog.
Dieses Ende bedeutet aber nur, daß man für die Weiterführung erneut eine Start-Message senden, also erneut auslösen muss.

Man kann beim Verlassen ein sog. State setzen. Das repräsentiert sozusagen einen Einsprungpunkt, wo das Gespräch weiter geführt werden soll. Setzt man den Haken, bleibt der momentan anliegende State beim Beenden unverändert. Das ist z.B. sinnvoll wenn ein Zweig mehrfach genutzt, und damit mit unterschiedlichen States beendet werden kann.

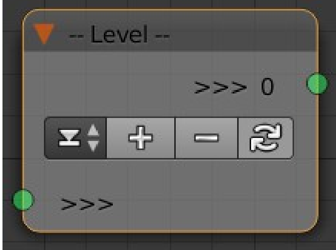
Zu Beginn eines Dialoges ist der State immer 0. QnD speichert die Verläufe sämtlicher Dialoge in der GlobalDict, wodurch diese in ein SaveGame mit einfließen.

Das o.g. „IsTalking“ Property wird wieder auf False gesetzt. Zudem wird ein End-Message diesen Formats gesendet.

Subject:	„QnD_Display“
Body:	„End|“ + ID des Players + “|“ + State (-1 bedeutet unverändert)
To:	(optional, siehe „MailTo“ Property)

(Messages transportieren auch numerische Werte nur als Strings.)

* Level Node



Eine einfache Verzweiger Node. Je nach dem welches Wert anliegt (default ist State), wird entsprechend verzweigt. Es wird der höchste, bzw. niedrigste Zweig genommen, sollte der Wert außerhalb der verfügbaren Sockets liegen.
Ist die Cycle-Funktion eingeschaltet, wird bei unpassenden Werten einfach vom anderen Ende aus weiter gezählt.

Es können verschiedene Typen eingestellt werden. Deren Inhalte sollten Integer sein.

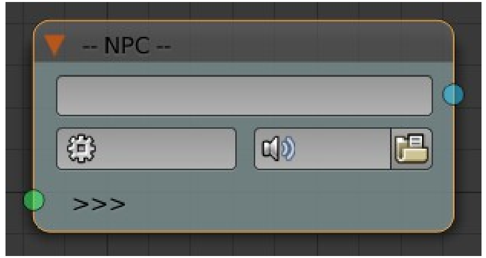
- Random – der nächste Zweig wird zufällig gewählt
- State – der anliegende State
- Global Dict – eine Variable im Global Dictionary
- Global Sub Dict – eine Variable in einem Dictionary des Global Dictionary
- Object Property – das Game Property eines Objects

Einige dieser Typen finden sich auch in anderen Nodes wieder. Im Global Dictionary der Game Engine kann man wiederum Dictionarys ablegen. z.B. für das Inventar u.a. Daher die beiden Typen. Der Python Zugriff auf „Global Dict“ und „Global Sub Dict“ erfolgt so:

```
VarGlobalDict = bge.logic.globalDict[„VariableName“]
VarGlobalSubDict = bge.logic.globalDict[„DictionaryName“][„VariableName“]
```

Um für das Property ein Objekt auszuwählen, kann man entweder eines in der Liste suchen, oder über den Button, das momentan im 3D View selektierte verwenden.

* NPC Node



Diese Node enthält die Ausgabe für den NPC. (Nonplayer Character) Neben dem Text gibt es noch ein Feld für spezielle Befehle. Diese können vom Empfänger verarbeitet werden um z.B. Animationen abzuspielen, eine spezielle Anzeige zu verwenden, Anzeigedauer, uvm. Dafür gibt es keine Spezifikation. Solange der Empfänger weiß was er damit anstellen soll, kann man da rein schreiben was man will. In der Demo habe ich es z.B. verwendet um eine Anzeigedauer, sowie eine Pause zu erstellen.

Ein weiterer Bereich ist für ein Audio-File reserviert. Evtl. möchte man Sprachausgabe realisieren, dann muß der NPC wissen was es wann zu sagen hat. Dies Feld beinhaltet aber nur den DateiNamen. Es existiert keine Abspielfunktion. Dies muss auch wieder vom Empfänger verarbeitet und erledigt werden.

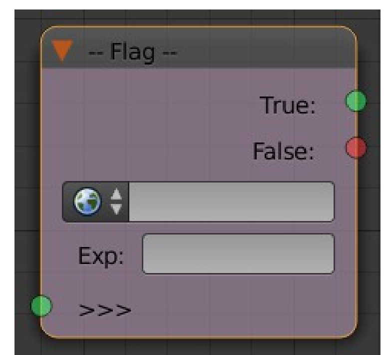
Die Daten werden wieder in Form einer Message gesendet:

Subject:	„QnD_Display“
Body:	„ToNPC|“ + Textfeld + “|“ + Special + “|“ + AudioFile
To:	(optional, siehe „MailTo“ Property)

Nachdem diese Message gesendet wurde, erwartet der Mailer eine Antwort, wann mit der nächsten Node weiter gemacht werden kann. Schließlich muß der Spieler erst mal lesen, bzw. hören was der NPC zu sagen hat. Ist man soweit, sendet man eine Message zurück:

Subject:	„QnD_Mail“
Body:	„Next:“
To:	(optional) an alle wenn leer

* Flag Node

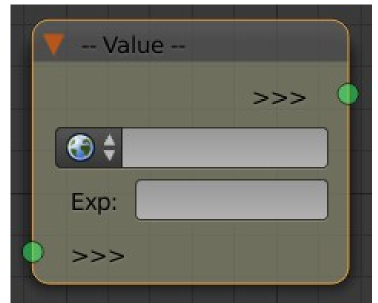


Über diese Node ist es möglich auf Ereignisse und Zustände des laufenden Spiels zu reagieren, und entsprechend zu verzweigen. Man kann auf diese Weise z.B. abfragen ob der Player ein bestimmtes Item im Inventar hat, wie weit er mit einer Quest ist, ob er ein bestimmtes Areal betreten hat, uvm.

Man gibt die Variable an die geprüft werden soll, darunter einen Ausdruck. Per default ist Global Dict gewählt. Also z.B. Quest1 =2 bedeutet: Hat die Variable „Quest1“ im Global Dict den Wert 2 wird nach True verzweigt, anderenfalls nach False. Existiert die Variable noch gar nicht, ist das Ergebnis ebenso False. Im Ausdruck gehen natürlich noch andere Operatoren wie: >= != < > usw.

Da Flag schon als Abfrage gilt, ist das doppelte Gleichheitszeichen „==“ nicht nötig.

* Value Node



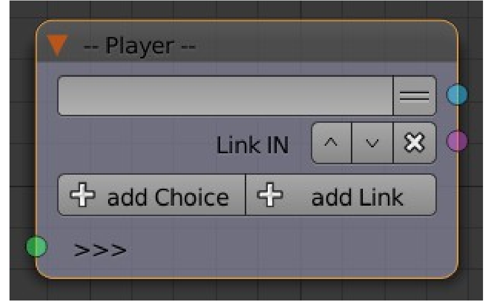
Mit dieser Node weist man Variablen oder Properties einen Wert zu. Es ist so auch möglich States zu setzen, ohne den Dialog zu beenden.

Die Zuweisung erfolgt auch hier über einen Ausdruck.
Dies mal bedeutet Quest1 =2, das die Variable „Quest1“ den Wert 2 erhält.
Auch solche Ausdrücke sind erlaubt: Quest1 +1. So wird sie z.B. um einen Wert erhöht. Existiert die Variable dagegen noch gar nicht, wird sie mit dem Wert 0 erzeugt und sofort verrechnet. Beim ersten Durchlauf der Node hätte die Variable „Quest1“ also den Wert 1.

Beim Verwenden von Strings, sind diese in Anführungszeichen zu setzen. „“

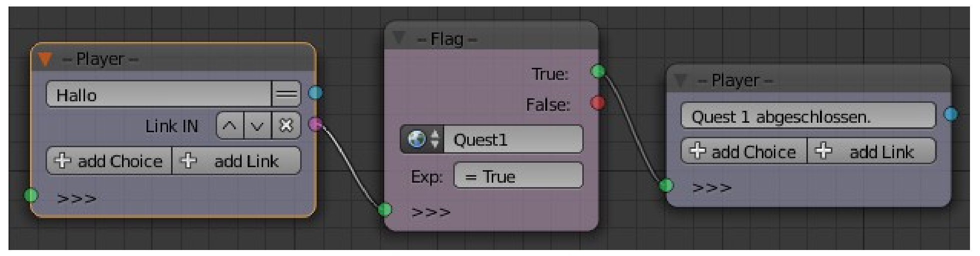
Werden Flag.- und Value Nodes durchlaufen, aktualisiert sind der Joker für ($Flag) und ($Val). Diese Joker finden in Texten Verwendung, wo sie später durch ihren Wert ersetzt werden. Sie enthalten immer die Variable die zuletzt behandelt wurde.

* Player Node



Eine Reihe von Auswahlmöglichkeiten wird über diese Node gesendet. Die Reihenfolge der Texte ● und Links ● läßt sich beliebig verändern. Für die spätere Auswertung beginnt die oberste Zeile bei Position 0.

Ein Link IN ist eine Platzhalter Zeile. Ein solcher Socket erwartet als nächste Node eine Flag oder eine Level Node, gefolgt von mindestens einer weiteren Player Node. Ob der Text aus dieser weiteren Player Node gezeigt wird hängt davon ab ob das Flag oder der Level dies zulassen. Das hört sich jetzt vielleicht komplizierter an als es ist.



(Die zweite Zeile wird angezeigt, oder auch nicht.)

Je nachdem ob das Flag True oder False wertet, wird die entsprechende Player Node in den Platzhalter eingepflegt. Hier muss nicht zwingend an beiden Sockets eine Node hängen. Ist am treffenden Socket keine weitere Player Node angeschlossen, wird die Zeile einfach übersprungen.
Verwendet man eine Level Node, wird die Node am Socket eingebunden welcher dem Wert der verwendeten Variable entspricht.
Das Prozedere wird so lange wiederholt, bis alle Link INs in der Kette ausgewertet sind. Es ist kein Problem wenn die erste Node komplett aus Links besteht. Sollte sie in irgend einer Situation komplett leer sein, wird ein End ohne State ausgelöst.

Die ausgehende Message sieht folgendermaßen aus:

Subject:	„QnD_Display“
Body:	„ToPly|“ + Antwort 1 + “|“ + Antwort 2 + “|“ + Antwort 3 + “|“ + usw.
To:	(optional, siehe „MailTo“ Property)

Auch nach einer Player Node erwartet der Mailer eine Antwort. Welche der möglichen Antworten hat der Spieler gewählt? Wiederum geschieht das durch eine Message:

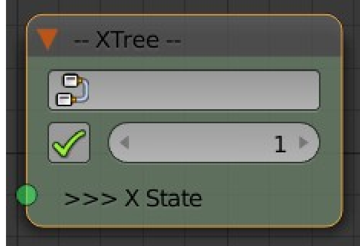
Subject:	„QnD_Mail“
Body:	„Select:“ + Nummer der Auswahl
To:	(optional) an alle wenn leer

Die Nummer ist einfach der Platz in der Liste, welche von der Player Node gesendet wurde. Wie man das löst hängt jetzt davon ab, wie man den Dialog zum Spieler aufgebaut hat.

Man kann sich nochmal die Demo anschauen. Ich habe z.B. jeder Auswahlzeile schon bei der Erzeugung ihren Message-Body in ein String Property übergeben. Die ausgewählte Zeile sendet so einfach selbst das String.

** Xtree Node*

Kurz und bündig, mit dieser Node kann man einen neuen Tree an den momentanen NPC anlegen, und beim angegebenen State hinein springen.



Ende

Wie die Messages an das „QnD_Mailer.py“ Script zustande kommen ist unwichtig. In der Demo verwende ich teilweise Python und Logic Bricks. Sie müssen auch nicht zwingend vom Player aus gehen, sondern können von überall her kommen. Das eröffnet vielfältige Möglichkeiten.

Ich hoffe das AddOn unterstützt euch bei euren Projekten, und wünsche viel Spaß beim Basteln und Questen.

Doc