HOCHSCHULE **TRIER**
Trier University of Applied Sciences
Informatik - Computer Science

Zielgerichtete Künstliche Intelligenz für Storytelling in Computerspielen

Goal Oriented Artificial Intelligence for Storytelling in Video Games

Jonas F. Luft

Bachelor Abschlussarbeit

Betreuer: Prof. Dr. Christof Rezk-Salama

Trier, 15.08.2014

Kurzfassung

Die größte Stärke des Mediums Computerspiel – die Möglichkeit der Interaktion – wurde bisher noch nicht ausreichend genutzt, um dem Spieler Einfluss auf das Storytelling zu geben. Diese Arbeit entwickelt eine Möglichkeit, das Storytelling in Spielen interessanter zu gestalten.

Als ein möglicher Lösungsansatz wird hier ein System vorgeschlagen, das Entwicklern von Spielen die Möglichkeit gibt, mit programmiertechnisch geringem Aufwand ein Framework zu erstellen, mit dem die Autoren problemlos interessante, vielseitige und interaktive Stories schreiben, aber dabei trotzdem die ihnen vertrauten dramaturgischen Techniken anwenden können.

Das Ziel ist es, Story auf eine Weise zu präsentieren, so dass der Spieler bedeutungsvolle Entscheidungen treffen kann, ohne dabei die narrative Integrität der Handlung zu gefährden.

Die Idee ist, die Handlung in die atomaren Einzelschritte zu zerlegen, welche die Handlung vorantreiben. In Kombination mit der Möglichkeit der Interaktion zu jedem Ereignis werden variable Handlungsverläufe ermöglicht. Um ein abwechslungsreiches Spielerlebnis zu bieten, wird GOAP, ein Planungsverfahren der künstlichen Intelligenz, genutzt, um aus den einzelnen Ereignissen unter Berücksichtigung kausaler Zusammenhänge eine Sequenz zu erstellen, die ein logisches Voranschreiten der Handlung gewährleistet.

So far the greatest potential of videogames – interactivity – hasn't been sufficiently utilised to grant players influence over storytelling. This thesis develops a way to make storytelling in games more interesing.

As a possible approach a system is proposed that gives game developers the possibility to create a framework with which authors can write diverse and interactive stories while being able to use their familiar dramatic techniques. The aim is to present story in a way that enables the player to make meaningful decisions without compromising the narrative integrity.

The idea is to divide the plot into its atomic components that constitute plot progress. Combined with the possibility to make choices at each event it is possible to achieve variable storylines. GOAP, an AI planning algorithm, is used to create a sequence of the individual events while logical progresseion of plot is maintained.

Inhaltsverzeichnis

1 Einleitung 1
- 1.1 Case study 1
- 1.2 Begriffsbestimmung 6
- 1.3 Zielsetzung 7

2 Lösungsmodell 9
- 2.1 Entwurf 9
  - 2.1.1 Idee 9
  - 2.1.2 Pfadplanung 11
- 2.2 Spezifikation 12
  - 2.2.1 Definition 12
  - 2.2.2 Verfahren 13
  - 2.2.3 Beispiel 13
- 2.3 Funktionsweise 14
  - 2.3.1 GOAP 14
  - 2.3.2 Transfer 19

3 Implementation 22

4 Ergebnisse 25
- 4.1 Effekte 25
- 4.2 Verwendung 26
- 4.3 Einschränkungen 27

5 Ausblick 29

Literaturverzeichnis 31

Erklärung des Kandidaten 33

1

Einleitung

1.1 Case study

Videospiele im Allgemeinen und deren Storytelling im Speziellen sind bisher selten in den Genuss wissenschaftlicher Untersuchungen gekommen (vor allem weil sie noch nicht lange in der breiten Öffentlichkeit Beachtung gefunden haben). Daher kann nicht viel auf bestehende akademische Arbeiten zurückgegriffen werden. Aus diesem Grund muss unter anderem auch auf die Erfahrungen und Terminologie anderer Fachgebiete Bezug genommen werden (siehe dazu 1.2).

Zunächst werden bisherige Herangehensweisen von Spielen an das Storytelling untersucht, um zu bestimmen, welche Unzulänglichkeiten diese Ansätze besitzen, aber auch worauf aufgebaut werden kann. Zusätzlich werden Beispiele aus anderen Medien hinzugezogen, um Erkenntnisse zu erlangen, wie aus der Entwicklung dieser Bereiche Nutzen gezogen werden kann und welche Probleme entstehen, wenn diese für Spiele Anwendung finden sollen.

Während Videospiele seit ihrer Entstehung, wie etwa Pong ([Pon72]), gänzlich ohne Story auskommen können, haben sie sich mittlerweile zu einer Medienform entwickelt, die sich hervorragend dazu eignet, Geschichten auf eine höchst immersive Weise zu vermitteln. Story ist ein maßgeblicher Bestandteil der Qualität eines Computerspiels geworden, auch wenn selbst innerhalb der Branche Uneinigkeit darüber herrscht, wie wichtig sie tatsächlich ist, wie sich in folgendem Zitat von John Carmack widerspiegelt: „Story in a game is like a story in a porn movie. It's expected to be there, but it's not that important." ([Kus03]). Dies und die fehlende Bereitschaft, Spiele als Literaturform anzusehen, liegt vermutlich daran, dass die bisherige Einbindung von Story in Spielen bisher nicht das volle Potential und die besonderen Stärken des Mediums nutzt.

Simultan zu den reinen Geschicklichkeitsspielen haben sich Spiele entwickelt, die sich fast vollständig auf Story für den Unterhaltungseffekt verlassen. Dies begann mit dem Spiel Adventure ([Cro76]), das auf den Erfolg klassischer Pen & Paper-Rollenspiele (P&P), insbesondere Dungeons and Dragons (D&D [GA74]), aufbaut. Adventure ist der Namensgeber für das resultierende Genre von Adventure-Games, das nach den ersten rein textbasierten Vertretern im Zuge der technischen Entwicklung der Hardware zunehmend auch grafische Spiele, wie etwa Rogue ([TW80]) oder später Myst([mys93]), einschloss.

Die erfolgreichsten Vertreter der modernen Spiele können als ein Produkt beider Entwicklungen verstanden werden, die sowohl viel Geschicklichkeit (oder analytische Fähigkeiten) voraussetzen, als auch versuchen, eine interessante und fesselnde Geschichte zu erzählen.

Interactivised Movies

Das Storytelling in Spielen orientiert sich bisher an traditionellen Medien, wie Theater oder Film und greift auf die dort entwickelten und bewährten Kunstgriffe zurück. Daher bezeichnet Chris Crawford Spiele, die sich eher wie ein Film mit aufgepflanztem Gameplay anfühlen als „interactivised movies“ ([Cra12]).

Beispiele dafür lassen sich in jedem Genre moderner Spiele finden: von First Person Shooter (FPS, wie Gears of War [gow06]) über Massively Multiplayer Online (MMO, wie World of Warcraft [wow05]) zu Role Playing Games (RPG, wie Final Fantasy [ff87]). Dabei unterscheidet sich nur, wo der Schwerpunkt beim jeweiligen Genre gesetzt wird (Action geladene Spiele und Strategiespiele setzen naturgemäß eher auf die Meisterschaft des Spielers, Adventures und RPG mehr auf Story und das Erleben einer lebendigen Welt). Aber auch Spiele, die sich vollständig darauf verlassen, Spannung durch Action zu vermitteln (zum Beispiel Call of Duty [cod03]), scheinen sich verpflichtet zu fühlen, eine Handlung zu besitzen, präsentieren diese aber nur in kleinen Portionen abwechselnd mit Action-geladenen Szenen ohne jegliche nachvollziehbare Handlung. Eine langweilige oder gar schlechte narrative Komponente wird häufig mit einem starken Fokus auf Multiplayer-Funktionalität versucht wieder wettzumachen. Wie eine gezwungener Versuch, Story einzubringen, ohne dabei auf das Medium einzugehen, dazu führen kann, dass ein Spiel für manche Spieler unerträglich werden kann, zeigt das Extrembeispiel Metal Gear 4 ([mg408]). Metal Gear ist ein FPS, das durch die Notwendigkeit eines vorsichtigen, taktischen Vorgehens (wie aus früheren Teilen der Reihe bekannt) ein forderndes und interessantes Erlebnis sein könnte. Aber das Spiel wird von ständigen Cutscenes (also eingeschobenen Filmsequenzen) unterbrochen (die sich teilweise nicht einmal überspringen lassen), in denen der Spieler keinerlei Interaktionsmöglichkeit besitzt und still abwarten muss, bis diese vorüber sind. Dadurch wird der Spielfluss dermaßen gestört, dass die Immersion verloren geht und der Spieler vollständig aus dem Flow gerissen wird. Noch schlimmer ist, dass diese Unterbrechungen sogar für nur Sekunden lange Cutscenes geschehen, die absolut unnötig sind, um die Story voranzubringen und die man ohne jeden Verlust entweder vollständig aus dem Spiel hätte streichen, oder zumindest dem Spieler die Möglichkeit geben können, selber den Protagonisten durch diese zu steuern. Der Inhalt der Story ist dabei für die Qualität des Spiels vollkommen unerheblich, wenn schon allein das Storytelling derart störend ist.

In der Vergangenheit wurden bei Spielen verschiedene Herangehensweisen an das Storytelling erprobt, diese Problematik zu vermeiden. In Half-Life 2 ([hl204]) wurde auf Cutscenes im herkömmlichen Sinne verzichtet. Zwar gibt es immer noch einzelne Episoden, in denen das Vorankommen des Spielers pausiert wird, um Story zu vermitteln, aber diese stellen nicht mehr eine so deutliche Unterbrechung

des Spiels dar. Der Spieler behält die Kontrolle über die Spielerfigur und kann sich genauso bewegen, wie sonst auch. Er muss lediglich warten, während zum Beispiel ein anderer Charakter mit ihm spricht, bis sich eine Tür öffnet und er zum nächsten Level gelangen kann. Dadurch wird die Immersion erheblich weniger gestört. Hierbei handelt es sich aber nur um den Schein der Interaktivität, da die Aktionen des Spielers eigentlich keine Auswirkungen haben.

Story versus Gameplay

Das grundlegende Problem besteht in der Trennung von Story und Gameplay. Im Zuge der Entstehung von Spielen hat sich eine funktionale und bewährte Methodik für Gameplay entwickelt. Das Handwerkszeug für Storytelling wurde schon vorher perfektioniert. Angefangen mit dem antiken Drama bis hin zu Film und Fernsehen haben sich klare Regeln herausgebildet, wie man erfolgreich Geschichten erzählt. Diese Herangehensweise und die darauf aufbauenden wissenschaftlichen Disziplinen wurden dann zunehmend auch in Spielen eingesetzt. Diese bewährten Konzepte stellten sich als kosteneffizienter und mit weniger Risiko behaftet heraus, weil Autoren eingestellt werden können, die Erfahrung aus anderen Medien haben. Außerdem ist es für den Entwicklungsprozess auch einfacher, Story und Gameplay getrennt zu behandeln, damit nicht entweder die Programmierer oder die Autoren ständig ihre Arbeit an die des anderen anpassen müssen.

Das Resultat ist, dass das Storytelling meistens vollkommen von Interaktivität befreit ist (wenngleich Spiele außerhalb des Storytellings natürlich sehr interaktiv sind), obwohl sie die größte Stärke des Mediums Videospiel ist und größeres Potential für Spiele als Literatur (im Sinne einer Darreichungsform für Erzählungen) bietet. Die Einzigartigkeit von Spielen besteht darin, dass dem „Leser“, also dem Spieler, die Möglichkeit gegeben wird, mit der Welt, in der ihre Geschichte spielt, zu interagieren. Dies kann zu einem viel immersiveren Erlebnis führen, weil ihm nicht nur eine Geschichte vorgesetzt wird, sondern er ein Teil von ihr sein kann. Besonders stark wird dies erreicht, wenn dem Spieler derartiger Einfluss auf die Handlung erlaubt wird, immer solche Entscheidungen zu treffen, dass sich seine eigene Persönlichkeit in der Story widerspiegelt und so für jeden Spieler ein individuelles Erlebnis darstellt, das quasi auf ihn „maßgeschneidert“ ist. Das Fehlen dieser Möglichkeit kann dazu führen, dass der Spieler sich nicht mit der Geschichte identifizieren kann, weil ihm ein Vorgehen verwehrt bleibt, dass für ihn intuitiv das richtige ist, aber vom Autor nicht gewünscht (oder bedacht) ist und er frustriert das Weite sucht, als ob man einen schlechten Film schaut, bei dem der Hauptcharakter ständig die offensichtlichsten Lösungen nicht erkennt. Damit das Medium also erfolgreich werden und einen festen Platz neben den traditionelleren Formen des Erzählens einnehmen kann, ist es unabdingbar, dass Spiele ihre eigenen Storytelling-Ansätze entwickeln, die diese Stärke ausnutzen.

Stattdessen wirkt die Videospielbranche wenig Innovationsfreudig. Die größeren, wirtschaftlich orientierten Firmen setzen lieber auf bewährte Konzepte und stecken lieber mehr Bemühungen in bessere Grafik (ähnlich Hollywood Blockbuster

Filmen, die Unmengen an Geld für Special Effects aufwenden) als mit neuartigen Storytelling-Ansätzen zu experimentieren.

Quests

Dennoch ist ein Trend sichtbar, dass Versuche gemacht werden, mehr Entscheidungsmöglichkeiten einzubauen, die auch die Story beeinflussen. Schon früh wurde begonnen, die Story und eine Hauptstory und mehrere kleinere Nebenstränge zu teilen, die in einzelne Kapitel getrennt werden, die häufig *Quests* genannt werden. Dabei ist es dem Spieler möglich, nur die Hauptstory zu spielen und den Rest außen vor zu lassen, wobei ihm die verschiedenen Nebenquests lediglich mehr Hintergrundgeschichte zur Welt vermitteln, oder einen spielerischen Vorteil (wie Gegenstände oder Erfahrungspunkte) für die schwieriger werdenden Hauptquests geben. Dieser Weg wurde erst in RPG eingesetzt, um eine große Welt lebendiger zu gestalten (wie in der Elder Scrolls Reihe [tes12]), findet aber mittlerweile auch in vielen FPS Anwendung, um die Spielzeit zu verlängern (wie beispielsweise bei Far Cry 3 [fc312]).

Entscheidungsbäume

Ein anderer Ansatz ist die Verwendung von Entscheidungsbäumen. Hier werden dem Spieler an verschiedenen, festgelegten Stellen in der Story, Entscheidungen vorgelegt. Je nachdem, welche Entscheidung er trifft, folgt darauf eine andere Entwicklung der Story. Dabei können die entstehenden Stränge auch später wieder konvergieren. In vielen Fällen beschränkt sich der Unterschied in der Story auf das Ende, welches meistens auch nur als Cutscene dargestellt wird (zum Beispiel in Heavy Rain [hr10]). Der Grund hierfür ist, dass insbesondere bei den Millionen-Investitionen, die mittlerweile bei großen Titeln üblich sind, jede Szene, die ein Spieler durch eine Entscheidung nicht erreicht, überflüssig investierte Arbeitszeit und Mittel darstellt. Deshalb sind die meisten Spiele von der Story sehr linear aufgebaut und Entscheidungsfreiheit wird höchstens vorgegaukelt.

Moralsystem

Eine dritte Möglichkeit, die sich auch mit den anderen kombinieren lässt, ist ein Moralsystem. Dabei kann der Spieler im Verlauf des Spiels viele Entscheidungen treffen, wobei diese ihn jeweils ein Stück weiter gut oder böse werden lassen. Dabei kann man eine einzelne böse Handlung durch mehrere gute Taten wieder ausgleichen. Bessere Umsetzungen bestrafen oder belohnen dabei den Spieler nicht, wenn er sich für eine Seite entscheidet, sondern geben ihm ein gleichwertiges, aber anderes Erlebnis (wie zum Beispiel Black & White [bw01]). Bei den meisten Spielen, die ein Moralsystem einsetzen, macht es für die gesamte Story keinen wesentlichen Unterschied, außer vielleicht einem anderen Ende (BioShock [bio07]), einem anderen Aussehen des Hauptcharakters (Fable [fab04]) oder anderen Dialogoptionen (Mass Effect [me07]). Manche Spiele zwingen den Spieler dazu, sich entweder für

nur gute oder nur böse Handlungen zu entscheiden, weil manche fortgeschrittene Fähigkeiten, die im späteren Spiel benötigt werden, erst erreicht werden können, wenn der Spieler einen hohen gut- oder böse-Wert hat. Dadurch wird der große Bereich zwischen den Extremen nicht ausgenutzt, obwohl gerade diese Grauzone mit besonders schwierigen moralischen Entscheidungen erheblich zum dramatischen Effekt beitragen könnte.

Simulation

Zwischen Story und freien Entscheidungsfreiheit muss immer ein Kompromiss gefunden werden. Je mehr Freiheiten man dem Spieler lässt, desto schwieriger ist es, eine zusammenhängende Geschichte zu erzählen. Eine übergreifende Narration, in der ein Autor dem Spieler etwas vermitteln will, ist immer ein Eingriff in die Entfaltung des Spielers. Ein Autor kann unmöglich alle möglichen Wege bedenken, die seine vielen Spieler gehen wollen werden.

Es gibt Spiele, die deshalb vollkommen auf eine solche author-story verzichten und stattdessen darauf setzen, dass der Spieler durch die freie Interaktion mit der Welt seine eigene Geschichte schreibt. Diese Spiele werden meistens eher als Simulation bezeichnet, da sie dem Spieler einfach nur eine Welt vorsetzen, die so groß und komplex ist, dass er darin tun und lassen kann, was *er* will. Das einzige, was ihn einschränkt ist der Detailgrad der Simulation, der vor allem durch die Rechenleistung der heutigen Computer begrenzt wird. Solche Spiele nutzen zum Teil prozedurale Generation, um die Welt noch größer und abwechslungsreicher zu machen (zum Beispiel Elite [BB84]). Dadurch, dass dem Spieler keine Story von einem Autor vorgesetzt wird, die ihm vorschreibt, was er tun *soll*, entsteht die Story während des Spiels automatisch durch das, was der Spieler tun *will*, also eine *player-story*. Hier ist sozusagen Gameplay und Story eine Einheit. Beispiele für diesen Spieltyp sind die Sims-Reihe ([sim13]) oder Dwarf Fortress ([AA06]). Der Nachteil daran ist, dass es durch die fehlende Kontrolle eines Autoren schwierig ist, Dinge wie Komposition, Pacing oder Message auf klassische Weise zu vermitteln.

Andere Medien

Es lassen sich viele Inspirationen und Lehren anderen Medien entnehmen. Insbesondere diese Medien, die „ergodisch“ sind (nach Aarseth [Aar97]), da nur hier der Aspekt der Interaktion mit dem Leser einen gewissen Stellenwert hat, so dass es für Spiele interessant wird. Neben den bereits erwähnten P&P-Rollenspielen, sind *Gamebooks* (wie CYOA [cyo88]) ein populäres Beispiel für Narrationen, bei denen die konkrete Handlung erst während des Lesens durch die Interaktion des Lesers festgelegt wird. Dabei ist ein Gamebook ein gedrucktes Äquivalent zu den Entscheidungsbäumen in Videospielstories. Später entwickelte sich *Hyperfiction* (wie Afternoon [Joy90]), wo Gamebooks mit der neuen Technologie der Hyperlinks umgesetzt wurde. Ähnlich dazu sind *Interactive Fiction* (IF), wozu stiltreue Nachfolger von Adventure gezählt werden und *Visual Novels* (die vor allem in Japan beliebt sind).

Ein etwas anderer, modernerer Ansatz ist *Interactive Drama* (oder Interactive Storytelling). Hier werden Computer genutzt, um algorithmisch eine „Storyworld“ ([Cra12]) zu erzeugen, in der sich der Spieler/Leser bewegen kann, um die Handlung zu erforschen. Zu diesem Thema gibt es seit einigen Jahren eine rege Diskussion und viele interessante Ideen, die moderne Algorithmen nutzen, um die vorher genannten Probleme des fehlenden Autors zu lösen. Bei Interactive Drama Anwendungen übernimmt der Computer als Drama Manager, mithilfe von diversen Algorithmen der künstlichen Intelligenz (KI), die Rolle eines Regisseurs (ähnlich einem Dungeon Master im P&P), um dem Spieler das Gefühl einer zusammenhängenden Story zu geben und die Storyworld lebendig zu gestalten. Interactive Drama ist meistens eher *character-driven*, lebt also von vielen Charakteren und deren Beziehungen, im Gegensatz zu den anderen bisherigen Beispielen, die eher *plot-driven* sind, also vor allem Ereignisse betreffen.

1.2 Begriffsbestimmung

Story und Storytelling

Diese Arbeit entwickelt eine Möglichkeit, das Storytelling in Spielen interessanter zu gestalten. Dazu ist zuvorderst Storytelling von Story zu unterscheiden. Wo Story den *Inhalt* einer Erzählung beinhaltet, also das Setting, Thema, welche Ereignisse geschehen, wer die Charaktere sind und womöglich die Message (oder „Moral von der Geschichte“), wohingegen das Story*telling* die *Art und Weise* des Erzählens bezeichnet. Die konkrete Story (oder Geschichte) ist für das Storytelling unerheblich.

Geschichte und Handlung

Eine weitere Unterscheidung ist für die folgende Arbeit essentiell: Geschichte (story) und Handlung (plot). Da diese Begriffe nie eine genaue Definition genossen haben und vor allem in der deutschen Literatur sehr schwammig und universal verwendet werden, sollten andere Termini hinzugezogen werden. Dies wären neben story/plot (nach E. M. Forster [For27]) außerdem fabula/szujet (im russischen Formalismus, wie zum Beispiel bei B. Tomaschewski [Tom25]) oder histoire/discours (G. Genette [Gen72]). Hier interessant sind plot/fabula/discours, die analog auch bei Aristoteles als „mythos“ gefunden werden können (siehe [Fuh94]). Für diese Arbeit genügt es, Handlung als die kausale Abfolge von Ereignissen zu definieren.

Interaktivität

Weiterhin genauer zu betrachten ist der Begriff „interaktiv“, da dieser ursprünglich ein werbewirksames Schlagwort war, das in der Anfangszeit der personal computer verwendet wurde, um von der Batch-Verarbeitung älterer Systeme abzugrenzen. In der Literatur wird der Begriff der Interaktivität teilweise sehr ungenau und manchmal missverständlich eingesetzt, um neue Literaturformen zu beschreiben,

die dem Leser beziehungsweise Benutzer eine Möglichkeit der Partizipation geben. Genauer und korrekter definiert, sei Interaktivität eine Rückkopplungsschleife, in der mehrere Akteure abwechselnd zuhören, denken und sprechen ([Cra12]).

Linearität

Ein weiterer kritischer Begriff ist „Linearität“. Während Benutzer heutiger Videospiele in der Regel das gleiche darunter verstehen, hat die fehlende Definition in der Literaturwissenschft zu jahrelangen Diskussionen geführt. Um Missverständnissen aus dem Weg zu gehen, wird stattdessen auf wohldefinierte Begriffe der Graphentheorie zurückgegriffen.

1.3 Zielsetzung

Als ein möglicher Lösungsansatz, um die beschriebenen Probleme zu lösen, wird hier ein System entwickelt, das Entwicklern von Spielen die Möglichkeit gibt, mit programmiertechnisch geringem Aufwand ein Framework zu erstellen, mit dem die Autoren problemlos interessante, vielseitige und interaktive Stories schreiben können, aber dabei trotzdem die ihnen vertrauten dramaturgischen Techniken anwenden können.

Die Aufgabe des Systems ist, eine emergente Narration durch eine kausal logische Abfolge von gegebenen Ereignissen zu erstellen. Das Ergebnis ist eine Storytellingkomponente zur Einbindung in Spielen, die durch ihre Art der Interaktivität der als „interactive fiction“ bekannten Literatur (oder auch zum Beispiel Gamebooks wie *Choose Your Own Adventure* ([cyo88])) ähnelt. Der Vorteil dieses Systems im Gegensatz zur Verwendung der klassischen Herangehensweise, wie sie im Print – aber auch in modernen Spielen – verwendet wird, ist ein deutlich einfacherer Produktionsprozess (insbesondere bei großer Anzahl von Entscheidungen), eine mögliche komplexere Spiellogik und ein höherer Wiederspielwert.

Spielereinfluß

Das Ziel ist es, Story auf eine Weise zu präsentieren, so dass der Spieler nicht entmündigt wird, sondern bedeutungsvolle Entscheidungen treffen kann, ohne dabei die narrative Integrität der Handlung zu gefährden. Dazu müssen einerseits die Interaktionsmöglichkeiten so gestaltet werden, dass sich der Verlauf der Story entsprechend des Verhaltens des Spielers verändert. Der Spieler muss das Gefühl bekommen, dass seine Handlungen Einfluss auf die Entwicklung der Geschichte haben. Seine Persönlichkeit muss sich in seinen Aktionen und der daraus resultierenden individuellen Story wiederspiegeln können. Dadurch kann sich der Spieler noch besser mit dem Protagonisten identifizieren, wodurch die Geschichte noch mehr fesselt und zu tieferer Immersion führt. Außerdem wird der Wiederspielwert erhöht, da in der Regel bei einem Durchlauf mehrere mögliche Sequenzen von Passagen durch Spielerentscheidungen unerreichbar werden und erst bei einem wiederholten Durchspielen entdeckt werden können.

Narrative Integrität

Andererseits soll das Resultat immer noch ein narratives Ganzes ergeben, um ästhetische Erfüllung zu gewährleisten. Essentiell dafür ist die Auflösung der Story im Sinne von „narrative closure“. Noël Carroll beschreibt dies als das Resultat der Beantwortung aller sich ergebenden Fragen innerhalb einer narrativen Struktur ([Car07]). Die narrative Struktur basiert auf Ganzheit und Einheit der Handlung. Laut Aristoteles ist dies gegeben, wenn „sich das Ganze verändert und durcheinander gerät, wenn irgendein Teil umgestellt oder weggenommen wird“ ([Fuh94]). In anderen Worten: Wenn kein Element der Handlung überflüssig ist und jedes nur an seinem jeweiligen Platz im Verhältnis zu den anderen Elementen stehen kann, dann besteht die notwendige Struktur, damit ein befriedigendes Ende möglich ist.

Dies erklärt sich damit, dass wenn irgendein Handlungselement besteht, das nicht für den Abschluss notwendig ist, dies dazu führt, dass an dieser Stelle Fragen offen bleiben, die nicht mehr aufgelöst werden. Dieses Prinzip findet sich zum Beispiel unter dem Namen *Chekhov's gun* wieder: „Wenn im ersten Akt ein Gewehr an der Wand beschrieben wird, dann sollte es im zweiten oder dritten abgefeuert worden sein“. Ein Beispiel, wo dies bewusst nicht befolgt wird, wäre Monkey Island ([MI90]): Hier haben ein Großteil der Gegenstände keinerlei Zweck, sondern dienen nur der Verwirrung des Spielers, um die Lösungen der Rätsel weniger offensichtlich zu machen (dies wird als *red herring* bezeichnet).

Plot Holes

Die Ordnung der Handlungselemente ergibt sich aus ihren logisch-kausalen Zusammenhängen. Dabei muss ein Ereignis nicht unbedingt deterministisch durch die vorherigen bestimmbar sein, aber zumindest können die vorherigen Ereignisse die Notwendigkeit für ein folgendes sein. Durch die logisch-kausalen Zusammenhänge werden so genannte *plot holes* vermieden und die Kontinuität gewahrt.

2

Lösungsmodell

2.1 Entwurf

2.1.1 Idee

Die Idee ist, die Handlung in atomare Einzelschritte zu zerlegen, also in all die einzelnen Ereignisse, die sie ausmacht. Diese kleinstmöglichen Bausteine, die die Handlung vorantreiben, werden dann dynamisch miteinander verknüpft, um so ein abwechslungsreiches Spielerlebnis zu bieten.

Im folgenden werden diese Einzelschritte *Szenen* genannt. Die Szenen müssen so klein wie möglich sein, aber groß genug, um einen Handlungsfortschritt zu beschreiben, der nicht genau so gut mit einem anderen Schritt zusammengefasst werden könnte. Man könnte dieses Konzept einer Szene mit den aus dem Film bekannten *plot points* vergleichen. Diese atomaren Elemente einer Erzählung werden manchmal in der Literaturwissenschaft als „Narreme“ bezeichnet ([BC12]).

Jede Szene hat eine Reihe an *Bedingungen* an die schon stattgefundene Handlung, die erfüllt sein müssen, damit diese Szene stattfinden kann (zum Beispiel `SteinDerWeisenGefunden: wahr`). Außerdem kann eine Szene verschiedene *Auswirkungen* haben, die eintreten, wenn die Szene stattgefunden hat. Jede dieser Auswirkungen hat eine Reihe an *Effekten* auf den Handlungsverlauf.

Die Gesamtheit all dieser Zustände, die Bedingungen für Szenen sein können, oder auf die Szenen einen Effekt haben können, sei definiert als der *plotstate*. Er ist somit die Summe aller für die Handlung relevanten Tatsachen zu einem bestimmten Zeitpunkt. Der Plotstate verändert sich also nach jeder Szene und bestimmt, welche der Szenen als nächstes eintreten können.

Storyline

Szenen haben aus Sicht der Handlung nur kausale Zusammenhänge, keine temporale Abfolge. Auch wenn die Szenen durch das Erzählen der Geschichte in eine zeitliche Sequenz gebracht werden, ist die Reihenfolge ein indirekter Effekt und nicht vorbestimmt. Vielmehr ergibt sich die Reihenfolge dadurch, dass durch die Ereignisse einer Szene gewisse Tatsachen eintreten, die die Voraussetzung dafür sind, dass andere Ereignisse erst eintreten können. Dies wird durch Bedingungen

und Effekte der Szenen sichergestellt. Zum Beispiel ist es in einer Kriminalgeschichte unerheblich, in welcher Reihenfolge der Detektiv verschiedenen Verdächtigen nachgeht, sofern ihm nicht durch einen der erste Hinweis auf einen anderen gegeben wird.

Diese Tatsachen, beziehungsweise Bedingungen, durch die Szenen kausal miteinander verknüpft werden sind dafür notwendig, dass die Geschichte logisch korrekt ist (ein verstorbener Charakter kann nicht wieder auftauchen), aber vor allem auch dafür, dass die Handlung schlüssig erscheint (der Protagonist ist auf einer Reise fern der Heimat, weil er wegen eines Verbrechen verbannt wurde). Wenn ein Widerspruch im kausalen Zusammenhang auftritt und der Spieler dies bemerkt, resultiert dies unweigerlich in einem Bruch der so genannten *willing suspension of disbelief*. Dadurch wird der Spieler nicht nur aus der Immersion gerissen, sondern empfindet die Handlung auch als verwirrend oder sogar schlampig und könnte zu der Ansicht kommen, dass dadurch dieses Spiel seine Zeit nicht mehr wert ist.

Durch die kausalen Zusammenhänge und die sequentielle Abfolge von Szenen werden die vorher ungeordneten Handlungsfragmente erstmals zueinander in Bezug gesetzt. Dies könnte man als den „roten Faden“ oder die *storyline* bezeichnen (Aristoteles nennt es die „Knüpfung“ [Fuh94]), die im Grunde genommen eine Traversierung über den Graphen aller möglichen Handlungen, die dieser Pool an Szenen hergibt, darstellt.

Dénouement

Für eine Auflösung der Geschichte ist nicht nur eine plausible Entwicklung notwendig, es existieren auch bestimmte Bedingungen für das Ende. Dabei ist eine weitere Grundidee, dass mehrere verschiedene *Enden* existieren (können). Dabei haben die Enden jeweils verschiedene Bedingungen, die bestimmen, wann sie eintreten. Ebenso hat jedes Ende einen bestimmten *Präferenzwert*, der einordnet, wie „gut“ dieses Ende im Vergleich zu allen anderen ist, wie man dies als „goodness“ von Gamebooks (insbesondere CYOA [cyo88]) kennt. So hat zum Beispiel ein Ende, das durch den Tod des Spielers eintritt, wahrscheinlich den geringsten Präferenzwert und das Ende, bei dem der Spieler den Drachen besiegt und die Prinzessin befreit und geheiratet hat, den höchsten. Dadurch wird erheblich die Replayability gesteigert, weil der Spieler, wenn er ein unbefriedigendes Ende erreicht, das Spiel von vorne beginnt, mit dem Ziel, diesmal andere Entscheidungen zu treffen, um zu einem erfüllenderen Ende zu gelangen.

Interaktivität

Jede Szene besteht aus einem Inhalt, der dem Benutzer dargestellt wird und einer Interaktionsmöglichkeit, die dem Spieler gegeben wird. Diese Aktion/Reaktion-Kombination sorgt dafür, dass der Plotfortschritt immer direkt mit der Interaktivität zusammenhängt. Daraus folgert sich, dass die Handlung nicht ohne Mitwirkung des Spielers voranschreiten kann.

Wie der Inhalt ausgegeben wird, ist abhängig von der Art des Spieles. Da der Inhalt nur Informationen an den Spieler liefert, aber für die kausalen Zusammenhänge unerheblich ist, wird er nicht vom System berücksichtigt. Dadurch bleibt es auch universell für verschiedene Anwendungsmöglichkeiten mit unterschiedlichsten Ausgabemethoden verwendbar. Das Spiel, in dem das System Verwendung findet, ist selbst dafür verantwortlich, die aktuellen Szene abzufragen und die Verknüpfung zu den Ausgabeinformationen herzustellen.

Im Allgemeinen können in einem Spiel die bedeutungsvollen Interaktionsmöglichkeiten zweierlei sein:

- Challenge
- Choice

Eine *Challenge* ist eine Herausforderung, die dem Spieler gesetzt wird, die er entweder bestehen kann oder nicht. In den meisten Spielen sorgt der Fehlschlag dazu, dass das Spiel vorbei ist oder der Spieler zumindest diese Herausforderung immer wieder erneut wiederholen muss, bis er sie besteht. Durch das hier vorgeschlagene System entsteht allerdings die Möglichkeit, den Spielfluss nicht zu unterbrechen. Stattdessen führt es dazu, dass dieser Fehlschlag als Teil der Story behandelt werden kann und die zukünftige Handlung daran angepasst wird, so dass ein anderes Spielerlebnis entsteht, das dennoch eine geschlossene Erzählung (narrative whole) bildet und zu einem befriedigenden Ende führen kann.

Dadurch kann die Entwicklung der Handlung stark an das Können des Spielers angepasst werden. So kann etwa das Meistern der Herausforderungen zu einem längeren Spiel führen, oder sich die Stimmung der Story verändern. Zum Beispiel kann das häufige Fehlschlagen dazu führen, dass die Story sich mehr wie eine Tragödie entwickelt, wobei sie sonst ein klassisches Drama darstellen würde.

Bei einer *Choice* wird dem Spieler deutlich eine Entscheidung auferlegt. Bei dieser Entscheidung können beliebig viele Möglichkeiten gegeben werden.

Dadurch wird auch die Replayability gesteigert, da der Spieler nach Abschluss des Spiels erneut spielen kann, um die anderen Möglichkeit zu erkunden, wie sich die Story entwickelt haben könnte.

Für das System ist es unerheblich, ob eine Interaktion eine Challenge oder eine Choice ist. Für das System ist beides eine Szene mit mehreren verschiedenen möglichen Auswirkungen. Wie dem Spieler diese Interaktionsmöglichkeit präsentiert wird, ist Aufgabe der Gameplay-Mechaniken des Spiels. Es muss lediglich die Rückmeldung gegeben werden, zu welchem Ergebnis es gekommen ist. Dafür bietet sich eine Art Lookup-Tabelle an, die die Inhalte auf die Namen oder Handles der Szenen und Auswirkungen abbildet.

2.1.2 Pfadplanung

Um die einzelnen Szenen entsprechend den bisher aufgestellten Grundsätzen zu verknüpfen, wird ein Pfadplanungssystem verwendet. Das Planungssystem wird deshalb verwendet, weil bei einer größeren Menge an Szenen und Zuständen es beinahe unmöglich wird, einen Graphen über alle möglichen Plotentwicklungen

von Hand zu konstruieren, wie dies bei decision-trees getan wird, wie sie von heutigen Spielen für variable Handlungsverläufe verwendet werden. Da also nicht von vornherein ein Graph definiert ist, kann kein gewöhnlicher Wegfindungsalgorithmus, wie zum Beispiel A*, verwendet werden, der diesen traversiert. Stattdessen bietet sich eine Variante an: iterative deepening A* (die genaue Funktionsweise wird in Abschnitt 2.3.1 ausführlich erläutert). Zwar könnte man auch einen Graphen algorithmisch konstruieren, dies würde aber Exponentialzeit benötigen, die von IDA* nur im worst-case erreicht wird. Die hier benutzte Implementierung von IDA* sucht nicht über ein Netz von Knoten, sondern über Zustände. Durch die Menge von Zuständen und wie sie von den Szenen verändert werden können, wird ein Raum der möglichen Plotzustände aufgespannt. Durch diesen Raum der möglichen Plotstates muss ein Pfad gefunden werden. Man könnte diesen Pfad, die Storyline, als den Handlungsverlauf bezeichnen. Er ist die Sequenz von Ereignissen, die der Spieler tatsächlich in diesem Durchlauf erlebt. Alle übrigen möglichen Szenen bleiben ihm vorerst verborgen und regen zu einem erneuten Spielen an. Dieser Pfad ist eine Sequenz von Szenen bis hin zu einem Ende. Wenn mehrere mögliche Enden vorgesehen sind – was sich anbietet – müssen diese jeweils eine unterschiedliche „Goodness“ besitzen. Also eine Wertung, die angibt, in welcher Reihenfolge sie vom Planer angesteuert werden. Dabei wird immer das nächste gewählt, wenn das derzeitige nicht mehr erreichbar ist. Angemessen wäre, zuerst die Enden anzusteuern, die für den Spieler am wenigsten ansprechend sind. Dadurch wird erreicht, dass ein „schöneres“ Ende nach längerer Spielzeit und interessanteren Entscheidungen erreicht wird.

Der Algorithmus wählt dabei nur die Szenen, deren Bedingungen erfüllt sind. So wird der Grundsatz der kausalen Verknüpfung gewahrt. Dabei reagiert der Algorithmus auf Spielereingaben, sobald diese irgendwelche Bedingungen ändern, die dazu führen können, dass eine Abweichung vom Pfad stattfindet. Dann wird ein neuer Pfad über andere Szenen gesucht. Dadurch werden die an sich ungeordneten Knoten/Szenen erst im Verlauf der Planung und des Spiels in Relation gesetzt.

2.2 Spezifikation

2.2.1 Definition

Das Modell zur Verarbeitung der Handlung wird folgendermaßen definiert:

- der Plotstate, eine Liste von Zuständen $W = \{w_0, \ldots, w_n\}$. Zur Vereinfachung des Algorithmus werden die Zustände als Wahrheitswerte $w_i \in \{0, 1\}$ definiert.
- einen Startzustand W_0,
- eine Menge an Enden $G = \{g_0, \ldots, g_n\}$. Ein Ende ist ein Tupel $g_i = (C, p)$. Dabei ist $C \subseteq W$ der Zielzustand, eine Menge an Zuständen, die festlegen, wann dieses Ende erreicht ist und einen Präferenzwert p, der bestimmt, in welcher Reihenfolge die Enden angestrebt werden.
- eine Menge an Szenen S bestehend aus einzelnen Szenen, die definiert sind als $s_i = (C, O)$. Wobei $C \subseteq W$ eine Menge an Zuständen ist, die die Bedingung für

diese Szene sind. $O = \{o_0, \ldots, o_n\}$ ist eine Menge an den möglichen Auswirkungen $o_i = (E)$ dieser Szene. Diese Auswirkungen haben eine Menge an Zuständen $E \subseteq W$, welche den Effekt dieser Auswirkung auf den Plotstate bezeichnen.

2.2.2 Verfahren

1. Setze den Startzustand als momentanen Plotstate.
2. Wähle das Ende mit höchster Priorität.
3. Plane einen Pfad vom momentanen Plotstate zum derzeitigen Ende.
   Wenn kein gültiger Pfad gefunden wurde: Lösche das derzeitige Ende, gehe zu Schritt 2.
4. Stelle die erste Szene dar, die der Plan vorsieht.
5. Warte auf Spielerreaktion, wende den Effekt auf den Plotstate an und lösche die derzeitige Szene aus dem Plan.
   Wenn die Spielerreaktion von der geplanten abweicht, lösche den derzeitigen Plan und gehe zu Schritt 3.
6. Wenn der Zielzustand erreicht ist, terminiere.
   Sonst gehe zu Schritt 4.

2.2.3 Beispiel

Abbildung 2.1 zeigt eine beispielhafte Situation. Es existieren drei Ziele g_0, g_1, g_2, gekennzeichnet als Quadrate mit Prioritätswert (kleinere Zahl bedeutet höhere Priorität) und ihren Bedingungen C (kleine Quadrate). Außerdem gibt es vier Szenen (große Kreise) $s0, \ldots, s3$, die unterschiedliche Bedingungen sowie diverse Auswirkungen $o_0, \ldots, o_n$ (kleine Kreise) mit entsprechenden Effekten haben. Der Startzustand $W_0 = \{1, 1, 0, 1, 0\}$ ist als Dreieck dargestellt.

Zuerst wählt der Planer das Ende mit der höchsten Priorität (g_0) und sucht den kürzesten Pfad (blau) dorthin.

Wenn der Spieler in der Szene s_0 sich nicht für die erwartete Auswirkung o_0, sondern stattdessen für o_1 (gelb) entscheidet, ist der Plan nicht mehr gültig und es muss erneut geplant werden (siehe Abbildung 2.2). Die nächste dargestellte Szene ist demnach s_1.

Wenn der Spieler sich bei Szene s_1 erneut nicht plangemäß verhält (Abbildung 2.3), wird wieder neu geplant. Nun findet der Planer aber keinen möglichen Pfad mehr zu g_0. Also wird g_0 gelöscht und stattdessen das Ende mit der nächst höheren Priorität (g_1) gewählt.

Abbildung 2.4 zeigt ein mögliches Ende: Nach vier Schritten ist der derzeitige Plotstate $W_4 = \{0, 0, 1, 1, 1\}$. Somit sind alle Bedingungen des aktuellen Endes g_2 erfüllt und die Handlung abgeschlossen.

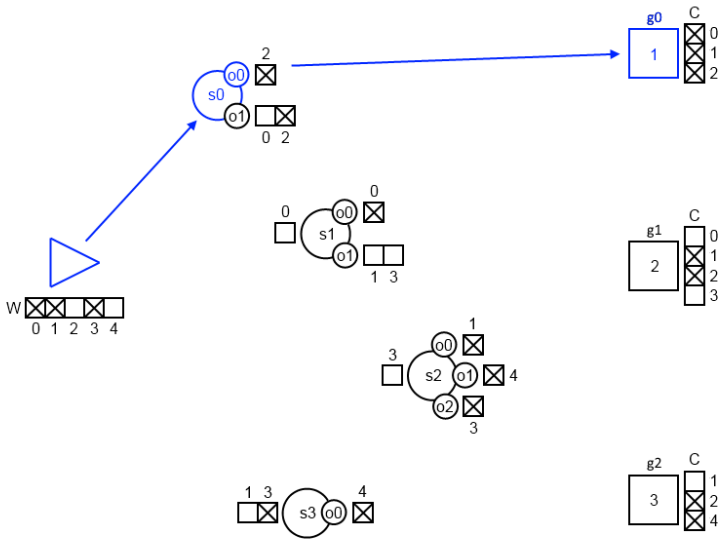

Abb. 2.1. Ausgangszustand nach erster Planung

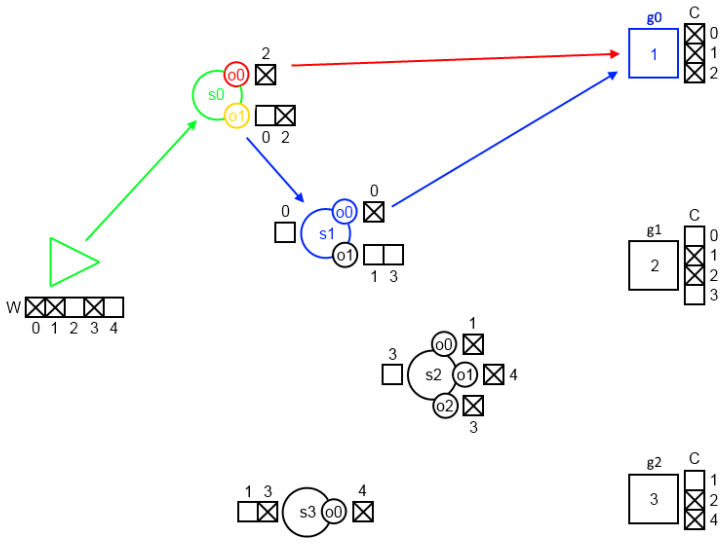

Abb. 2.2. Neuer Plan nach Abweichung bei s_0

2.3 Funktionsweise

2.3.1 GOAP

Um den Entwurf umzusetzen, wird ein Verfahren zugrunde gelegt, das sich *Goal Oriented Action Planning* (GOAP) nennt. Es basiert auf dem STanford Research Institute Problem Solver (STRIPS) ([Ork06]). Dieses Verfahren wurde von Jeff Orkin entworfen ([Ork04]) und von ihm in FEAR ([fea05]) zur Verwendung geführt. Es eignet sich hervorragend für die Entscheidungsfindung einer KI zum Beispiel für First-Person-Shooter, wie etwa FEAR. Dort ersetzt es herkömmliche Metho-

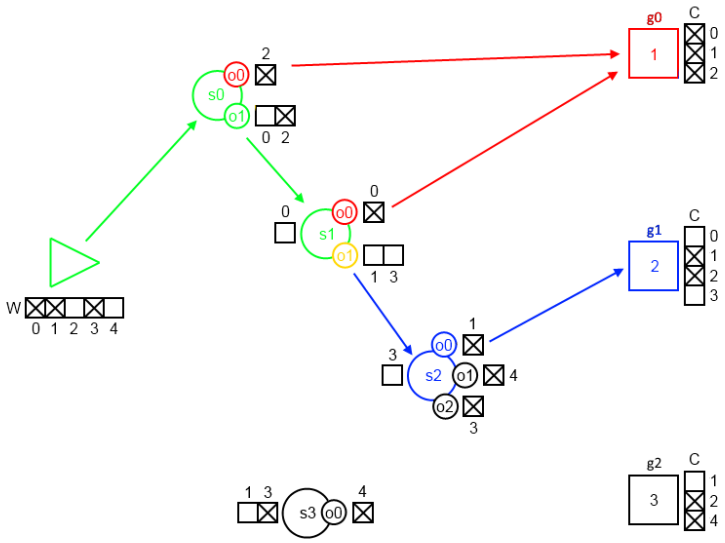

Abb. 2.3. Neuer Plan nach Abweichung bei s_1

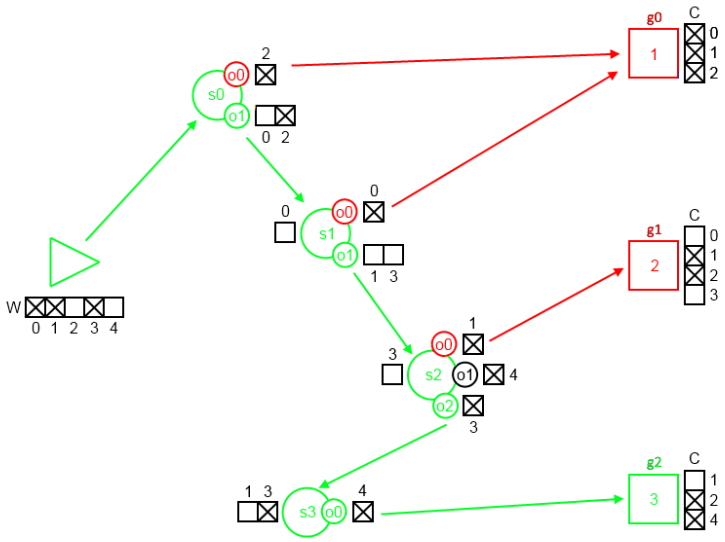

Abb. 2.4. Das Ziel ist erreicht

den, wie Zustandsautomaten, Entscheidungsbäume oder regelbasierte Systemen die schnell vom Spieler durchschaut werden, sehr mechanisch wirken und vor allem bei komplizierteren Spielen für Programmierer und Designer sehr unübersichtlich und schwer zu pflegen oder zu erweitern sind.

GOAP verwendet einen Suchalgorithmus, der den optimalen Weg von einem Startzustand zu einem Zielzustand findet. Der Zielzustand wird als *Goal/Ziel* bezeichnet. Dem Algorithmus stehen so genannte *Actions* zur Verfügung, die den Zustand verändern können, um Schritt für Schritt zum Goal zu gelangen. Außerdem haben sie Bedingungen an den derzeitigen Zustand, die erfüllt sein müssen,

damit sie verwendet werden können. Der Zustand wird in einem Konstrukt, das *WorldModel* genannt wird, modelliert.

Beispielhaft für GOAP wäre folgendes Szenario: Ein KI-Gegner in einem FPS nimmt den Spieler wahr und bekommt daraufhin das neue Ziel, diesen Spieler zu töten. Dazu stehen ihm verschiedene Mittel und Wege zur Verfügung. Zum Beispiel könnte er dem Spieler im Nahkampf gegenübertreten, ihn mit seiner Feuerwaffe unter Beschuss nehmen oder mit der Umgebung interagieren, indem er zum Beispiel ein stationäres Geschütz benutzt. Jede dieser Aktionen hat bestimmte Voraussetzungen/Bedingungen. So muss er für den Nahkampf bis dicht an den Spieler herantreten, für den Schusswaffengebrauch zumindest im selben Raum sein und für das Geschütz erst zu diesem gehen. Außerdem müssen die Waffen geladen sein und eventuell auch andere Bedingungen erfüllt sein, die das eigene Überleben sichern.

Der Suchalgorithmus findet nun einen Weg zu dem Ziel. Der Weg stellt eine Verkettung von Aktionen dar, die notwendig sind, um das Ziel zu erreichen. Ein mögliches Ergebnis wäre beispielsweise, dass zuerst der Raum betreten, dann Deckung gesucht, die Waffe gezogen, nachgeladen und dann gefeuert wird, bis das Ziel eliminiert wurde.

Bei einer großen Anzahl von möglichen Aktionen, aus der die KI wählen kann, um ihre Ziele zu erreichen, werden von Hand konstruierte Zustandsautomaten oder ähnliches so unhandlich, dass eine Veränderung nach Beginn der Entwicklung praktisch unmöglich wird. GOAP erlaubt, dass auch noch nachträglich unproblematisch neue Aktionen oder Ziele hinzugefügt werden können. Sie müssen lediglich mit Bedingungen versehen und dem Pool hinzugefügt werden, aus dem sich der Suchalgorithmus bedienen kann. Außerdem können so große Teile des KI-Verhaltens in unterschiedlichem Kontext wiederverwendet werden. Zum Beispiel kann ein neuer Gegnertyp erstellt werden, der zwar keine Schusswaffen benutzt, sich aber sonst das Verhalten mit den anderen KI-Gegnern teilt. Hierzu muss nur bei den jeweiligen Aktionen eine neue Bedingung `kannWaffenBenutzen` hinzugefügt werden und diese beim neuen Gegnertyp auf falsch gesetzt werden. Ein günstiger Nebeneffekt davon ist, dass hier sehr gut Code von Daten getrennt werden kann, was für manche Software-Architekturen sehr sinnvoll ist.

Statt einem KI-Gegner im herkömmlichen Sinne, wie sie zum Beispiel bei Shootern auftauchen, muss man sich den „Gegenspieler“, der mittels GOAP entscheidet, für die Zwecke des hier entworfenen Systems eher wie einen Dungeon Master im klassischen P&P vorstellen. Die KI plant at-runtime einen Weg, den der Spieler gehen muss, während dem er verschiedene Herausforderungen bestehen muss und diverse Ereignisse stattfinden, um zu einem Ziel zu gelangen. Dabei möchte der DM den Spieler immer wieder zu einem „vorzeitigen Ende“ führen, welchem dieser nur durch Erfolge und interessantem Spiel entgehen kann, bis er irgendwann ein Ende erreicht, das ihn erfüllt beziehungsweise ihm genügt. Dieses Gegenspiel, in dem Spieler und DM zusammen die Geschichte vorantreiben, entspricht Aarseths Konzept von *intrigue* ([Aar97]). Analog entspräche die KI des Systems dem *intrigant* ergodischer Werke. Etwas weiter gefasst könnte man diesen Ansatz – wie auch das vergleichbare Konzept des *drama manager* aus dem Interactive Drama

– als einen möglichen Schritt in Richtung des von Brenda Laurel vorgeschlagenen *Playwright* sehen ([Lau86]).

WorldModel

Damit die KI arbeiten kann, muss sie ein Verständnis von der Welt haben. Alle Informationen, die für sie wichtig sind, müssen irgendwo gespeichert sein. Dafür verwendet GOAP ein so genanntes WorldModel, in dem der Zustand der Welt an einem bestimmten Zeitpunkt modelliert ist.

Die wichtigsten Daten sind, welche Bedingungen derzeit erfüllt sind und welche nicht. Relevant dafür sind all die Bedingungen, die Actions haben können oder all die Faktoren, auf die sie einen Einfluss haben können.

Was außerdem im WorldModel hinterlegt wird, ist eine Liste an allen möglichen Actions. Dies macht vor allem dann Sinn, wenn Actions nur einmal durchgeführt werden können und danach nicht wieder stattfinden können sollen, was hier für die Szenen zutrifft.

Actions

Eine Action ist ein Einzelschritt eines Plans, in dem etwas *geschieht*. Das heißt, eine Action bewirkt eine Veränderung im WorldModel. Im Sinne klassischer FPS-KI lässt eine Action einen Charakter etwas *tun*. Im allgemeinen hinterlässt eine Action die Welt in einem anderen Zustand, als dem, in der sie sie vorgefunden hat. Actions können in der Regel nicht immer ausgeführt werden. Sie haben gewisse Anforderungen an das WorldModel, also *Voraussetzungen*, die erfüllt sein müssen. Dem gegenüber stehen *Effekte* einer Action, also die Veränderung, die diese Action am WorldModel verursacht. Eine Action kann beliebig viele Effekte und beliebig viele oder keine Voraussetzungen besitzen. Voraussetzungen und Effekte erlauben, dass Actions in eine gültige Sequenz zu einem Plan zusammengeführt werden können. Actions können mit unterschiedlichen Kosten versehen werden, die in der Planung berücksichtigt werden, um Actionsequenzen mit geringeren Kosten zu bevorzugen.

Goal

Das Goal ist der Zielzustand, der erreicht werden soll. Es muss direkt mit einem WorldModel verglichen werden können, um zu überprüfen, ob dieses WorldModel das Goal erfüllt. Ein Goal ist eine Teilmenge der Zustände des WorldModels, die mit konkreten, vordefinierten Werten belegt sind. Wenn das Goal erreicht ist, terminiert der Algorithmus.

Planung

Das Planungsverfahren beginnt mit dem Setzen eines Startzustands (als WorldModel), eines Goals und der Bereitstellung aller zur Verfügung stehenden Actions. Über den Suchraum aus Actions soll nun durch einen Suchalgorithmus ein Weg zum Ziel gefunden werden. Der bekannteste, A* geht davon aus, dass mindestens

ein möglicher Weg zum Ziel existiert und sucht so tief wie nötig, um ihn zu finden. Dies ist aber bei GOAP nicht immer möglich. Häufig ist ein Ziel gar nicht erreichbar. Außerdem stehen meist sehr viele Actions zur Verfügung, von denen immer zumindest ein paar ausgeführt werden können. Dadurch ist es sehr schwierig, die von A* geführte offene und geschlossene Liste zu verwenden. Vielmehr bietet sich die Variante IDA* an, die der Tiefe nach sucht und nach einer festgelegten Tiefe abbrechen kann. Anderenfalls würde der Algorithmus nicht terminieren, wenn keine Lösung gefunden werden kann und unbegrenzt weiter Speicher anfordern.

IDA* verwendet eine Heuristikfunktion, wie alle A*-Varianten. Diese Heuristik schätzt, wie weit das Goal vom derzeit untersuchten Knoten (beziehungsweise WorldModel) entfernt ist, um so bestimmte Actions bevorzugt zu betrachten und womöglich unsinnige vorerst auszublenden. Dadurch kann die Durchschnittslaufzeit erheblich optimiert werden. Falls es nicht möglich ist, eine sinnvolle Heuristik zu entwickeln, kann auch die Null-Heuristik verwendet werden, wodurch der Algorithmus dann dem Dijkstra-Algorithmus entspricht.

Der Ablauf von IDA* ist folgender: Zu Beginn wird die Heuristik-Funktion des Startzustands aufgerufen und als ein cutoff-Wert gespeichert. Dann wird eine Tiefensuche durchgeführt, bis entweder ein gültiger Pfad gefunden wurde oder alle möglichen Actionsequenzen ausgeschöpft wurden, ohne ein Goal zu erreichen. Die Tiefensuche wird durch zweierlei Faktoren begrenzt: die maximale Suchtiefe und den anfangs initialisierten cutoff-Wert, der die Suche abbricht, wenn er kleiner ist, als die Gesamtkosten des bisherigen Plans. Wenn in einer Tiefensuche kein gültiger Pfad gefunden werden konnte, wird der cutoff-Wert auf den Wert der kleinsten gefundenen Plankosten der vorherigen Suche erhöht und eine erneute Suche begonnen.

Die Tiefensuche erzeugt eine Kopie des Startzustands, wendet darauf die erste mögliche Action (das heißt, deren Voraussetzungen erfüllt sind) an, indem die Effekte dieser Action auf das WorldModel angewendet werden und speichert den resultierenden Zustand in einer neuen Kopie für die nächste Tiefe. Für die nächste Tiefe wird außerdem gespeichert, wie hoch die Kosten für den bisherigen Pfad bis zu dieser Stelle waren. Dann wird in der nächsten Tiefe wieder die erste mögliche Action (auf diesem Pfad vorher schon gewählte Actions sind nicht mehr möglich) gewählt, auf den Zustand dieser Tiefe angewendet und das Ergebnis für die nächste Tiefe gespeichert. Wenn die maximal mögliche Suchtiefe erreicht wurde, ohne das Goal zu erreichen, wird zur vorherigen Tiefe zurückgesprungen und die nächste Action gewählt. Wenn an irgendeiner Stelle das Goal erreicht wurde, wird der dorthin gefundene Plan und dessen Kosten hinterlegt. Dann wird noch weiter gesucht, um einen günstigeren Plan zu suchen, dabei aber immer direkt abgebrochen, wenn die Kosten die des bisher günstigsten Plans übersteigen, um die Laufzeit so kurz wie möglich zu halten. Am Ende wird der gefundene Plan oder, falls keiner gefunden wurde ein neuer cutoff-Wert, zurückgegeben.

Transposition Table

Da es nicht wie bei A* eine closed-Liste gibt, um Duplikate zu vermeiden, die es aber bei GOAP-Anwendungen häufig gibt (das gleiche WorldModel kann durch verschiedene Actionsequenzen erreicht werden), wird stattdessen eine *Transposition Table* verwendet (nach dem englischen „transposition“ aus dem Schach). Diese ist eine Hashtabelle, bei der zu jedem Hashwert nur ein Element gespeichert werden kann. In ihr wird jedes bereits erreichte WorldModel hinterlegt. Falls nun der Hash eines WorldModels schon in der Tabelle vorgefunden wird, wird geprüft, welches in weniger Schritten (also in einer geringeren Tiefe) erreicht wurde und nur fortgefahren, wenn das neue WorldModel niedriger liegt (dieses kommt dann an den Platz des anderen in der Tabelle).

2.3.2 Transfer

Planung

Eine mögliche Heuristik sei durch folgende Funktion gegeben:
$h(W_{now}, g_{current}) = \#\{c \in C(g_{current}) | c \notin W_{now}\}$

Die geschätzte Entfernung zum Ziel ist also die Summe aller Zustände C des derzeitigen Ziels $g_{current}$, die nicht im derzeitigen Plotstate W_{now} erfüllt sind. Dies ist eine sinnvolle Heuristik, wenn durchschnittlich jede Action einen der fehlenden Zustände setzt, man also dem Ziel mit jeder Action „einen Schritt näher“ kommt. Wenn es eher die Regel ist, dass man Umwege über Zustände gehen muss, die für den Zielzustand irrelevant sind, spart diese Heuristik keine Kosten und es sollte der Einfachheit halber auf die Null-Heuristik $h(W_{now}, g_{current}) = 0$ zurückgegriffen werden.

Da hier die Bedingungen nur als Wahrheitswerte implementiert sind, bietet sich als Hash-Funktion an, einen Integer bitweise mit der Liste der Bedingungen des WorldModels zu füllen. Also wenn der erste Zustand der Liste wahr ist, ist das letzte Bit des Integers 1, anderenfalls 0, beim zweiten Zustand das vorletzte Bit und so weiter. Wenn die Anzahl der Zustände über die Länge des größten Integer-Datentyps der gewählten Programmiersprache hinaus geht, könnte stattdessen ein String verwendet werden. Bei dieser Hash-Funktion muss aber beachtet werden, dass in jedem WorldModel die gleichen Zustände enthalten sind und sie in der gleichen Reihenfolge stehen.

Enden

GOAP findet einen Weg zu *einem* Goal. Im Gegensatz dazu geht das hier entworfene System von *mehreren* Enden aus, die erreicht werden können. Alle Enden haben selbstverständlich unterschiedliche Bedingungen, die erfüllt sein müssen, aber sie haben auch alle einen jeweils unterschiedlichen Präferenzwert, der bestimmt, in welcher Reihenfolge sie der Planer erfasst. Es muss also stets eine Liste der Enden geführt werden. Diese Liste wird nach dem Präferenzwert aufsteigend sortiert (wenn ein kleinerer Wert höhere Präferenz bedeutet). Bei der ersten Planung wird

GOAP das erste Ende in dieser Liste als Goal übergeben. GOAP plant nun immer einen Weg zu diesem Ende. Falls GOAP irgendwann keinen gültigen Pfad mehr zurückliefert, ist dieses Ende nicht mehr erreichbar, weil es keine Actions beziehungsweise Szenen mehr gibt, die dahin führen können. Dann löscht das System dieses Ende aus der Liste und übergibt das nächste an GOAP und so weiter.

Falls die Liste irgendwann leer sein sollte, ohne dass ein Ende erreicht wird, hat sich der Spieler in eine Sackgasse manövriert. Dies passiert auch, wenn alle Szenen verbraucht wurden, ohne ein Ende zu erreichen. Wenn die Szenen erstellt werden, sollte man möglichst dafür sorgen, dass so eine Situation nicht eintreten kann. Gegebenenfalls könnte man auch einen Trap-State implementieren, der sozusagen ein zusätzliches Ende darstellt und immer dann erreicht wird, wenn keines der normalen Enden mehr erreicht werden kann. So bleibt der Spieler nicht einfach irgendwo stecken, sondern kann im Spiel darauf hingewiesen werden, dass er in eine Sackgasse geraten ist und das Spiel erneut spielen sollte. Dies könnte sogar ein gewünschter Effekt sein, wenn es so wenige mögliche Wege gibt, an ein Ende zu geraten, dass die Traversierung der Szenen ein Puzzle darstellt, welches zum Charme des Spiels gehört.

Szenen

Um das Prinzip der Szenen in GOAP umsetzen zu können, müssen vorher einige Zwischenschritte umgesetzt werden. Die Actions in GOAP erlauben nicht mehrere verschiedene Auswirkungen. Außerdem muss bei der Implementation darauf geachtet werden, an welcher Stelle die Interaktionsmöglichkeiten des Spielers eingebaut werden. Um Szenen also in GOAP anzuwenden, müssen diese erst in Actions übersetzt werden. Dazu bietet sich an, für jede mögliche Auswirkung einer jeden Szene eine eigene Action zu erzeugen. Dabei müssen aber alle Actions, die einer Szene x entspringen, eine zusätzliche Bedingung und einen zusätzlichen Effekt erhalten. Die Bedingung heißt zum Beispiel `SzeneXstattgefunden` und muss auf falsch gesetzt sein. Der Effekt aller Auswirkungen dieser Szene setzt diese Bedingung dann auf wahr. Dadurch wird vermieden, dass mehrere Auswirkungen einer Szene geplant werden können (und nebenbei auch, dass eine Szene mehrmals verwendet werden kann). Wenn die Szenen so in Actions umgewandelt werden, muss der GOAP-Algorithmus nicht weiter angepasst werden und könnte im Spiel an anderer Stelle – wie etwa für eine KI zur Steuerung von NPC (non player character) – wiederverwendet werden.

Interaktion

Die Interaktion des Spielers findet immer während einer Szene statt. Nachdem mit GOAP ein Pfad gefunden wurde, werden nacheinander die Actions vom Spiel ausgeführt. Das bedeutet, dass erst wieder die Szene aus den vorher aufgeteilten Actions rekonstruiert wird und dann der Inhalt, der zu dieser Szene gehört wiedergegeben wird. Dies kann je nach Spiel eine reine Textausgabe sein, oder die Darstellung und Simulation eines ganzen Levels oder Weltabschnitts. Bei einem

Open-World-Spiel wäre der Inhalt einer speziellen Szene zum Beispiel das Vorhandensein von bestimmten Charakteren oder Dialogoptionen, die für das nächste Storysegment relevant sind.

Nachdem dem Spieler die Ausgabe vorgelegt wird, wartet das Spiel auf eine Eingabe des Spielers. Bei einem Text-Adventure wäre das tatsächlich ein Prompt, der auf die Eingabe eines Strings wartet, bei einem 3D-Spiel eher das Betreten eines neuen Levels, die Interaktion mit einem bestimmten Objekt im Spiel oder bei Rollenspielen die Auswahl einer bestimmten Dialogoption bei einem bestimmten Hauptcharakter. Dabei sind alle Interaktionen mit sonstigen Gegenständen oder Objekten in der Welt irrelevant (zum Beispiel das Töten von zufällig spawnenden Monstern oder das Reden mit einem Händler).

Wenn die Eingabe des Spielers mit der vom Plan erwarteten Auswirkung der derzeitigen Szene übereinstimmt, wechselt das Spiel zur nächsten Szene und stellt ihre Ausgabe dar. Falls die Spielereingabe eine andere ist, wird der Plan invalidiert und das System muss neu planen.

3

Implementation

In dieser konkreten Implementierung werden für alle Zustände, also auch Bedingungen und Effekte von Szenen/Actions nur Boolean-Variablen als Flags verwendet. Um die Schnittstellen universell, robust und klar trennbar zu gestalten, wurden alle Wechselwirkungen mit String-Schlüsseln implementiert, nicht per Objekt-Referenzen. Diese Identifikationsschlüssel (IDs) werden zur Unterscheidung von Zuständen, Szenen und Enden, sowie der davon abstrahierten Actions und Goals verwendet. Das System und der Prototyp wurden in C# implementiert. Die Ein- und Ausgabe soll, angelehnt an IF oder Text-Adventures, nur über Text geschehen. Also muss für jede Szene ein Ausgabe-String und für jede Auswirkung aller Szenen ein Eingabe-String hinterlegt werden. Zusätzlich bekommt jedes Ende einen Ausgabe-String, um dem Spieler anzuzeigen, welches Ende erreicht wurde (und gegebenenfalls die Auflösung wiederzugeben).

Alle Daten lädt das System aus einer XML-Datei. Dazu gehören der Startzustand mit allen Zuständen, die für diese Handlung relevant sind, die Enden, die erreicht werden können inklusive ihrer Bedingungen und Ausgabe, sowie die Szenen samt Bedingungen, Ausgabe und Auswirkungen mit deren Eingaben und Effekten. Dabei werden alle Bedingungen der Szenen und Enden, die nicht im Startzustand enthalten sind, dort hinzugefügt und auf falsch gesetzt.

Die Zustände (in Abbildung 3.1 als Klasse `Condition` verzeichnet) wurden nicht als eigene Klasse oder struct implementiert, sondern finden sich als Key-Value-Paare (Schlüssel: String und Wert: Boolean) in Dictionaries der anderen Klassen wieder und reduzieren so Abhängigkeiten auf das Package `GOAP`.

Wenn die Ein- und Ausgabe des Spiels komplizierter als reiner Text sind, wie dies zum Beispiel bei einem modernen 3D-RPG der Fall wäre, dann muss das Spiel selbst eine Lookup-Tabelle führen, mit der sie die String-IDs mit den für sie relevanten Informationen verknüpft, wie etwa die Ausgaben der Szene mit einem bestimmten Level oder die Eingaben der Auswirkungen mit Dialogoptionen bei NPCs.

Zu Beginn nutzt die Klasse `NGOAPSystem` die Methode `readFromXML()`, um alle erforderlichen Daten von der Festplatte zu laden und für die spätere Verwendung in die Datenstrukturen zu füllen. Da `Scene` nicht von der GOAP-Komponente verwendet werden kann, wird die Methode `sceneToAction()` eingesetzt, um alle Szenen in Objekte vom Typ `Action` umzuwandeln (siehe dazu 2.3.2). Die da-

bei entstehenden Actions bekommen zur Identifikation einen Schlüssel der Form `SceneID_OutcomeID` (daher dürfen keine Unterstriche in IDs von Szenen oder Auswirkungen vorkommen), womit später auch wieder die dazugehörige Szene ermittelt werden kann. Die Kosten von Actions werden hier nicht genutzt und daher auf 1 festgelegt.

In `NGOAPSystem` werden auch immer der derzeitige Plotstate (als `Worldmodel`) und der Plan, der gerade durchgeführt wird, gespeichert. Außerdem sind hier alle noch erreichbaren Enden als `Goal` in einer Liste, sortiert nach ihrem Präferenzwert, hinterlegt, wobei immer das mit der höchsten `insistence` zusammen mit dem aktuellen Zustand (`currentState`) der GOAP-Komponente zur Planung übergeben wird.

Das restliche Programm, in das `NGOAP` eingebunden wird, greift ausschließlich auf `NGOAPSystem` zu. Es übergibt bei der Initialisierung einen Dateipfad an `readFromXML()`. Die Verbindung der Ein- und Ausgabe des Spiels mit dem System geschieht per `getCurrentOutput()`, das den Ausgabestring der derzeitigen Szene zurückgibt und `setInput()`, das einen String übergeben bekommt, der die dadurch bestimmte Auswirkung in Gang setzt, oder nichts tut, falls der Input irrelevant war (wenn die derzeitige Szene keine Auswirkung mit dieser Eingabe hat).

Falls das Spiel Mechaniken hat, welche die Handlung beeinflussen, die aber nicht durch Szenen und Auswirkungen dargestellt werden können (zum Beispiel Bedingungen, für die ein Boolean nicht ausreicht, wie etwa die Überprüfung, ob der Spieler eine bestimmte Menge Geld im Inventar hat), kann das Spiel aus der Spiellogik heraus mit `changeState(string id, bool newState)` auf den derzeitigen Plotstate zugreifen, und ein einzelnes Flag ändern. Dadurch wird der derzeitige Plan ungültig und neu geplant. Es muss aber nicht zwangsweise zu einem anderen Plan kommen, der vorherige kann immer noch angemessen sein.

So muss auch der Zustand manipuliert werden, wenn einzelne oder alle Szenen mehrmals verwendbar sein sollen, da als die Szenen in Actions umgewandelt werden, jeder resultierenden Action automatisch die Bedingung, dass diese Szene noch nicht verwendet wurde, hinzugefügt wurde. Dies ist notwendig, da sonst der Planungsalgorithmus alle Auswirkungen der Szenen nacheinander verwenden könnte, obwohl sie sich gegenseitig ausschließen. Außerdem wird so vermieden, dass er während dem Planungsprozess eine mehrmalige Wiederholung von nur einer Szene als möglichen Pfad betrachtet. Dadurch wird nicht nur eine Endlosschleife vermieden, der Algorithmus wird auch effizienter, weil er weniger Möglichkeiten überprüfen muss. Also muss nach dem Passieren einer Szene mit `changeState()` diese Bedingung wieder auf falsch gesetzt werden, damit diese Szene noch einmal vorkommen kann.

Immer wenn die Klasse `NGOAPSystem` keinen gültigen Plan `currentPlan` mehr hat, ruft sie auf `Planner` die Methode `planActions()` auf, die mittels IDA* einen Pfad sucht und diesen zurückgibt. Falls kein Plan zurückgegeben wurde, ist das derzeitige Ende nicht mehr erreichbar und es wird das nächste in der Liste `possibleEndings` für die folgenden Planungen verwendet.

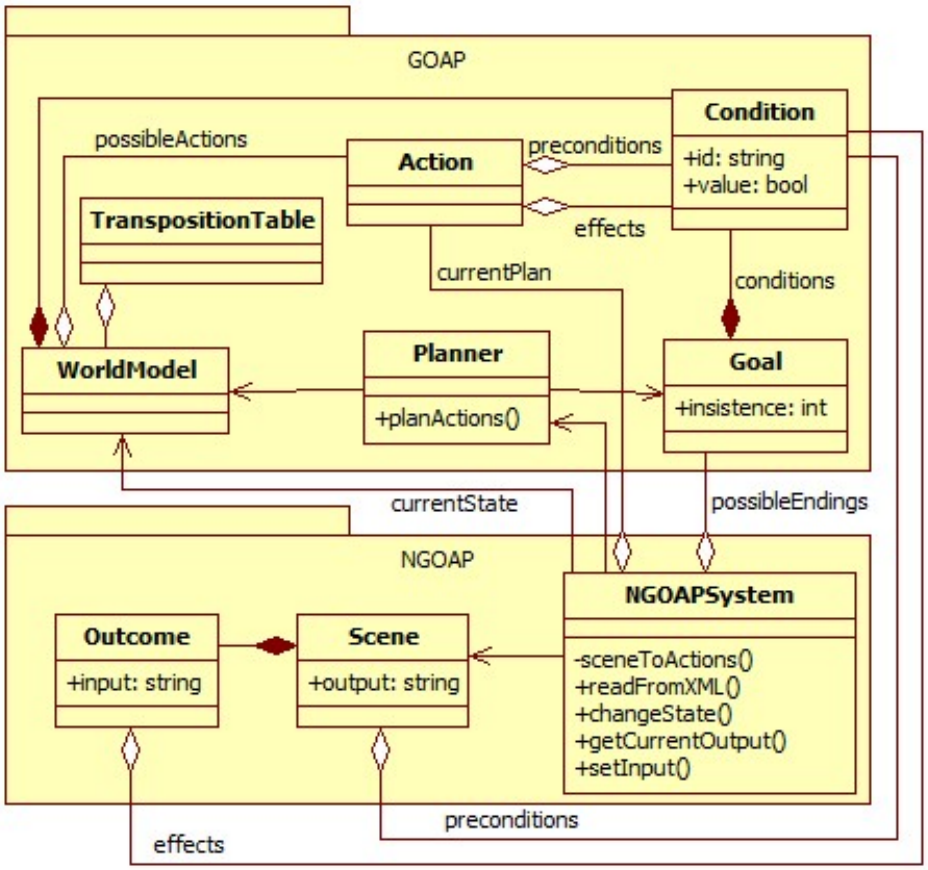

Abb. 3.1. Klassendiagramm

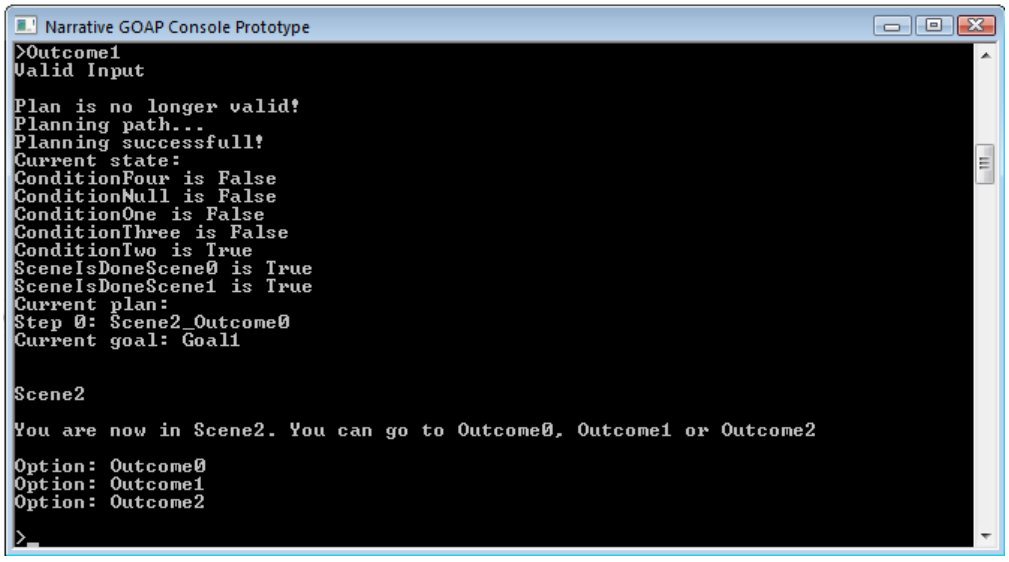

Abb. 3.2. Der Prototyp in Aktion

4

Ergebnisse

4.1 Effekte

Wenn das System in einem Spiel implementiert wurde, ist die Benutzung seitens eines Autors einfach. Er muss Enden und Szeneninhalte schreiben und am wichtigsten: Es muss Handlung in all den kausalen Zusammenhängen gesehen werden, nicht als zeitliche lineare Abfolge. Wenn er die Bedingungen und Effekte richtig definiert hat, übernimmt das System den Rest. Der Autor kann dann mittels einer einfachen Umsetzung – wie dem hier entwickelten Prototypen – seine Arbeit unabhängig der Entwicklung der Spiellogik testen. So kann zum Beispiel überprüft werden, ob einzelne Enden gar nicht angesteuert werden (weil sie durch den derzeitigen Pool an Szenen nicht erreicht werden können) oder Sackgassen entstehen.

Wenn bestimmte Szenen mehrmals vorkommen dürfen, können auch Schleifen entstehen, aus denen nur bestimmte Entscheidungen wieder heraus führen. Auch wenn dies im ersten Moment irritierend sein könnte, ist dies ein deutliches Hinweis an den Spieler, dass er sich an irgendeiner Stelle anders entscheiden muss, um die Handlung voranzutreiben. Der Algorithmus ist an dieser Stelle auf den Spieler angewiesen, das System kann sich nicht über ihn hinwegsetzen. So wird der Spieler nicht entmündigt. Jede Entscheidung hat eine Bedeutung – selbst wenn der Autor effektlose Szenen eingebaut hat, wird der Algorithmus diese einfach nicht benutzen, da sie nicht näher zum Ziel führen.

Szenen, die für ein vorheriges Ziel notwendig waren, aber ihre Bedeutung verloren haben, entsprechen einem retardierenden Moment oder können als ein so genannter *red herring* verstanden werden.

Es wird nicht nur die narrative Integrität (im Sinne vom Vermeiden von plot holes) gewahrt, es kann auch die maximale Frequenz der Interaktivität erreicht werden. Das bedeutet, dass jede Szene mit einer Interaktionsmöglichkeit für den Spieler einhergeht und auf jede Interaktion des Spielers eine dazu passende weitere Entwicklung der Handlung folgt. So wechseln sich also Story und Spielereingriff regelmäßig ab, ohne dem Spieler Einfluss zu verwehren (vorausgesetzt, die Szenen stellen die kleinstmöglichen Handlungsveränderungen dar).

4.2 Verwendung

Um mit der Nutzung des Systems die gewünschten Effekte zu erzielen, muss beim Erstellen der Inhalte auf ein paar Dinge geachtet werden.

Transparenz

Wichtig für die Qualität der Handlungsinhalte ist, dass bei den logischen Zusammenhängen eine gewisse Transparenz besteht. Das bedeutet, dass dem Spieler der Effekt seiner Entscheidungen immer ersichtlich sein sollte. Um dies sicherzustellen, sollte die Zeit zwischen der Eintscheidung und dem Eintreten der dadurch bedingten Veränderung nicht zu groß sein. Da die Spieler in der Regel ihren Spielstand nicht oft genug speichern (oder ältere Spielstände überschreiben), ist es sehr ärgerlich, wenn eine Entscheidung, die am Anfang getroffen wird, dafür sorgt, dass ganz am Ende (womöglich mehrere Stunden Spielzeit später) ein Weg nicht mehr eingeschlagen werden kann, weil jetzt erst klar geworden ist, welche vorherigen Entscheidungen dafür notwendig sind. Also ist es wichtig, auch während des Spielverlaufs mehrere Möglichkeiten zu geben, frühere Effekte wieder zu revidieren um wieder die Möglichkeit zu bekommen, zu einem vorher eingeschlagenen Pfad zurückzukehren, da viele Spieler keine Lust haben, große Teile des Spiels zu wiederholen, nur um eine einzelne Fehlentscheidung wieder gut zu machen. So kann das aus dem Game-Design bekannte Konvexitätsprinzip befolgt werden.

Außerdem müssen die Effekte auch einen tatsächlichen Zusammenhang mit der vorhergehenden Szene und der getroffenen Entscheidung haben. Diese Zusammenhänge sollten dem Spieler auch immer sichtbar gemacht werden. Zum Beispiel sollte dem Spieler in einem Dialog vermittelt werden, dass wenn er sich dafür entscheidet, einem bestimmten Charakter zu helfen, sich sein Verhältnis zu diesem Charakter feindlich gesonnenen NPC verschlechtern kann. In dem Sinne gilt auch besondere Vorsicht in der Verwendung von versteckten Zusammenhängen. So macht es zwar womöglich storytechnisch Sinn, dass der Inhalt eines Tresors durch eine Falle zerstört wird wenn dieser nicht mit dem richtigen Schlüssel geöffnet wird, es ist aber ärgerlich für den Spieler, dies erst herauszufinden, wenn es schon zu spät ist.

Noch verwirrender sind Entscheidungen, bei denen mehrere Optionen denselben Effekt besitzen. Entscheidungen ohne Effekt sollten nur dann eingesetzt werden, wenn dies für den Spieler eine Möglichkeit darstellt, diese Entscheidung noch nicht zu treffen und er zu einem späteren Zeitpunkt wieder zu ihr zurückkehren kann.

Persönliche Strategien

Diese Grundsätze sind notwendig, damit der Spieler eine Strategie entwickeln kann, was sehr wichtig für das Meistern eines Spiels ist. Der Spieler sollte früh lernen können, wie weitreichend die Effekte der Entscheidungen sind, welche Ziele er erreichen kann und wie er sich entscheiden muss, um zu bestimmten Stellen, die ihm in Aussicht gestellt worden sind, zu gelangen.

Es ist immer eine gute Idee, dem Spieler lieber viele kleine Entscheidungsmöglichkeiten geben, die die Handlung nur minimal verändern, aber in der Summe zu

sehr unterschiedlichen Ergebnissen führen kann, als nur ein paar wenige Stellen zu haben, an denen der Verlauf der Geschichte beeinflusst werden kann.

Dabei müssen immer genug verschiedene Auswirkungen einer einzelnen Entscheidungen vorhanden sein, so dass sich alle Spieler mit ihren verschiedensten Persönlichkeiten darin wiederfinden können. Das bewirkt eine bessere Identifikation mit dem Spiel, als wenn der Spieler nur zwischen zwei oder drei Möglichkeiten entscheiden kann, die ihm gleichermaßen unangemessen erscheinen.

Gut ist langweilig

Um einen möglichst großen dramatischen Effekt zu erzielen, sollte darauf geachtet werden, dass die Entscheidungen nicht zu leicht fallen. Es ergibt wenig Sinn, den Spieler entscheiden zu lassen, ob er sterben oder weiterspielen möchte. Entscheidungen werden dann besonders interessant, wenn sie ein großes Spektrum von moralischen Grauzonen ausschöpfen. Nach dem Motto „gut ist langweilig“ sollten dem Spieler schwierige Entscheidungen auferlegt werden, bei denen es nicht eine „richtige“ Option gibt, sondern mehrere, die unterschiedliche und weitreichende Konsequenzen haben. Dies kann weiter verstärkt werden, indem durch eine besonders riskante Entscheidung seitens des Spielers der Pfad stärker und öfter verändert wird, wodurch die Konsequenzen dieser Entscheidung in den nächsten Szenen verarbeitet werden können. Dadurch wird dafür gesorgt, dass interessantere Entscheidungen mit einem längeren und tiefgreifenderen Spiel belohnt werden.

4.3 Einschränkungen

Dennoch hat dieses System gewisse Einschränkungen. Der Spieler hat bei einer Entscheidung immer nur die Optionen, die ihm der Designer beziehungsweise Autor gewährt. Dabei sind für ihn in der Regel immer alle Möglichkeiten sichtbar. Der Spieler kann keine eigenen, kreativen Lösungen finden, mit denen der Autor nicht gerechnet hat, ihm steht immer nur eine begrenzte Auswahl zur Verfügung. Es gibt nie die Möglichkeit für eine Art von „Plan C“. Als Beispiel: Wenn dem Spieler von einem Antagonisten die Entscheidung auferlegt wird, entweder selber zu sterben, oder seinen Freund sterben zu sehen, so wird er nicht die Möglichkeit haben, sich im Versuch beide Leben zu retten Hals über Kopf auf ihn zu stürzen, wenn dies vom Autor nicht vorgesehen war. Und falls es vom Autor vorgesehen war, so wird dem Spieler durch die Umsetzung der Ausgabe direkt die Möglichkeit sichtbar sein und er sie natürlich direkt ergreifen, ohne die anderen überhaupt zu beachten. Die einzig mögliche Art, kreative Lösungen zu erlauben, ist, nur einfache Bedingungen (wie `GegnerIstTot`) zu verwenden und dem Spieler durch diverse Spielmechaniken zu erlauben, dorthin zu gelangen. Dabei kann der Spieler aber auch nur so einfallsreich sein, wie es die Simulation des Spiels hergibt.

Generell gefährdet die Freiheit des Spielers die *willing suspension of disbelief*. Es fehlt der wohlgeformte Spannungsbogen, der durch einen Autoren gezielt eingesetzt wird, um den Spieler so sehr zu fesseln, dass ihm gewisse Ungereimtheiten nicht

auffallen. Ein Spiel, das mit diesem System das Storytelling umsetzt wird nie so eine narrative Qualität aufweisen wie ein gut komponierter Film. Dies wird um so mehr der Fall sein, wenn dem Spieler mehr Freiheiten gegeben werden. Allerdings sollte der Gewinn an Spielspaß durch die Interaktivität ausreichend sein, um dies zu entschuldigen.

5

Ausblick

Das hier vorgeschlagene System ist nur ein Anfang. Ein erster Schritt, wie ohne größeren Aufwand das Storytelling in Spielen schon jetzt verbessert werden kann, um explizite author-story interaktiver zu machen. Es gibt aber noch viele weitere Möglichkeiten, wie das Potential von Computern genutzt werden kann, um mit algorithmischen Verfahren Storytelling interessanter zu machen. Dabei ist die Rechenleistung von modernen Maschinen noch lange nicht der begrenzende Faktor.

Wertung von Szenen

Es gibt noch einigen Spielraum, wie auch dieses System noch verbessert werden kann. GOAP bietet die Möglichkeit, Actions für den Planungsalgorithmus mit verschiedenen Kosten zu versehen, um bestimmte Aktionenssequenzen zu bevorzugen. Durch diesen unterschiedlichen Wert von Aktionen kann dafür gesorgt werden, dass bestimmte Pfade eher eingeschlagen werden. So kann beispielsweise dafür gesorgt werden, dass einzelne Szenen, die besonders wichtig für die Story sind, vor anderen, weniger interessanten Szenen angesteuert werden.

Dramenstruktur

Da es schwierig ist, mit der Planungs-KI einen Spannungsbogen zu erzeugen, muss man etwas improvisieren. Man könnte die Szenen in solcher mit steigender und fallender Handlung teilen, um dann entsprechend dem Regeldrama erst einen Pfad für die Komplikation bis zur Peripetie zu planen und danach einen Pfad bis zum Dénouement. Also wird das System zwei mal genutzt: einmal für die Planung von der Exposition zum Höhepunkt, wobei hier verschiedene Höhepunkte angesteuert werden können. Und zweitens für die restliche Planung zum Schluss, wobei der vorher erreichte Höhepunkt den Startzustand darstellt, nur Szenen, die zur fallenden Handlung gehören, benutzt werden können und es wieder mehrere mögliche Enden gibt, die verschiedene Auflösungen darstellen.

Dynamische Bedingungen

Die Bedingungen in ihrer jetzigen Implementation als Boolean-Werte sind etwas einschränkend. Zwar können sie so am besten von Autoren benutzt werden, die

keinen Einblick in die Logik des Spiels haben, aber sie machen es schwierig, dynamische Zusammenhänge zu beschreiben. Es wäre eine deutliche Verbesserung und Vereinfachung der Benutzung, wenn eine symbolische Repräsentation möglich wäre, wie zum Beispiel eine Bedingung der Form `SpielerIstAmOrtX`, wobei X eine Variable ist und nicht statisch festgelegt sein muss.

Nebenhandlung

Ein weiterer möglicher nächster Schritt, der aber nicht ganz so einfach umzusetzen ist, wäre verschiedene untergeordnete Nebenstories zu unterstützen, wie dies aus Spielen mit Quests bekannt ist. Dafür müsste eine Hierarchie für Enden eingeführt werden, also zumindest in eine Haupt- und sonstige Nebenhandlungen getrennt werden. Die Schwierigkeit hierbei besteht darin, einen Weg zu finden, dass der Planer bereits begonnene Nebenstories auch dann abschließt, wenn das Ende der Hauptstory schon nahe liegt. Anderenfalls würde die Handlung der Nebenstory unaufgelöst bleiben.

Player-Model

Es wurden schon viele verschiedene Ansätze vorgeschlagen, die zwar in eine andere Richtung gehen, aber problemlos mit dem hier entwickelten System kombiniert werden können. So wurden in den letzten Jahren viele Lösungen für interactive drama vorgeschlagen, die man problemlos in Spiele einfließen lassen könnte.

Es wurden etwa Ansätze entwickelt ([TBSW07], [YR14]) ein personalisiertes Spielerlebnis zu bewirken, indem der Planungsalgorithmus (dort als „Drama Manager“ bezeichnet) ein „player-model“ erzeugt, das die Präferenzen des Spielers für das Storytelling modelliert. Damit können dann die vorgeschlagenen Handlungsverläufe automatisch an die Spielerpersönlichkeit angepasst werden.

Intervention

Ein vorgeschlagenes Verfahren ([You99]) mit dem verhindert werden kann, dass Spieleraktionen die narrative Integrität gefährden, wäre, bei einer unerwarteten Aktion des Spielers zu intervenieren, anstatt neu zu planen, wie es bei dem hier entwickelten System geschieht. Dies könnte zum Beispiel dadurch geschehen, die Aktion des Spielers auf eine Weise scheitern zu lassen, die für ihn storytechnisch plausibel erscheint.

Literaturverzeichnis

AA06. Adams, Tarn und Adams, Zach: *Dwarf Fortress*, 2006.

Aar97. Aarseth, Espen J.: *Cybertext: Perspectives on Ergodic Literature*. John Hopkins University Press, 1997.

BB84. Braben, David und Bell, Ian: *Elite*, 1984.

BC12. Baikadi, Alok und Cardona-Rivera, Rogelio E.: *Towards Finding the Fundamental Unit of Narrative: A Proposal for the Narreme*, 2012. Third Workshop on Computational Models of Narrative.

bio07. *BioShock*, 2007. 2K Games.

bw01. *Black & White*, 2001. Lionhead Studios.

Car07. Carroll, Noël: *Narrative Closure*. Philosophical Studies, (35), 2007.

cod03. *Call of Duty*, 2003. Infinity Ward.

Cra12. Crawford, Chris: *On Interactive Storytelling*. New Riders, 2012.

Cro76. Crowther, William: *Adventure*, 1976.

cyo88. *Choose Your Own Adventure Series*. Bantam Books, 1979–1988.

fab04. *Fable*, 2004. Big Blue Box.

fc312. *Far Cry 3*, 2012. Ubisoft Montreal.

fea05. *F.E.A.R.*, 2005. Monolith Productions.

ff87. *Final Fantasy*, 1987. Square.

For27. Forster, Edward Morgan: *Aspects of the Novel*. Edward Arnold, 1927.

Fuh94. Fuhrmann, Manfred: *Aristoteles: Poetik. Griechisch/Deutsch*. Reclam, 1994.

GA74. Gygax, Gary und Arneson, Dave: *Dungeons & Dragons*, 1974.

Gen72. Genette, Gérard: *Figures I–III*, 1966–1972.

gow06. *Gears of War*, 2006. Epic Games.

hl204. *Half Life 2*, 2004. Valve Corporation.

hr10. *Heavy Rain*, 2010. Quantic Dream.

Joy90. Joyce, Micheal: *Afternoon: A Story*, 1990.

Kus03. Kushner, David: *Masters Of Doom*, Kapitel 8, Seite 120. Random House, 2003.

Lau86. Laurel, Brenda Kay: *Toward the Design of a Computer-Based Interactive Fantasy System*. Doktorarbeit, Ohio State University, 1986.

me07. *Mass Effect*, 2007. BioWare.

mg408. *Metal Gear 4*, 2008. Kojima Productions.

MI90. *The Secret of Monkey Island*, 1990. LucasArts.

mys93. *Myst*, 1993. Cyan.

Ork04. Orkin, Jeff: *Symbolic Representation of Game World State: Toward Real-Time Planning in Games.* In: *Proceedings of the AAAI Workshop on Challenges in Game AI*, 2004.

Ork06. Orkin, Jeff: *Three States and a Plan: The AI of F.E.A.R.* In: *Proceedings of the Game Developers Conference*, 2006.

Pon72. *Pong*, 1972. Atari Inc.

sim13. *The Sims Series*, 2000–2013. Maxis.

TBSW07. Thue, David, Bulitko, Vadim, Spetch, Marcia und Wasylishea, Eric: *Interactive Storytelling: A Player Modelling Approach.* Technischer Bericht, University of Alberta, 2007.

tes12. *The Elder Scrolls Series*, 1994–2012. Bethesda Game Studios.

Tom25. Tomaschewski, Boris Wiktorowitsch: *Theorie der Literatur*, 1925.

TW80. Toy, Michael und Wichman, Glenn: *Rogue*, 1980.

wow05. *World of Warcraft*, 2005. Blizzard Entertainment.

You99. Young, R. Michael: *Notes on the Use of Plan Structures in the Creation of Interactive Plot.* Technischer Bericht, AAAI technical Report FS-99-01, 1999.

YR14. Yu, Hong und Riedl, Mark O.: *Personalized Interactive Narratives via Sequential Recommendation of Plot Points.* IEEE Transactions on Computational Intelligence and AI in Games, 2014.

A

Erklärung des Kandidaten

Die Arbeit habe ich selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Datum	Unterschrift der Kandidatin / des Kandidaten