Prozedurale Levelgenerierung für 2D Plattformspiele

Bernhard Handler

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Oktober 2012

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 4. Oktober 2012

Bernhard Handler

Inhaltsverzeichnis

Vorwort

Als ich vor mehr als 15 Jahren die Game Boys meiner Freunde mit *Super Mario Land* zum Glühen brachte, war mir noch nicht im Geringsten bewusst, dass ich bereits mit der Recherche für diese Arbeit begonnen hatte – wahrscheinlich genauso wenig wie meiner Mutter, die mir sonst vielleicht doch erlaubt hätte, einen eigenen Game Boy zu besitzen. Auch wenn ich also noch etwas länger warten musste, bis ich eine Spielkonsole mein Eigen nennen durfte, konnte ich mich in allen wichtigen Belangen auf die Unterstützung meiner Eltern und überhaupt meiner ganzen Familie verlassen, wofür ich mich an dieser Stelle aufrichtig bedanken möchte.

Ein großer Dank gilt auch meinem Betreuer Wolfgang Hochleitner, der durch seine konstruktiven Vorschläge und sein wertvolles Feedback einen wesentlichen Beitrag zum Entstehen dieser Arbeit geleistet hat.

Auch bei allen Teilnehmern der durchgeführten Benutzerbefragung möchte ich mich herzlich bedanken. Durch ihre ehrliche Meinung konnten aufschlussreiche Daten zur Verbesserung und Weiterentwicklung des vorgestellten Systems gesammelt werden.

Zu guter Letzt danke ich all meinen Freunden und Kollegen, die durch ihre kontinuierliche Ermutigung dafür gesorgt haben, dass ich mein Ziel trotz so mancher Sackgasse, in die der Entstehungsweg dieser Arbeit geführt hat, nie aus den Augen verloren habe.

Kurzfassung

Im herkömmlichen Entwicklungsprozess eines Plattformspiels nimmt die manuelle Erstellung der Levels durch Leveldesigner einen wesentlichen Teil der verfügbaren Ressourcen in Anspruch. Die vorliegende Arbeit stellt daher einen Ansatz vor, mit dem Levels für 2D Plattformspiele prozedural generiert werden können. Die Generierung erfolgt in einem zweistufigen Verfahren: In der ersten Stufe wird ein möglicher Lösungsweg generiert, indem zufällig parametrisierte Spieleraktionen aneinandergereiht werden. Erst in der zweiten Stufe wird die eigentliche Levelgeometrie erzeugt, indem vordefinierte Grundelemente so variiert und in das Level eingepasst werden, dass sie den vorgegebenen Lösungsweg implizieren. Diese Vorgehensweise garantiert, dass alle generierten Levels tatsächlich spielbar sind.

Der entwickelte Lösungsansatz wurde exemplarisch in einem Plattformspiel umgesetzt und mittels einer Benutzerbefragung evaluiert. Die Evaluierungsergebnisse zeigen, dass die definierten Ziele erreicht werden konnten und der vorgestellte Ansatz in der Lage ist, eine praktisch unbegrenzte Vielzahl an spielbaren Levels für ein Plattformspiel zu erzeugen.

Abstract

The conventional process of developing a platform game requires that a substantial amount of the available resources is used for the manual creation of levels by level designers. The present work therefore introduces an approach that allows the procedural generation of levels for 2D platform games. The generation follows a two-tier procedure: Firstly, a possible solution path is generated by stringing together randomly parameterized player actions. Secondly, the actual level geometry is created by modifying basic predefined elements and fitting them into the course of the level to imply the given solution path. This procedure guarantees that all generated levels are fully playable.

The devised approach was implemented in an example platform game and evaluated by conducting a user study. The results of the evaluation show that the defined objectives could be met and that the presented solution is capable of generating a virtually unlimited number of playable platformer levels.

Kapitel 1

Einleitung

1.1 Motivation

Die Entwicklung eines Computerspiels ist ein ressourcenintensiver Prozess, der im Extremfall Teams mit hunderten Mitarbeitern über mehrere Jahre hinweg beschäftigen kann. Die prozedurale Generierung von Spielinhalten, d. h. ihre algorithmische Erzeugung durch eine Computerprozedur, stellt eine vielversprechende Möglichkeit dar, den manuellen Arbeitsaufwand einer Spieleproduktion zu reduzieren bzw. auf andere Aspekte der Entwicklung zu fokussieren. Die Vision eines Systems, das eine praktisch unbegrenzte Vielfalt an Inhalten in einem Bruchteil jener Zeit erzeugen kann, die ein Mensch dafür benötigt, hat bereits einige Forschungsarbeiten auf diesem Gebiet angetrieben. Auch in der vorliegenden Arbeit soll ein Teilgebiet aus dem großen Bereich der prozeduralen Generierung näher betrachtet werden: Die Generierung von Levels speziell für das Genre der 2D Plattformspiele (oft auch als Jump 'n' Runs bezeichnet).

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, das Leveldesign von bestehenden 2D Plattformspielen zu analysieren und in weiterer Folge einen Lösungsansatz zu entwickeln, der in der Lage ist, Levels für ein solches Spiel prozedural zu generieren. Die Beschränkung auf ein spezielles Genre ist notwendig, um die Levelgenerierung optimal an die Gegebenheiten dieses Genres anpassen zu können. Plattformspiele sind dabei besonders interessant, weil ihr grundsätzliches Spielziel im Durchqueren einer Reihe von Levels besteht und sich daher auch die Spielerfahrung vorrangig aus dem Leveldesign ableitet.

Wichtige Zielvorgaben, die der Lösungsansatz zur Levelgenerierung unbedingt erfüllen soll, sind die Sicherstellung der Spielbarkeit aller generierten Levels, ein möglichst geringer Rechenaufwand, um die Generierung dynamisch zur Laufzeit ausführen zu können, und die weitestgehende Unabhän-

gigkeit von manuellen Eingriffen im Generierungsprozess. Der entwickelte Ansatz soll exemplarisch in einem Plattformspiel umgesetzt und mittels einer Benutzerbefragung evaluiert werden, um mögliche Probleme zu identifizieren und in weiterführenden Arbeiten behandeln zu können.

1.3 Aufbau dieser Arbeit

Nach diesem einleitenden Kapitel beginnt die Arbeit mit einer Beschreibung des Genres der Plattformspiele in Kapitel 2. Neben einer Begriffserklärung werden die typischen Elemente eines Plattformspiels angeführt und erläutert. Außerdem werden exemplarisch einige bedeutende Plattformspiele vorgestellt, die für die Entwicklung des Genres prägend waren.

Kapitel 3 beschäftigt sich im Anschluss eingehend mit dem Thema der prozeduralen Generierung von Spielinhalten. Nach einer Definition dieses vielschichtigen Begriffes wird ein Überblick über die wichtigsten Anwendungsgebiete und Methoden gegeben. Auch der Einsatz in existierenden Spielen wird anhand aussagekräftiger Beispiele aufgezeigt.

Nach diesen allgemeinen Ausführungen zur prozeduralen Generierung von Spielinhalten erfolgt in Kapitel 4 eine Beschreibung von Ansätzen, die sich speziell mit der prozeduralen Levelgenerierung für Plattformspiele auseinandersetzen und daher für die vorliegende Arbeit besonders relevant sind.

Kapitel 5 stellt schließlich den Hauptteil der Arbeit dar, in dem der eigene Lösungsansatz im Detail erläutert und schematisch illustriert wird. Angefangen von den definierten Systemanforderungen und Grundannahmen finden sich hier Erklärungen zu den einzelnen Schritten des Generierungsprozesses bis hin zu einer Beschreibung der technischen Umsetzung.

Kapitel 6 widmet sich dann der Evaluierung des entwickelten Ansatzes. Hier werden Methodik und Ablauf der durchgeführten Benutzerbefragung erklärt, bevor die Auswertungsergebnisse dargestellt und diskutiert werden.

In Kapitel 7 erfolgt schlussendlich ein Resümee über die gesamte Arbeit und ein Ausblick über mögliche zukünftige Entwicklungen im Bereich der prozeduralen Generierung von Spielinhalten im Allgemeinen und im Bereich der vorgestellten Levelgenerierung für Plattformspiele im Speziellen.

Kapitel 2

Plattformspiele

2.1 Definition

In der Literatur finden sich wenige Definitionen, die sich ausdrücklich und ausführlich mit dem Begriff des Plattformspiels[1] auseinandersetzen. In den meisten Klassifizierungen werden Plattformspiele einer größeren Gruppe von aktionsorientierten Spielen zugeordnet und nicht im Detail behandelt. Falls Plattformspiele explizit erwähnt werden, bleiben die Ausführungen meist kurz. Wolf [35, S. 270] definiert sie beispielsweise als

> games in which the primary objective requires movement through a series of levels, by way of running, climbing, jumping, and other means of locomotion.

Ähnlich charakterisiert Koster [13, S. 232] das Genre:

> Platform games: Any of a broad class of games where you attempt to traverse a landscape collecting objects or touching every space on the map.

Eher beiläufig beschreiben Salen und Zimmerman den Ablauf von Plattformspielen in [25, S. 523]:

> These games allow players to struggle against obstacles, explore fantastic lands, fight menacing enemies, and even die.

Zentrale Merkmale von Plattformspielen sind also das Durchqueren der Spielwelt und die dazu notwendige Überwindung von Hindernissen durch präzise abgestimmte Bewegungsmanöver (hauptsächlich durch Springen und Laufen, wovon auch die Bezeichnung Jump 'n' Run abgeleitet ist).

[1] Im deutschen Sprachgebrauch wird häufig auch die Bezeichnung *Jump 'n' Run* für diese Art von Computerspielen verwendet. Obwohl aus dem Englischen entlehnt, ist dieser Begriff dort wenig gebräuchlich. Weitaus üblicher ist die Bezeichnung *Platform Games*, deren deutsche Entsprechung *Plattformspiele* auch in dieser Arbeit vorrangig Anwendung findet.

2.2 Typische Elemente

Analysiert man unterschiedliche Plattformspiele, so kristallisiert sich schnell eine Reihe von wiederkehrenden Spielelementen heraus, die typisch für das Genre sind. Die Kenntnis dieser genredefinierenden Elemente ist essentiell, da sie die Grundbausteine für das Leveldesign und somit auch für das in dieser Arbeit vorgestellte System zur prozeduralen Levelgenerierung bilden. Die folgenden Abschnitte beschreiben daher die grundlegenden Komponenten von Plattformspielen in Anlehnung an die von Smith u. a. vorgenommene Kategorisierung [30].

2.2.1 Spieleravatar

Der Spieleravatar bezeichnet die visuelle und physikalische Repräsentation des vom Spieler gesteuerten Charakters in der Spielwelt. Die grundlegenden Aktionen, die der Spieler über seinen Avatar ausführen kann, sind in der Regel das Laufen und Springen. In vielen Plattformspielen werden dem Avatar allerdings zusätzlich zu diesen Basisaktionen weitere Fähigkeiten eingeräumt, beispielsweise die Fähigkeit, im Sprung einen weiteren Sprung auszuführen (Doppelsprung), von Wänden abzuspringen, auf Gegner zu schießen, zu schwimmen, zu reiten oder gar die Zeit zu beeinflussen.

Oft steuert der Spieler über den gesamten Spielverlauf denselben Avatar, manche Plattformspiele bieten jedoch auch mehrere spielbare Charaktere. Die Unterschiede zwischen den einzelnen Charakteren können dabei allein in ihrer grafischen Repräsentation oder auch in ihren spielrelevanten Aktionsmöglichkeiten liegen. Einige Spiele generieren zusätzliche Spieltiefe, indem sie das Wechseln zwischen den Charakteren zu einer Aktion im Spiel machen. Um ein Level erfolgreich zu absolvieren, müssen dann die unterschiedlichen Charakterfähigkeiten kombiniert und koordiniert werden.

Natürlich hat der Aktionsspielraum des Spieleravatars (oder der Spieleravatare) entscheidenden Einfluss auf das Leveldesign. Die Spielumgebung muss auf die Handlungsmöglichkeiten des Spielers abgestimmt sein und soll den geschickten Einsatz der zur Verfügung stehenden Fähigkeiten fordern.

2.2.2 Plattformen

Plattformen sind das namensgebende Element für das Genre der Plattformspiele, dementsprechend zentral sind sie für den Aufbau eines solchen Spiels. Unter dem Sammelbegriff Plattform werden alle Oberflächen in der Spielwelt verstanden, auf denen der Spieler stehen und laufen kann. Diese Definition ist natürlich sehr weit gefasst, entsprechend groß ist das Spektrum an unterschiedlichen Plattformtypen, die in gängigen Plattformspielen beobachtet werden können. Im Standardfall handelt es sich um statische Objekte, die häufig ebene, genauso aber auch beliebig geneigte Oberflächen aufweisen.

Daneben gibt es eine Reihe von dynamischen Plattformen, die zusätzlich den Faktor Zeit ins Spiel bringen: Plattformen, die sich auf vordefinierten Pfaden hin- und herbewegen, Plattformen, die nur in bestimmten Zeiträumen gefahrlos betreten werden können, und Plattformen, die sich nach dem Betreten auflösen, sind nur einige der vielzähligen Varianten.

Unterschiede können außerdem in den physikalischen Eigenschaften der Plattformflächen liegen. So beeinflussen beispielsweise unterschiedliche Reibungskoeffizienten die Bewegung des Spieleravatars – rutschige Eisflächen oder klebrige Teerböden sorgen für wechselnde Bedingungen, auf die der Spieler reagieren muss.

2.2.3 Hindernisse

Neben Spieleravatar und Plattformen sind Hindernisse die dritte wesentliche Komponente, die in sämtlichen Plattformspielen anzutreffen ist. Erst durch die Hindernisse in der Spielwelt wird ein Plattformspiel komplettiert, da sie dem Spieler eine Aufgabe geben: Die Überwindung ebendieser Hindernisse und das dadurch ermöglichte Erreichen eines bestimmten Zielpunktes im Level.

Prinzipiell bezeichnet der Begriff Hindernis alle Objekte in der Spielwelt, die dem Spieler Schaden zufügen können und deshalb gemieden werden müssen. Ebenso vielfältig wie die unterschiedlichen Plattformtypen fallen daher die zu beobachtenden Arten von Hindernissen aus. Auch hier kann grundsätzlich zwischen statischen Hindernissen (z. B. unbewegliche Stacheln) und nicht-statischen Hindernissen (z. B. Gegner, die entlang vordefinierter Wegpunkte patrouillieren) unterschieden werden. Während viele Hindernisse nur überwunden werden können, indem der Spieler ihnen durch gut koordinierte Bewegungen ausweicht, können andere auch gezielt zerstört werden. Dazu stehen dem Spieler entweder spezielle Angriffsaktionen zur Verfügung, oder das Spiel verlässt sich auf seine Kernmechanik und macht das Eliminieren von Hindernissen zu einer weiteren Bewegungsaufgabe. Eine häufig vorkommende Möglichkeit Gegner auszuschalten besteht beispielsweise darin, gezielt auf sie zu springen.

2.2.4 Bewegungshilfen

In diese Kategorie fallen Spielobjekte, die die Bewegungsmöglichkeiten des Spielers modifizieren. Beispiele dafür sind etwa Leitern, die der Spieler hinauf- und hinunterklettern kann, Seile, an denen der Spieler schwingen kann, und Trampoline, die dem Spieler zusätzliche Sprungkraft verleihen. Festzuhalten ist dabei, dass all diese Objekte in der Regel an einen Ort gebunden sind und den Bewegungsspielraum des Spielers nicht grundsätzlich, sondern nur für die Dauer der Interaktion mit dem jeweiligen Objekt verändern. Andernfalls wären sie als zusätzliche Fähigkeit des Spieleravatars einzuordnen.

2.2.5 Einzusammelnde Objekte

Während das primäre Ziel eines Plattformspiels ist, das jeweilige Levelende zu erreichen, stellt das Einsammeln von bestimmten in der Spielwelt verteilten Objekten oft ein sekundäres Spielziel dar. Die Objekte können dabei eine Vielzahl unterschiedlicher Formen annehmen, eine bekannte Repräsentation sind beispielsweise Goldmünzen. Durch das Einsammeln erhält der Spieler in der Regel gewisse Belohnungen, wobei wiederum unterschiedlichste Belohnungssysteme denkbar sind. Manche Spiele vergeben Punkte in Abhängigkeit von der Anzahl eingesammelter Objekte, andere belohnen den Spieler mit zusätzlichen Fähigkeiten oder zusätzlicher Lebensenergie.

Neben dieser für die Spielermotivation wichtigen Belohnungsfunktion können die einzusammelnden Objekte auch als Orientierungshilfe dienen, indem sie entlang wichtiger Pfade platziert werden. So können sie etwa auf den weiteren Weg durch ein Level weisen oder die optimale Sprungkurve zur Überwindung eines Hindernisses anzeigen. Sonst erfolgt die Platzierung der Objekte oft so, dass der Spieler für riskante Bewegungsmanöver oder das Auffinden versteckter Gebiete belohnt wird.

2.2.6 Trigger

Unter dem Begriff Trigger werden alle Objekte zusammengefasst, mit denen der Spieler interagieren kann, um den Zustand der Spielwelt zu beeinflussen. Exemplarisch dafür ist etwa ein Schalter, der betätigt werden muss, um ein Tor zu öffnen und den weiteren Weg durch ein Level freizugeben. Auch hier kann der Faktor Zeit eingesetzt werden, um die Herausforderung für den Spieler zu erhöhen – oft hält die ausgelöste Zustandsänderung nämlich nur für einen begrenzten Zeitraum an und zwingt den Spieler so zu reaktionsschnellem Handeln. Außerdem können Abhängigkeiten zwischen einzelnen Schaltern das Auslösen in einer bestimmten logischen Abfolge notwendig machen und somit ein Rätselelement in das sonst eher auf Geschicklichkeit ausgelegte Genre einführen.

2.2.7 Speicherpunkte

Speicherpunkte in der Spielwelt dienen dazu, den Fortschritt des Spielers festzuhalten. Sollte der Spieler ein Leben verlieren, wird er an den letzten passierten Speicherpunkt zurückgesetzt und muss nicht das gesamte Level erneut absolvieren. Die Platzierung der Speicherpunkte kann dabei großen Einfluss auf die empfundene Schwierigkeit eines Levels haben. Je größer die Distanz zwischen den einzelnen Punkten ausfällt, desto größer ist der potenzielle Rückschlag nach einem Tod des Spieleravatars und desto höher ist folglich auch das Fehlerpotenzial, wenn ein längerer Levelabschnitt wiederholt werden muss. Manche Plattformspiele verzichten auf den Einsatz von Speicherpunkten und reduzieren stattdessen die Levelgröße, sodass es dem

Spieler zugemutet werden kann, jeweils ein gesamtes Level ohne Fehler durchzuspielen.

2.3 2D und 3D Spielwelten

Eine wesentliche Unterscheidung innerhalb des Genres der Plattformspiele erfolgt nach der Dimensionalität der Spielwelt. Während sich die ersten Plattformspiele aufgrund technischer Restriktionen zwangsweise in zweidimensionalen Umgebungen abspielten, machte die stetig wachsende Leistung der Spielehardware bald auch dreidimensionale Welten möglich. Die Wahl zwischen 2D und 3D ist dabei keineswegs eine bloße Frage der grafischen Repräsentation, sie hat auch entscheidenden Einfluss auf den Spielablauf und das Leveldesign.

2D Plattformspiele zeigen das Spielgeschehen in der Regel aus einer fixen Seitenansicht, was das Abschätzen von Entfernungen erleichtert und dadurch der grundsätzlichen Spielmechanik – dem möglichst präzisen Springen von Plattform zu Plattform – entgegenkommt. Beim Übergang in eine dreidimensionale Umgebung sind Positionierung und Ausrichtung der Kamera nicht länger derart trivial. Durch die Perspektive entstehende Verzerrungen und Verdeckungen erschweren das gezielte Ausführen von Bewegungsaktionen. Einige 3D Plattformspiele stellen daher das Erkunden der freieren Spielwelt und weniger das fehlerlose Absolvieren eines linearen Hindernisparcours in den Vordergrund. Andere Spiele hingegen beschränken die Bewegung im 3D Raum auf vordefinierte Pfade, um so die Vorteile der Linearität klassischer 2D Levels beizubehalten.

Aufgrund der vergleichsweise großen Unterschiede zwischen 2D und 3D Plattformspielen ist es schwierig, allgemein gültige Regeln für das Leveldesign beider Spieltypen aufzustellen. Der in dieser Arbeit präsentierte Ansatz zur prozeduralen Levelgenerierung ist daher speziell für das Erstellen zweidimensionaler Spielumgebungen konzipiert.

2.4 Beispiele

Im Folgenden werden exemplarisch einige wichtige Plattformspiele erläutert, die bezeichnend für das gesamte Genre sind und dessen Entwicklung entscheidend mitgeprägt haben.

2.4.1 Donkey Kong

Donkey Kong (1981) wird oft als erstes Plattformspiel angesehen, obwohl die Grundzüge des Genres bereits im ein Jahr zuvor erschienenen *Space Panic* zu erkennen waren [4]. Die zentrale Neuerung in *Donkey Kong* bestand darin, dass der Spieler springen konnte, um Hindernisse und Abgründe zu

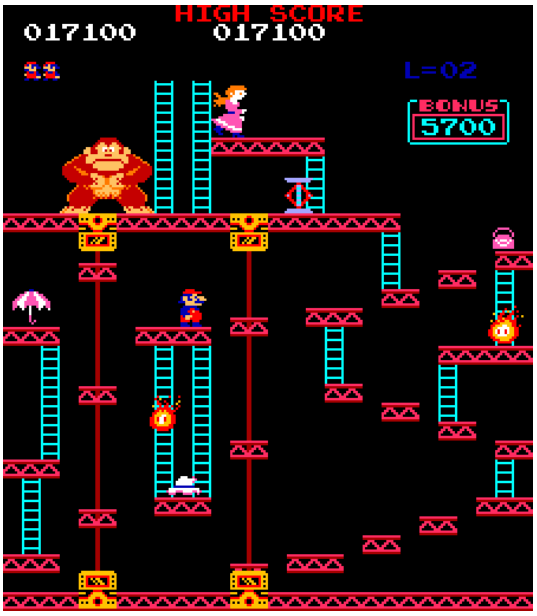


Abbildung 2.1: Spielszene aus *Donkey Kong* (1981) [37].

überwinden. Damit waren erstmals alle wesentlichen Elemente des Genres, wie wir es heute kennen, in einem Spiel vereint.

Aufgabe des Spielers ist es, in Form des Helden Jumpman seine Gefährtin aus den Fängen des namensgebenden Gorillas Donkey Kong zu befreien. Dazu muss eine Reihe von Plattformen erklommen werden und Gefahren wie rollenden Fässern und Feuer ausgewichen werden. Die Spielwelt ist dabei so aufgebaut, dass sie in ihrer Gänze auf einen Bildschirm passt – es gibt also noch kein Scrolling. Abbildung 2.1 zeigt eine Szene aus dem Spiel.

2.4.2 Super Mario

Super Mario ist die wohl bekannteste Plattformspielserie. Der 1985 veröffentlichte erste Titel der Serie – *Super Mario Bros.* – setzte den Grundstein für eine der erfolgreichsten Videospielreihen aller Zeiten [49].

Der Spieler verkörpert darin den Handwerker Mario, der auf der oben erwähnten Figur des Jumpman in *Donkey Kong* basiert. Laufend und springend bewegt sich Mario durch eine Reihe seitlich scrollender Levels, sammelt dabei Münzen und Power-Ups, weicht Hindernissen aus und kann Gegner ausschalten, indem er auf sie springt – das prototypische Spielprinzip aller nachfolgenden Plattformspiele. Auf den Erfolg des ersten Serienteils folgten und folgen bis heute zahlreiche Fortsetzungen, Weiterentwicklungen und Spin-offs. Abbildung 2.2 zeigt einen Ausschnitt aus dem Originalspiel von 1985.

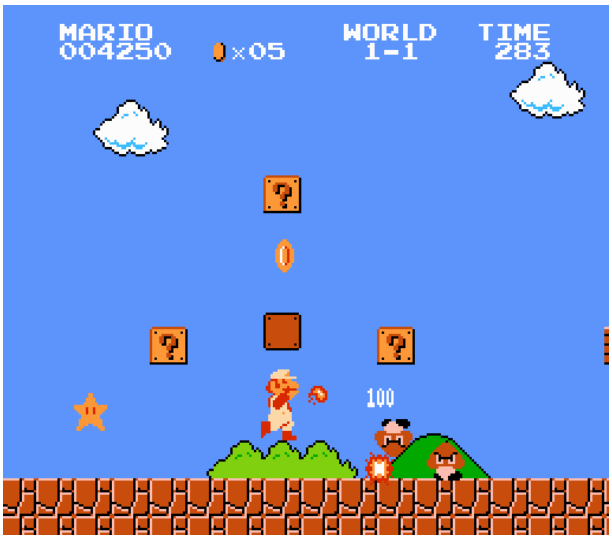


Abbildung 2.2: Spielszene aus *Super Mario Bros.* (1985) [38].

2.4.3 Crash Bandicoot

Bei *Crash Bandicoot* handelt es sich ähnlich wie bei *Super Mario* um eine ganze Serie von Plattformspielen. Das 1996 erschienene erste Spiel der Serie gilt zugleich als eines der ersten 3D Plattformspiele. Trotz der neuartigen 3D Umgebung wurde versucht, das bewährte Spielprinzip der klassischen 2D Vorlagen beizubehalten. Am bekannten Spielablauf änderte sich daher wenig: Der Spieler versucht, den Titelhelden Crash Bandicoot mit geschickten Bewegungsmanövern unbeschadet zum Levelende zu führen und dabei Früchte und Edelsteine einzusammeln (Abbildung 2.3).

Die Levels sind schlauchförmig angelegt und schränken die Bewegungsfreiheit des Spielers relativ stark ein, um im Gegenzug eine gute Übersicht im 3D Raum bieten zu können. Diese behutsame Adaption des erprobten Plattformspielprinzips auf eine zusätzliche Dimension machte die *Crash Bandicoot* Serie zu einem erfolgreichen Beispiel für 3D Plattformspiele [49].

2.4.4 LittleBigPlanet

LittleBigPlanet erschien 2008 für die Playstation 3 und hat sich mittlerweile mit Ablegern auf anderen Plattformen und der 2011 veröffentlichten Fortsetzung *LittleBigPlanet 2* ebenfalls zu einer eigenen Spieleserie entwickelt. Obwohl die Spiele 3D Grafik einsetzen, findet das Spielgeschehen in der Regel in einer 2D Ebene statt und wird aus der klassischen Seitenansicht gezeigt. Nur an einigen ausgewählten Stellen ist ein Ebenenwechsel notwendig. Das Spielgefühl und der grundsätzliche Levelaufbau unterscheiden sich daher kaum von herkömmlichen 2D Plattformspielen (Abbildung 2.4).

Abbildung 2.3: Spielszene aus *Crash Bandicoot* (1996) [39].

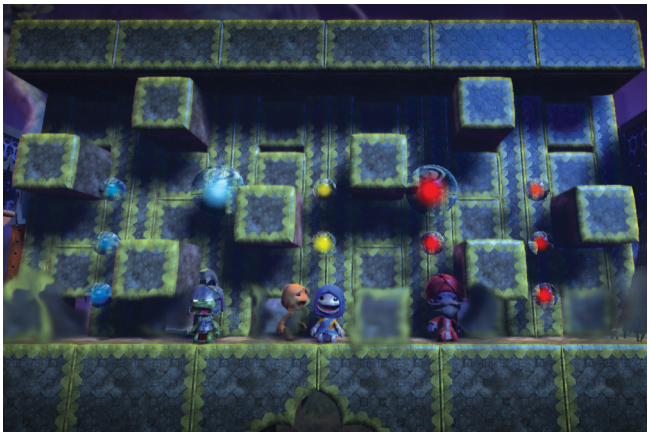


Abbildung 2.4: Spielszene aus *LittleBigPlanet* (2008) [40].

Die Besonderheit von *LittleBigPlanet* liegt in der abwechslungsreich gestalteten Spielwelt sowie im umfangreichen Leveleditor, der im Spiel enthalten ist und es den Spielern ermöglicht, eigene Levels zu erstellen und mit anderen zu teilen. Diese nutzergenerierten Inhalte führen zu einem noch größeren Variantenreichtum und befördern den Spieler vom bloßen Konsument zum Produzent.

Der Erfolg von *LittleBigPlanet* und *LittleBigPlanet 2* [49] zeigt, dass Plattformspiele nach wie vor ein beliebtes Genre darstellen, das trotz seiner langen Geschichte noch Raum für Innovationen bietet.

Kapitel 3

Prozedurale Generierung von Spielinhalten

3.1 Definition

Prozedurale Generierung im Allgemeinen und die prozedurale Generierung von Spielinhalten im Speziellen sind ein breites Forschungsgebiet, das sich über unterschiedliche Disziplinen erstreckt – von der Computergrafik über die Bildverarbeitung bis hin zur künstlichen Intelligenz. Dementsprechend schwierig ist es, eine exakte Definition des Begriffs aufzustellen. Hendrikx u. a. beschreiben das Gebiet folgendermaßen [9]:

> In contrast to manual content production, Procedural Content Generation for Games (PCG-G) is the application of computers to generate game content, distinguish interesting instances among the ones generated, and select entertaining instances on behalf of the players.

Die Definition von Togelius u. a. lautet etwas allgemeiner [34]:

> Procedural content generation (PCG) in games refers to the creation of game content automatically using algorithms.

Togelius u. a. räumen dabei aber sofort ein, dass eine genaue Abgrenzung des Begriffs unmöglich ist und ihre Definition nicht als universeller Grundsatz, sondern vielmehr als Diskussionsgrundlage verstanden werden soll.

Auch wenn sich der vielschichtige Begriff der prozeduralen Generierung also nicht auf eine allgemein gültige Definition reduzieren lässt, herrscht in den genannten Erklärungen Konsens über den zentralen Aspekt dahinter: Den Ersatz manueller Arbeit durch das Ausführen einer wohldefinierten Prozedur durch den Computer, um Spielinhalte zu erzeugen. Welche Inhalte das sein können und welche Methoden dabei Anwendung finden, wird in den folgenden Abschnitten erläutert.

(a) (b) (c)

Abbildung 3.1: Beispiele für Bestandteile einer virtuellen Spielwelt, die mit unterschiedlichen Werkzeugen prozedural erstellt wurden: mit *L3DT* generiertes Terrain (a) [41], mit *SpeedTree* generierte Vegetation (b) [42] und mit *CityEngine* generiertes Stadtmodell (c) [43].

3.2 Anwendungsgebiete

So lange es Computerspiele gibt, so lange gibt es den Wunsch, den Computer nicht nur als Kontrolleur der Spielregeln, sondern auch als Kreateur der Spielinhalte einzusetzen. Daher finden sich kaum Inhalte, die nicht bereits mittels prozeduralen Methoden zu erzeugen versucht wurden. Im Folgenden wird ein Überblick über die wichtigsten Anwendungsgebiete gegeben.

3.2.1 Spielwelt

Die Kreation virtueller Spielwelten ist ein Paradebeispiel für den Einsatz prozeduraler Generierungsmethoden. Natürlich stellen unterschiedliche Spiele sehr unterschiedliche Anforderungen an die Welt, in der sie spielen. Während abstrakten Rätselspielen mitunter ein schmuckloses Raster als Spielfeld reicht, versuchen andere Spiele, eine möglichst realitätsnahe Umgebung mit glaubhafter Landschaft, Vegetation und Bebauung zu erschaffen. Aufgrund der großen Ausdehnung und des hohen Detailgrades, der von solchen virtuellen Welten erwartet wird, stößt die manuelle Inhaltserstellung hier schnell an ihre Kapazitätsgrenzen. Diese Problematik hat viele Forschungstätigkeiten auf diesem Gebiet inspiriert und unterschiedliche prozedurale Lösungen hervorgebracht, die das Erstellen von Terrains, Vegetation, Straßennetzwerken, Siedlungsplänen und Gebäuden automatisieren [6, 18, 28]. Abbildung 3.1 zeigt exemplarisch die Ergebnisse solcher Ansätze.

3.2.2 Texturen

Texturen sind ein wesentliches Element, um grafische Details von Spielobjekten zu definieren. Auch hier haben sich viele Methoden etabliert, die Texturen algorithmisch und somit weitgehend ohne manuellen Aufwand erzeugen

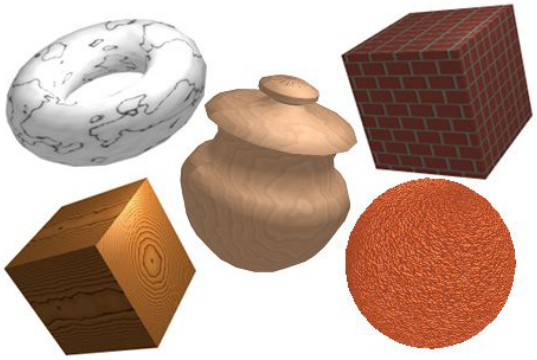


Abbildung 3.2: Prozedural texturierte Objekte [44].

können [7]. So werden beispielsweise Rauschfunktionen (oft Perlin-Noise [21]) verwendet, um die Muster von Materialien wie Marmor, Stein oder Holz synthetisch zu generieren (Abbildung 3.2). Daneben kann die Selbstähnlichkeit gewisser Strukturen ausgenutzt werden, um sie mittels prozeduraler Ansätze nachzubilden (z. B. Fliesenböden, Pflasterstraßen oder Ziegelmauern).

3.2.3 Animation

Auch für das Erstellen von Animationen existieren prozedurale Methoden [10, Kap. 6]. Dabei wird der Ablauf der Animation nicht wie herkömmlich durch manuell gesetzte Schlüsselbilder definiert, sondern aufgrund bestimmter Parameter von einer Computerprozedur berechnet. Das Haupteinsatzgebiet prozeduraler Animationstechniken ist die Simulation von komplexen physikalischen Phänomenen und Bewegungsabläufen. So werden zum Beispiel Partikelsysteme eingesetzt, um Feuer und Rauch darzustellen (Abbildung 3.3). Moderne Physikengines können außerdem die Bewegung von Kleidung und Stoffen realitätsnah berechnen. Auch die Dynamik von Haaren und Tierfellen ist mit prozeduralen Ansätzen effizienter abzubilden als mit manuellen Animationstechniken.

3.2.4 Sound

Die Erzeugung von Soundeffekten und Musik stellt ein weiteres Anwendungsgebiet prozeduraler Generierung dar. Der Fachbereich der algorithmischen Komposition beschäftigt sich etwa mit der automatischen Erstellung von Partituren [8]. Bereits Mozart wird die Entwicklung eines sogenannten „Musikalischen Würfelspiels“ zugeschrieben, das aufbauend auf einer Grundkomposition die zufällige Zusammenstellung einzelner Takte zu einer neuen Komposition ermöglicht. Auch die Klangsynthese, also das Erzeugen von Tönen mittels künstlich generierten Audiosignalen, kann als Form der prozeduralen

Abbildung 3.3: Partikelsystem zur prozeduralen Animation von Feuer [45].

Generierung angesehen werden. Damit kann einerseits der Klang von Musikinstrumenten nachempfunden werden, aber auch nicht-musikalische Soundeffekte wie Regen- und Windgeräusche lassen sich auf diese Weise erzeugen.

3.2.5 Story

Die Story als narrativer Rahmen eines Computerspiels stellt aufgrund der Interaktivität des Mediums ein besonderes Einsatzgebiet für prozedurale Techniken dar. Anders als in nicht-interaktiven Medien wie Büchern und Filmen, kann der Rezipient in einem Computerspiel aktiv ins Geschehen eingreifen. Soll die Narration dann dynamisch auf die Handlungen des Spielers reagieren, ist der Einsatz von prozeduralen Systemen unerlässlich. Dabei herrscht jedoch eine große Diskrepanz zwischen zwei scheinbar gegensätzlichen Zielen: Einerseits sollen die unvorhersehbaren Aktionen des Spielers die Story beeinflussen, andererseits soll das Spiel weiterhin eine interessante und kohärente Geschichte erzählen. Dieser Konflikt wird unter anderem in [14] thematisiert. Generell ist um den Begriff des Interactive Storytellings ein sehr aktives Forschungsgebiet mit unterschiedlichsten Lösungsansätzen entstanden, über die z. B. in [16] ein Überblick gegeben wird.

3.2.6 Rätsel

Das Lösen von Rätseln ist Teil vieler Spiele und stellt in einigen Fällen sogar die Kernmechanik dar. Gerade für logikbasierte Rätsel, die genau definierten Regeln folgen, bieten sich prozedurale Generierungsmethoden an. So existiert beispielsweise eine Vielzahl an Sudokugeneratoren, die Varianten des Rätsels in konfigurierbaren Schwierigkeitsgraden erstellen können. Unterschiedliche Rätsel erfordern naturgemäß unterschiedliche Generierungsalgorithmen – in der Literatur finden sich daher einige Arbeiten, die Ansätze für verschiedene Arten von Rätseln vorstellen [1, 11, 29].

3.3 Methoden

Obwohl Algorithmen zur prozeduralen Generierung von Spielinhalten auf ihren spezifischen Anwendungsfall abgestimmt sind, lässt sich eine Reihe von allgemeinen Methoden und Konzepten identifizieren, die dabei wiederholt zum Einsatz kommen. Hendrikx u. a. [9] haben eine Taxonomie dieser Methoden erstellt und dabei sechs Klassen definiert, die in den folgenden Abschnitten näher beschrieben werden.

3.3.1 Zufallszahlengeneratoren

Der Zufall ist ein wesentliches Element vieler Techniken zur prozeduralen Inhaltserstellung. Oft wird der Begriff prozedurale Generierung auch achtlos mit zufälliger Generierung gleichgesetzt. Diese Assoziation ist jedoch problematisch, da das generierte Ergebnis keineswegs ein reines Zufallsprodukt ist; vielmehr folgt die Generierung einem deterministischen Ablauf, der eine Reihe von Einschränkungen für die generierten Inhalte definiert. Innerhalb dieser Einschränkungen können die Inhalte aber aufgrund eines stochastischen Prozesses zufällig variiert werden. In der Programmierung werden dazu *Pseudozufallszahlengeneratoren* verwendet, die ausgehend von einem Startwert eine zufällig scheinende Zahlenfolge erzeugen, die allerdings von einem deterministischen Algorithmus erzeugt wird und daher nicht wirklich zufällig ist.

3.3.2 Generative Grammatiken

Generative Grammatiken sind Regelsysteme, die ursprünglich aus der Linguistik stammen und beschreiben, wie aus der Kombination von Wörtern eine unbegrenzte Anzahl grammatikalisch korrekter Sätze einer Sprache konstruiert werden kann. Viele Ansätze zur prozeduralen Generierung haben solche Grammatiken für ihre Zwecke adaptiert. Sie beziehen sich dann nicht länger auf Worte, sondern beispielsweise auf Elemente eines Plattformspiels, die basierend auf den Regeln einer eigens entwickelten Grammatik zu einem Level zusammengesetzt werden. Eine häufig verwendete Variante von generativen Grammatiken sind außerdem *Lindenmayer-Systeme*, die unter anderem zur prozeduralen Modellierung von Pflanzen eingesetzt werden [23, 24].

3.3.3 Bildfilter

Filter sind ein wichtiges Instrument in der Bildverarbeitung und spielen im Rahmen der prozeduralen Generierung vor allem bei der Erstellung von Texturen eine Rolle. Auch bei der Generierung von Terrains, wo Höheninformationen oft in Form von 2D Texturen (sogenannten Heightmaps) abgespeichert werden, kommen Filteroperationen zum Einsatz – beispielsweise um eine Glättung durchzuführen.

3.3.4 Räumliche Algorithmen

Räumliche Algorithmen manipulieren Raum, um Spielinhalte zu generieren. Ein simples Beispiel dafür ist die *Kachelung* der Spielwelt, also die Erzeugung größerer Flächen durch die lückenlose Anordnung gleichförmiger Teilflächen (Kacheln). Daneben finden verschiedene Unterteilungsalgorithmen (*Grid Subdivision*, *Voronoi-Diagramme*) Anwendung, um dynamisch Details an ausgewählten Stellen in der Spielwelt hinzuzufügen [2, 22]. So kann etwa die Auflösung des Terrains im Umkreis des Spielers prozedural erhöht werden, während weiter entfernte Abschnitte reduziert dargestellt werden, um Rechenleistung zu sparen. Auch *Fraktale* – rekursive Gebilde, die aus verkleinerten Kopien ihrer selbst bestehen – eignen sich für diesen Anwendungszweck, da sie einen theoretisch unendlichen Detailgrad in einer einfachen rekursiven Funktion kapseln.

3.3.5 Modellierung und Simulation komplexer Systeme

Wenn einzelne mathematische Funktionen nicht mehr ausreichen, um Inhalte prozedural zu generieren, kommen komplexere Modelle und Simulationen zum Einsatz. In diese Kategorie fallen unter anderem *zelluläre Automaten*, die ein räumlich diskretes Modell von Zellen simulieren. Jede Zelle kann dabei unterschiedliche Zustände annehmen, die von den vorangehenden Zellzuständen in einer festgelegten Nachbarschaft abhängen. Ein bekanntes Beispiel für einen solchen zellulären Automaten ist John Conways Spiel des Lebens (Game of Life).

Eine weitere Methode zur Modellierung und Simulation von komplexen Systemen sind *agentenbasierte Modelle*. Dabei werden einzelne Individuen – die Agenten – mit relativ simplen Verhaltensmustern simuliert, aus deren Interaktion sich in der Gesamtheit jedoch komplexe Abläufe ergeben. Dieses Phänomen wird als Emergenz bezeichnet und kann genutzt werden, um basierend auf simplen Regeln komplexere Spielinhalte zu generieren.

3.3.6 Künstliche Intelligenz

Das Gebiet der künstlichen Intelligenz befasst sich damit, menschliche Intelligenz maschinell nachzubilden. Es existieren also teils große Parallelen zur prozeduralen Generierung, die versucht, manuelle Kreation zu automatisieren. Viele Ansätze aus dem Bereich der künstlichen Intelligenz sind folglich auch für die prozedurale Generierung von Spielinhalten relevant. So können beispielsweise *genetische Algorithmen* verwendet werden, die den Prozess der biologischen Evolution nachahmen, um dadurch Lösungen für Optimierungs- und Suchprobleme zu finden. Auch *künstliche neuronale Netze* sind Teil mancher Generierungsmethoden. Außerdem spielen *automatisierte Planung* und Lösungsansätze für *Constraint-Satisfaction-Probleme* häufig eine wichtige Rolle, wenn Spielinhalte algorithmisch erzeugt werden und dabei bestimmte

Bedingungen (Constraints) erfüllt werden müssen, um die Spielbarkeit der Ergebnisse zu garantieren.

3.4 Online- und Offline-Generierung

Eine wichtige Unterscheidung im Rahmen der prozeduralen Generierung von Spielinhalten ist hinsichtlich ihres Einsatzes zur Laufzeit oder zur Entwicklungszeit des Spiels zu treffen:

- Bei der sogenannten **Online-Generierung** werden die Generierungsalgorithmen dynamisch zur Laufzeit ausgeführt – die erzeugten Inhalte können also bei jedem Programmablauf variiert werden. Oft ist dies ein zentrales Argument für den Einsatz prozeduraler Generierung, da die immer neuen Spielinhalte zu einem gesteigerten Wiederspielwert führen sollen.
- Im Gegensatz dazu steht die **Offline-Generierung**, bei der Spielinhalte zwar mit Hilfe prozeduraler Techniken erstellt werden, dann aber wie manuell kreierte Inhalte abgespeichert und unverändert ins Spiel geladen werden. Diese Vorgehensweise wird verwendet, wenn grundsätzlich prozedural generierte Inhalte manuell auf ihre Spielbarkeit überprüft oder anderweitig verbessert werden sollen. Außerdem eignen sich einige der in Abschnitt 3.3 vorgestellten Methoden nur für den Offline-Einsatz, da sie aufgrund ihrer Komplexität nicht in Echtzeit berechnet werden können.

3.5 Einsatz in existierenden Spielen

Obwohl vergleichsweise viele wissenschaftliche Arbeiten existieren, die sich mit prozeduralen Techniken zur Erstellung von Spielinhalten befassen, ist ihr umfassender Einsatz in der Praxis noch eher die Ausnahme als die Regel. Im Folgenden werden einige repräsentative Spiele beschrieben, deren Inhalte zumindest teilweise algorithmisch generiert sind.

3.5.1 Rogue

Rogue ist ein 1980 entwickeltes Rollenspiel, in dem der Spieler auf der Suche nach einem Amulett ein unterirdisches, von Monstern bevölkertes Verlies erkundet. Die besondere Innovation von *Rogue* war, dass das Layout dieses Verlieses sowie die Platzierung der darin befindlichen Objekte bei jedem Spielstart prozedural neu generiert wurden. Dem Spieler stand damit erstmals eine praktisch unendliche Anzahl an unterschiedlichen Levels zur Verfügung, die er erforschen konnte (Abbildung 3.4).

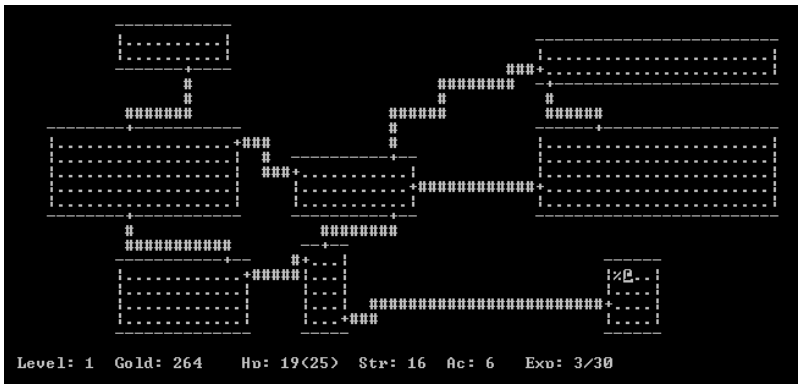


Abbildung 3.4: Prozedural generiertes Layout von Räumen und Verbindungskorridoren in *Rogue* (1980) [46]. Zur grafischen Repräsentation der gesamten Spielwelt kamen ASCII-Zeichen zum Einsatz.

Abbildung 3.5: Spielszene aus *Diablo III* (2012) [47].

3.5.2 Diablo

Bei *Diablo* handelt es sich um eine Action-Rollenspielserie mit mittlerweile drei Einzeltiteln (*Diablo III* erschien im Mai 2012), die ähnlich zu *Rogue* das Erkunden von Dungeons und den Kampf gegen die zahlreichen darin anzutreffenden Monster beinhaltet (Abbildung 3.5). Auch hier findet der Levelaufbau teilweise prozedural statt, indem manuell erstellte Levelteile algorithmisch zu einem kompletten Level zusammengesetzt werden.

3.5.3 Civilization

Civilization ist eine Serie rundenbasierter Strategiespiele mit dem Ziel, eine gesamte Zivilisation von der Steinzeit bis in die Moderne zu führen und dabei zur vorherrschenden Weltmacht aufzusteigen. Neben vorgefertigten Karten

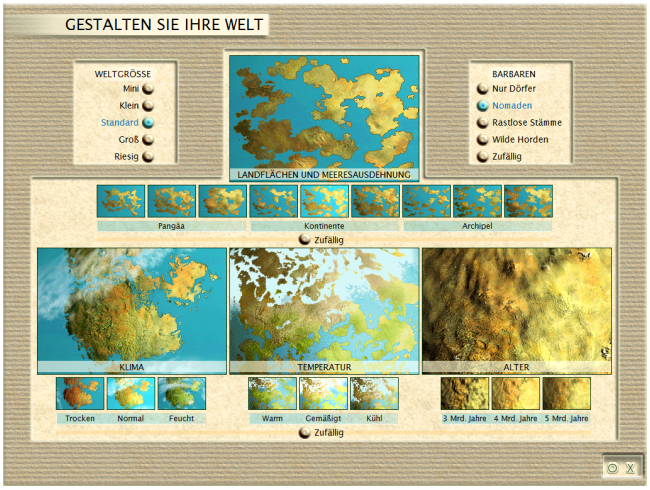


Abbildung 3.6: Der Kartengenerator in *Civilization III* (2001) erstellt unter Vorgabe einiger grundlegender Parameter prozedural neue Welten.

bieten die Spiele auch die Möglichkeit, Landschaften prozedural vom Computer generieren zu lassen, wobei der Spieler bestimmte Grundparameter regulieren kann (z. B. Meeresausdehnung oder Klima, siehe Abbildung 3.6). Die Generierung muss natürlich eine möglichst faire Verteilung von Ressourcen und Startpositionen garantieren können, um das Spielgleichgewicht nicht negativ zu beeinflussen.

3.5.4 Spore

Ziel von *Spore* (2008) ist es, die Evolution einer eigene Spezies von ihren Anfängen als Einzeller über mehrere Entwicklungsstadien bis hin zur weltraumbeherrschenden Zivilisation zu steuern. Das Spiel setzt dabei in vielen Aspekten auf prozedurale Generierung. So werden etwa die vom Spieler in einem umfangreichen Editor zusammengestellten Kreaturen prozedural texturiert und animiert (Abbildung 3.7). Auch die zu besiedelnden Landschaften und Planeten werden aufgrund bestimmter Regeln algorithmisch erzeugt und unterscheiden sich daher in jeder Partie.

3.5.5 Borderlands

Bei *Borderlands* handelt es sich um einen 2009 veröffentlichten Egoshooter mit Rollenspielelementen. Ein zentrales Merkmal des Spiels ist die große Vielfalt an Waffen und anderen Gegenständen, die dem Spieler zur Verfügung stehen. *Borderlands* verwendet ein prozedurales System, um diese Objekte aus einer Kombination von möglichen Elementen und Attributen zusammenzusetzen. Die entstehenden Waffen unterscheiden sich beispielsweise hinsicht-

Abbildung 3.7: Der Kreatureneditor in *Spore* (2008) ermöglicht das Zusammenstellen unterschiedlichster Lebewesen, die entsprechend ihrer Bestandteile prozedural texturiert und animiert werden. Kreaturen mit mehreren Beinen können beispielsweise laufen, während Einbeiner sich hüpfend fortbewegen und schlangenähnliche Wesen ohne Beine sich vorwärts winden.

Abbildung 3.8: Die Attribute einer prozedural generierten Waffe in *Borderlands* (2009) [48].

lich Genauigkeit, Schussfrequenz oder Magazingröße (Abbildung 3.8). Auch die Feinde im Spiel werden aus zufällig ausgewählten Charakteristiken und Angriffsmustern zusammengestellt.

Kapitel 4

Verwandte Ansätze

In der Literatur finden sich einige Arbeiten, die sich mit dem Thema der prozeduralen Levelgenerierung speziell für das Genre der Plattformspiele auseinandersetzen. Eine der ersten Arbeiten auf dem Gebiet stammt von Compton und Mateas aus dem Jahr 2006 [3]. Sie stellen fest, dass die Levelgenerierung für Plattformspiele eine besondere Herausforderung darstellt:

> Platformer level generation is a more difficult problem than level generation in either RPG or strategy games, since very small changes, such as slightly changing the width of a chasm, can change a whole level from challenging to physically impossible.

Es ergeben sich also sehr spezielle Einschränkungen durch die physikalischen Bewegungsmöglichkeiten des Spieleravatars, auf die bei der Generierung Rücksicht genommen werden muss, um die Spielbarkeit der entstehenden Levels sicherstellen zu können. In den folgenden Abschnitten wird ein Überblick über eine Reihe von relevanten Arbeiten gegeben, die sich bereits mit den Herausforderungen der prozeduralen Levelgenerierung für Plattformspiele befasst und unterschiedliche Lösungsansätze dazu entwickelt haben.

4.1 Rhythmusbasierte Ansätze

Der von Compton und Mateas vorgestellte Ansatz [3] ist von einem Modell inspiriert, das rhythmische Muster afrikanischer Musik beschreibt. Basierend auf der Annahme, dass Rhythmus und Timing auch zentrale Aspekte im Leveldesign von Plattformspielen darstellen, wurde dieses Modell adaptiert und zur prozeduralen Platzierung von Plattformen und Hindernissen in einer bestimmten rhythmischen Abfolge verwendet.

Smith u. a. [31, 32] entwickelten einen Levelgenerator, dessen grundlegende Funktionsweise ebenfalls auf Rhythmen basiert. In einem mehrstufigen Verfahren wird zuerst ein Rhythmus aus einzelnen Takten generiert, wobei jeder Takt einer Aktion des Spielers (z. B. einem Sprung) entspricht.

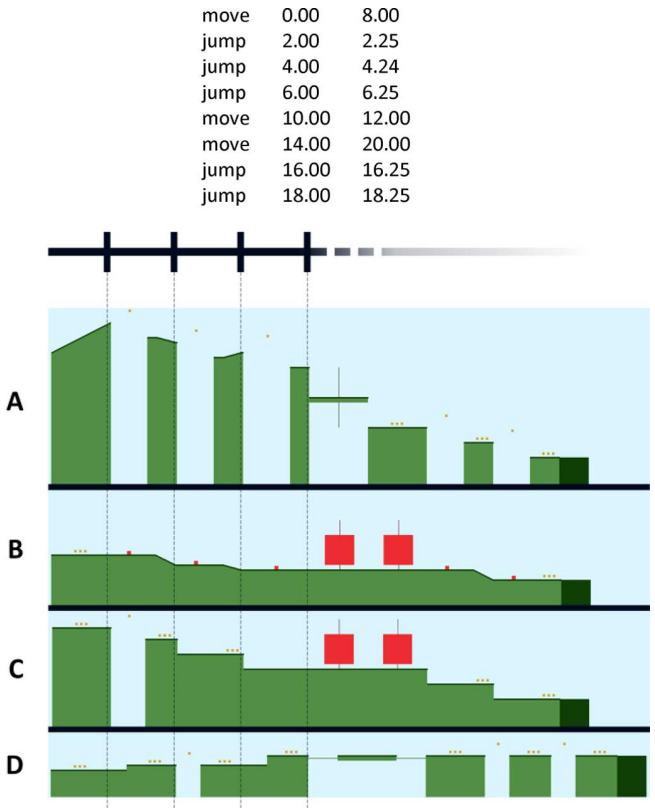


Abbildung 4.1: Ein Rhythmus und vier mögliche geometrische Repräsentationen, die daraus generiert werden können [31].

Im zweiten Schritt wird diese Zusammenstellung aus Aktionen unter Verwendung einer speziellen Grammatik (siehe dazu auch Abschnitt 3.3.2) und unter Berücksichtigung der gegebenen physikalischen Einschränkungen in passende Levelgeometrie umgesetzt (Abbildung 4.1). Aus der Aneinanderreihung unterschiedlicher Rhythmen bzw. deren geometrischer Repräsentation wird dann eine Sammlung potenzieller Levels generiert. Anhand gewisser Bewertungsheuristiken wird aus all diesen Levelkandidaten der beste ausgewählt und in einem abschließenden Schritt finalisiert. Dieser letzte Schritt besteht aus globalen Operationen wie beispielsweise dem Verteilen von einzusammelnden Objekten und dem Anpassen aller Plattformen an eine gemeinsame Grundlinie.

4.2 Ansätze mit manuell erstellten Teillösungen

Ein anderer Ansatz zur Levelgenerierung stammt von Mawhorter und Mateas [15]. Sie stellen ein System vor, das manuell gestaltete Levelfragmente pro-

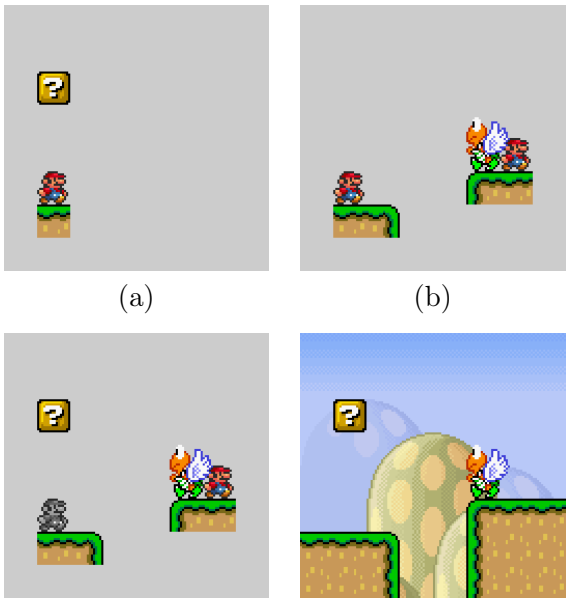


Abbildung 4.2: Ein Schritt der Levelgenerierung in [15]: Bestehende Levelgeometrie mit einem Ankerpunkt (a), Auswahl eines dazu passenden Levelfragments mit zwei Ankerpunkten (b), Kombination der beiden Teile anhand eines Ankerpunktes (c) und resultierender Levelabschnitt im Spiel (d).

zedural zu kompletten Levels zusammensetzt. Das Besondere dabei ist, dass die vorgefertigten Fragmente nicht einfach zufällig aneinandergereiht werden (die wohl simpelste Form prozeduraler Generierung), sondern anhand mehrerer potenzieller Ankerpunkte miteinander kombiniert werden. Diese Ankerpunkte sind innerhalb der einzelnen Fragmente definierte Positionen, die der Spieler beim Durchqueren des Levels besetzen könnte. Ausgehend von einem Startpunkt werden Schritt für Schritt passende Fragmente ausgewählt und an bestehende Ankerpunkte angefügt, wobei Überlappungen zwischen neuer und bestehender Levelgeometrie miteinander verschmelzen (Abbildung 4.2). Durch diese Art der Kombination und Verschmelzung von manuell erstellten Teilen ergeben sich mitunter ganz neue, emergente Levelabschnitte. Auch nichtlineare Levels mit mehreren möglichen Wegen, die in den sehr speziellen Regelsystemen anderer Ansätze oft nicht vorgesehen sind, können auf diese Weise entstehen. Nachteilig sind allerdings die Abhängigkeit von Anzahl und Variantenreichtum der vorgefertigten Levelfragmente sowie die – seltene aber doch vorhandene – Möglichkeit der Generierung unspielbarer Levelkonstellationen.

Auch *Spelunky* – ein von Derek Yu und Andy Hull entwickeltes Plattformspiel – beinhaltet prozedural generierte Levels, die auf manuell entworfenen

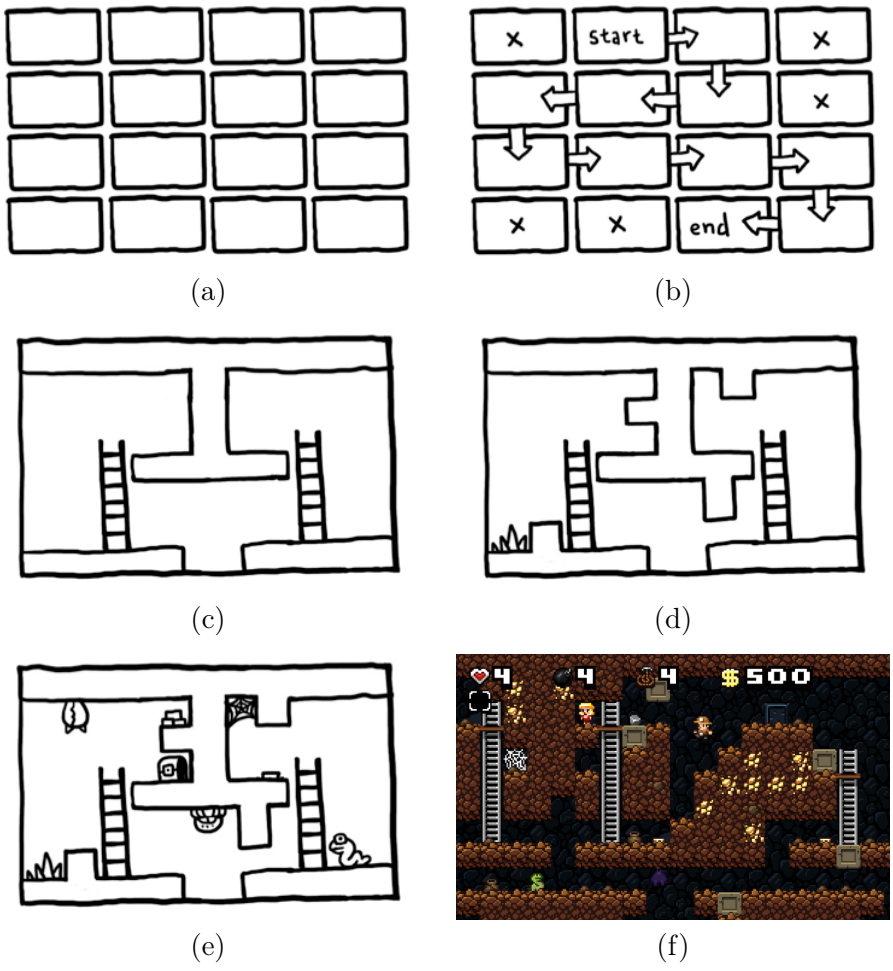


Abbildung 4.3: Levelgenerierung in *Spelunky*: Unterteilung des Levels in einzelne Räume (a), Generierung eines Pfades vom Levelstart zum -ausgang (b), Auswahl vorgefertigter Raumdesigns (c), Variation der Raumgeometrie (d), Hinzufügen von Hindernissen, Schätzen und Dekoration (e) bis zu einem fertigen Levelabschnitt in Spielgrafik (f) [51].

Teilabschnitten aufbauen [51]. Jedes Level wird zuerst in 16 gleich große Räume unterteilt, die in einem 4×4 Raster angeordnet sind. Dann wird ein Pfad vom Levelstart in der obersten Reihe des Rasters zum Levelausgang in der untersten Reihe generiert. Für jeden Raum wird schließlich aus einer Menge vorgefertigter Designs zufällig eines ausgewählt, das zudem prozedural variiert wird, indem aufgrund bestimmter Regeln zusätzliche Geometrie, Hindernisse und Schätze hinzugefügt werden. Abbildung 4.3 veranschaulicht den Prozess.

4.3 Ansätze mit genetischen Algorithmen

Sorenson und Pasquier [33] generieren Plattformspielwelten nach einer Vorgehensweise, die sich von den bisher vorgestellten wesentlich unterscheidet. Sie merken an, dass prozedurale Systeme bestimmte Voraussetzungen benötigen, um die Kreativität manueller Designs entfalten zu können:

> Generative systems exhibit creativity when they are able to construct solutions that are not simply parameterized variations of existing solutions.

Ihr Ansatz gibt daher keine festen, einschränkenden Regeln für die Generierung vor, sondern basiert auf der Verwendung eines genetischen Algorithmus, der inspiriert von der natürlichen Evolution versucht, eine Menge an möglichen Levels durch Rekombination und Mutation iterativ zu verbessern. Die Güte der Levels wird dabei nach jeder Iteration mit Hilfe einer Fitnessfunktion evaluiert, sodass nur die besseren Levels in die nächste Generation übernommen werden. Die Entwicklung einer geeigneten Fitnessfunktion stellt eine zentrale Herausforderung beim Einsatz von genetischen Algorithmen dar, da sie die Evolution steuert und die Qualität der Resultate entscheidend beeinflusst. Sorenson und Pasquier definieren maximale Fitness als maximalen Spielspaß, der sich aus der optimalen Übereinstimmung zwischen Anforderung an den Spieler und dessen Können ergibt. Mit einem stark vereinfachten Modell messen sie dazu die Schwierigkeit von Sprüngen über den Verlauf eines Levels und lassen sie durch genetische Operationen schrittweise optimieren. Am Ende des Prozesses steht ein Level mit entsprechend platzierten Plattformen und Abgründen.

Auch Mourato u. a. [19] setzen genetische Algorithmen ein, um automatisch Levels für ein Plattformspiel zu generieren. Sie unterteilen die Spielwelt in ein gleichmäßiges Raster, in dem jede Zelle einen von drei Zuständen annehmen kann: leer, Wand oder Boden. Bei einer Levelgröße von n Zellen ergeben sich also 3^n mögliche Levelkombinationen, was bei einem kleinen Levelabschnitt bestehend aus 30 Zellen bereits zu über 200 Billionen Kombinationsmöglichkeiten führt. Um in diesem großen Suchraum in vernünftiger Rechenzeit eine Lösung zu finden, können unmöglich alle Kombinationen durchgetestet werden. Daher wird ein genetischer Algorithmus verwendet, der – wie bereits oben erwähnt – die Fitness von ausgewählten Levelkandidaten messen und iterativ erhöhen soll. Die von Mourato u. a. vorgestellte Fitnessfunktion basiert dabei auf einer Reihe von Heuristiken, die unterschiedliche Levelqualitäten bewerten, beispielsweise die Struktur des Pfades vom Levelstart zum -ausgang, die Sinnhaftigkeit einer Zelle in Abhängigkeit von ihren Nachbarzellen und die Ausgewogenheit des visuellen Erscheinungsbildes des Levels.

Der Einsatz von genetischen Algorithmen zur Levelgenerierung zeigt in den vorgestellten Fällen erste positive Resultate (Abbildung 4.4). Vorteilhaft

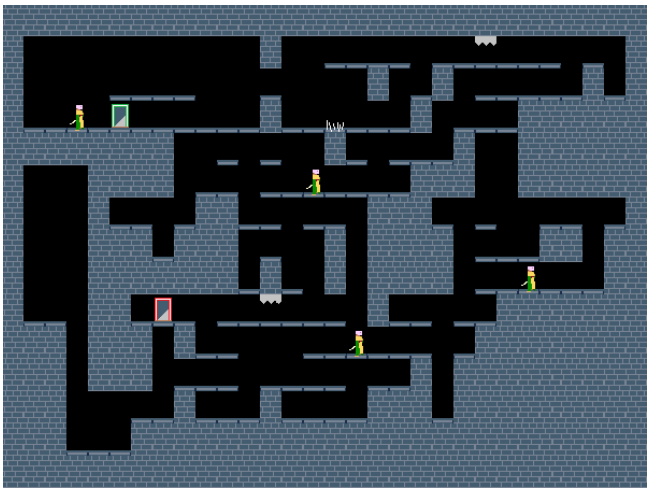


Abbildung 4.4: Ein mit Hilfe genetischer Algorithmen generiertes Level bestehend aus den drei Grundkomponenten leerer Raum, Wand und Boden. Hindernisse wie Gegner und Stachelfallen wurden in einem Nachbearbeitungsschritt hinzugefügt [19].

ist vor allem die Unabhängigkeit von spezifischen Regelsystemen, wodurch auch neuartige, kreative Lösungen außerhalb explizit vorgesehener Variationsmöglichkeiten entstehen können. Ein limitierender Faktor von genetischen Algorithmen ist allerdings ihre oft lange Laufzeit. Während der Ansatz von Sorenson und Pasquier nicht primär auf Effizienz ausgelegt ist und mit Laufzeiten von mehreren Stunden daher nur für den Offline-Einsatz in Frage kommt, kann der Ansatz von Mourato u. a. ein Level von akzeptabler Qualität zumindest in einigen Minuten generieren.

4.4 Erfahrungsgesteuerte Ansätze

Eine weitere Gruppe von Arbeiten [20, 27, 36] befasst sich damit, Levels anhand der Erfahrungen des Spielers zu generieren. Dazu wurden empirisch Daten von Spielern erhoben, die ein *Super Mario Bros.* nachempfundenes Plattformspiel spielten. Aufgezeichnet wurden unter anderem die benötigte Zeit zum Abschluss eines Levels, die Anzahl der Sprünge, die Anzahl und Art der eingesammelten Objekte, die Anzahl der getöteten Gegner sowie die Anzahl und Art der Spielertode. Außerdem bewerteten die Spieler einander gegenübergestellte Levels hinsichtlich ihrer emotionalen Präferenz in den sechs Kategorien Spaß, Herausforderung, Langeweile, Frustration, Vorhersehbarkeit und Anspannung. Die gesammelten Daten wurden verwendet, um neuronale Netze zu trainieren und in weiterer Folge Modelle aufzustellen, die den Zusammenhang zwischen den Eigenschaften eines Levels, der beob-

achteten Spielweise eines Spielers und den hervorgerufenen Emotionen beschreiben. Mit Hilfe dieser Modelle ist es dann möglich Levels zu generieren, die eine bestimmte Spielerfahrung bieten bzw. einem bestimmten Spielstil entgegen kommen.

Kapitel 5

Eigener Ansatz

5.1 Anforderungen

Die Entwicklung des vorgestellten Ansatzes zur prozeduralen Generierung von 2D Plattformerlevels wurde von drei zentralen Anforderungen geleitet, die unbedingt zu erfüllen waren:

- **Garantie der Spielbarkeit.** Die generierten Levels müssen ausnahmslos alle spielbar sein, d. h. es muss immer ein möglicher Lösungsweg vom Start- zum Zielpunkt existieren. Dazu müssen vor allem die physikalischen Möglichkeiten des Spieleravatars berücksichtigt werden, sodass beispielsweise keine Abgründe entstehen, die aufgrund einer zu geringen Sprungweite unmöglich überwunden werden können.
- **Möglichkeit der Online-Generierung.** Die Generierung muss ausreichend schnell erfolgen, um sie online – also zur Laufzeit des Spiels – ausführen zu können. Erst durch den Online-Einsatz kann die prozedurale Generierung als dynamisches Element in die Spielmechanik eingebunden werden, anstatt als bloßer Ersatz für die manuelle Erstellung von Spielinhalten zur Entwicklungszeit zu dienen.
- **Minimierung der Notwendigkeit manueller Eingriffe.** Die Generierung soll möglichst wenig auf manuell erstellte Teillösungen und die manuelle Einstellung bestimmter Parameter angewiesen sein. Zwar sollen die entstehenden Levels durchaus über sinnvolle Parameter variiert werden können, aber die grundlegende Funktion der Generierung soll nicht von der Korrektheit manueller Anweisungen abhängen.

5.2 Grundannahmen

Vor der Konzeption einer geeigneten Methode zur prozeduralen Levelgenerierung wurden einige Grundannahmen getroffen, die den Kontext und den Funktionsumfang der Generierung konkretisieren:

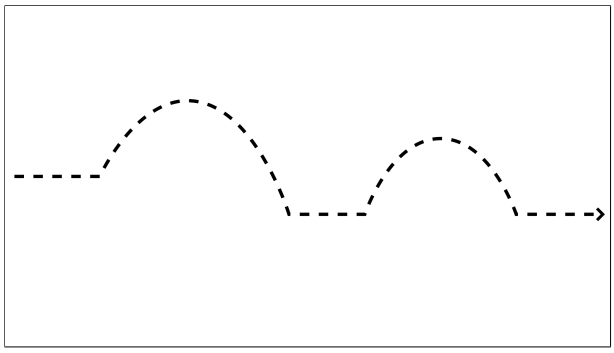
- Die Generierung ist speziell auf die Erstellung von Levels für Plattformspiele mit zweidimensionaler Spielmechanik ausgelegt. Dies geht schon aus dem Titel der Arbeit hervor. Auf die grundlegenden Unterschiede zwischen 2D und 3D Plattformspielen wurde außerdem bereits in Abschnitt 2.3 hingewiesen.
- Die möglichen Bewegungsaktionen des Spielers sind auf Laufen und Springen beschränkt. Diese beiden Aktionen definieren im Prinzip das Genre der Plattformspiele und reichen aus, um darauf basierend herausfordernde Spielumgebungen zu kreieren. Zusätzliche Aktionsmöglichkeiten sind natürlich denkbar, erhöhen aber auch die Komplexität der Levelgenerierung und werden daher vorerst nicht berücksichtigt.
- Das primäre Spielziel ist das Erreichen des Levelausgangs durch geschickt ausgeführte Bewegungsmanöver. Der Fokus der Generierung liegt daher vorrangig auf dem Erzeugen von eher geradlinigen Hindernisparcours und weniger auf dem Erstellen von weitläufigen Spielwelten, die explorativ entdeckt werden wollen.
- Der Verlauf eines Levels gestaltet sich primär horizontal von links nach rechts. Das Spielgeschehen wird von der Seite betrachtet und nur horizontal gescrollt. Die vertikale Ausdehnung eines Levels wird auf den verfügbaren Sichtbereich beschränkt, es gibt kein vertikales Scrolling. Daraus ergibt sich implizit eine bessere Übersicht über den Levelverlauf, sodass die Generierung darauf nicht mit expliziten Maßnahmen Rücksicht nehmen muss.

5.3 Grundkonzept

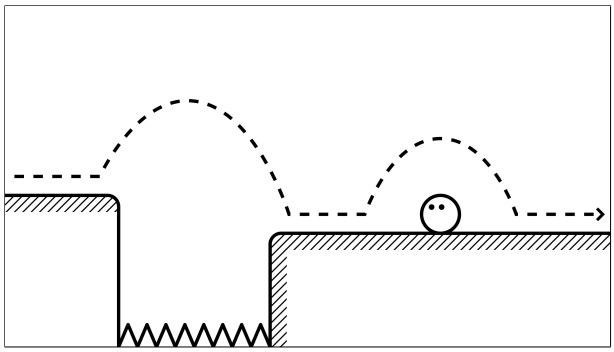
Um den definierten Anforderungen unter Berücksichtigung der erläuterten Grundannahmen nachzukommen, wurde eine darauf abgestimmte Vorgehensweise zur Levelgenerierung konzipiert, die aus den folgenden zwei grundlegenden Schritten besteht:

1. Generierung eines möglichen Lösungsweges,
2. Generierung der an den Lösungsweg angepassten Levelgeometrie.

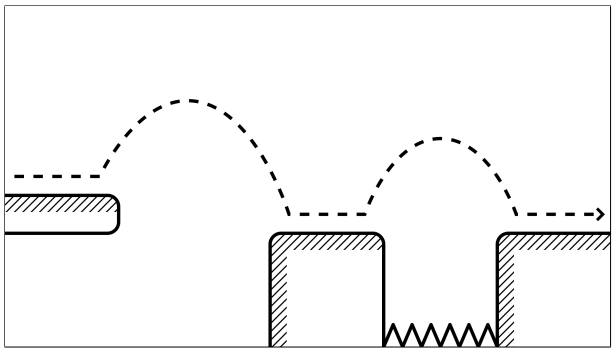
Der eigentliche Prozess der Levelgenerierung basiert also auf einem zuvor generierten Lösungsweg. Dadurch wird sichergestellt, dass in jedem Fall zumindest ein gültiger, vom Spieler begehbarer Weg durch das Level existiert. Die Generierung kann gewissermaßen als Invertierung des Spielprozesses verstanden werden: In einem herkömmlichen Plattformspiel versucht der Spieler, einen Lösungsweg durch ein vorgegebenes Leveldesign zu finden; das vorgestellte System zur prozeduralen Levelgenerierung geht nun den umgekehrten Weg und versucht, ein Leveldesign zu finden, das einen vorgegebenen Lösungsweg impliziert. Abbildung 5.1 illustriert die Vorgehensweise.



(a)



(b)



(c)

Abbildung 5.1: Die prozedurale Levelgenerierung erfolgt in zwei Phasen: Zuerst wird ein möglicher Lösungsweg generiert, der dann in entsprechende Levelgeometrie umgesetzt wird. Der in (a) gezeigte Lösungsweg könnte beispielsweise zu den geometrischen Interpretationen (b) oder (c) führen.

5.4 Generierung des Lösungsweges

In ihrer simpelsten Form besteht die Lösung zum erfolgreichen Durchqueren eines Plattformerlevels aus einer Serie entsprechend getimter Lauf- und

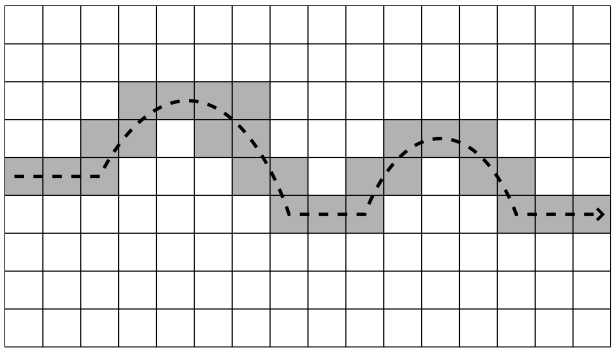


Abbildung 5.2: Collision Map zur näherungsweisen Speicherung des generierten Lösungsweges.

Sprungaktionen. In komplexeren Plattformspielen können noch zusätzliche Aktionen wie rechtzeitiges Ducken oder Schießen notwendig sein, um ein Level erfolgreich zu absolvieren – diese werden in den folgenden Betrachtungen jedoch nicht berücksichtigt.

Um nun einen neuen Lösungsweg zu generieren, werden die beiden Grundaktionen Laufen und Springen abwechselnd aneinandergereiht und innerhalb bestimmter Grenzen zufällig variiert, bis der resultierende Weg eine vorgegebene Länge erreicht hat. Die wesentlichen Größen, die den Verlauf der Bewegung beeinflussen, sind dabei Zeit t, Geschwindigkeit $\vec{v}$ und Gravitation $\vec{g}$. Die Bewegungsfunktion eines Laufabschnittes ergibt sich aus

$$\vec{r}(t) = \vec{r}_0 + \vec{v} \cdot t, \tag{5.1}$$

während der parabelförmige Kurvenverlauf eines Sprunges definiert ist durch

$$\vec{r}(t) = \vec{r}_0 + \vec{v} \cdot t - \frac{1}{2} \cdot \vec{g} \cdot t^2. \tag{5.2}$$

Mit Hilfe dieser Funktionen ist es also möglich, die geplante Position $\vec{r}$ des Spielers über den zeitlichen Verlauf eines Levels zu bestimmen. Die Gesamtheit all dieser Positionen ergibt dann den generierten Lösungsweg. Der Weg verläuft immer strikt von links nach rechts. Dadurch kommt es zu keinen Überschneidungen, die zu Konflikten bei der nachfolgenden Generierung der Levelgeometrie führen könnten.

Um spätere Kollisionsabfragen zu vereinfachen und zu beschleunigen, wird der Lösungsweg in Form einer zweidimensionalen Collision Map gespeichert (Abbildung 5.2). Die Auflösung dieser Map entspricht dem Raster, das auch für die gleichförmige visuelle Anordnung der Levelgeometrie verwendet wird (siehe Abschnitt 5.6).

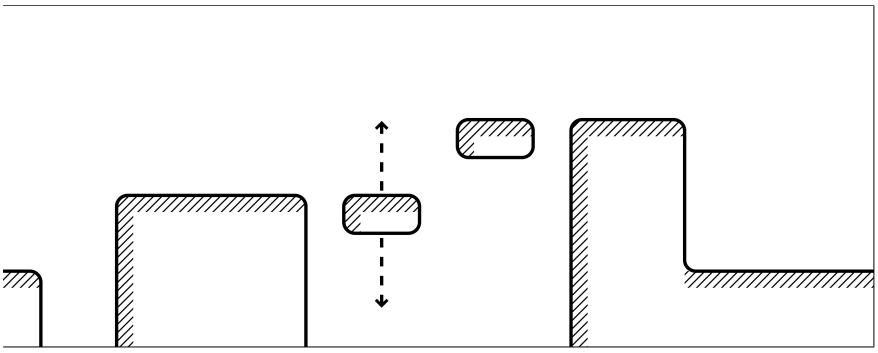


Abbildung 5.3: Unterschiedliche Plattformtypen, die bei der Realisierung von Laufpassagen im Lösungsweg entstehen können.

5.5 Generierung der Levelgeometrie

Nachdem ein zufälliger Lösungsweg erzeugt wurde, kann die Generierung der eigentlichen Levelgeometrie darauf aufbauend erfolgen. Dazu wird für jeden Abschnitt des Weges aus einer Menge an vordefinierten Grundelementen zufällig ein passendes ausgewählt und an den geplanten Bewegungsverlauf angepasst. Ein Laufabschnitt kann beispielsweise durch eine Plattform entsprechender Länge realisiert werden, während ein Sprung möglicherweise als Bewegung über einen dementsprechend dimensionierten Abgrund interpretiert wird. Die in der vorliegenden Implementierung vorhandenen Grundelemente können grob in vier Kategorien eingeteilt werden, die im Folgenden näher beschrieben werden.

5.5.1 Plattformen

Plattformen können überall dort entstehen, wo der Lösungsweg eine Laufpassage vorsieht (Abbildung 5.3).

Statische Plattformen

Eine statische Plattform definiert sich vorrangig durch ihre Länge, die entsprechend der vorgegebenen Dauer und Geschwindigkeit der Laufaktion gesetzt wird. Zusätzlich wird auf Basis konfigurierbarer Wahrscheinlichkeiten zwischen am Boden verankerten und schwebenden Plattformen unterschieden. Diese Unterscheidung ist jedoch nur visueller Natur und hat keine Auswirkung auf die Spielbarkeit.

Bewegte Plattformen

Bewegte Plattformen pendeln auf einem vorgegebenen Pfad zwischen einer Start- und Endposition hin und her. Zusätzlich zur Länge der Plattform

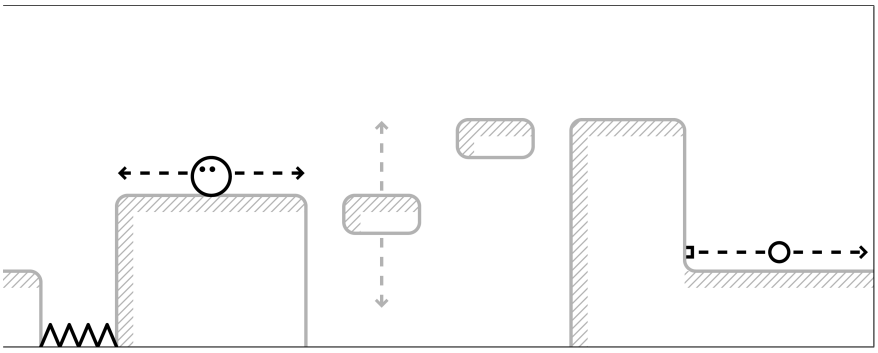


Abbildung 5.4: Verschiedene Arten von Hindernissen, die im Rahmen der Levelgenerierung erzeugt werden können.

müssen in diesem Fall Länge sowie Geschwindigkeit der Bewegung angepasst werden. Der Startzeitpunkt der Bewegung muss außerdem so gewählt werden, dass die Position der Plattform mit der im Lösungsweg festgelegten Position des Spielers zu diesem Zeitpunkt übereinstimmt.

5.5.2 Hindernisse

Hindernisse stellen unterschiedliche Gefahrenquellen für den Spieler dar und müssen mit entsprechend koordinierten Bewegungsaktionen umgangen werden. Sie entstehen vor allem bei der Umsetzung von Sprüngen in Levelgeometrie (Abbildung 5.4).

Statische Hindernisse

Statische Hindernisse verfügen über keine zeitliche Komponente, es handelt sich dabei also um unbewegliche Elemente, die den Lösungsweg allein durch ihre räumliche Anordnung und Ausdehnung erschweren. In der konkreten Implementierung sind es statische Abgründe und Stachelfallen, die in diese Kategorie fallen. Beide dieser Hindernistypen können in ihrer Länge verändert und dadurch an im Lösungsweg vorgesehene Sprungaktionen angepasst werden.

Bewegte Hindernisse

Im Gegensatz zu statischen Hindernissen spielt bei bewegten Hindernissen auch der Faktor Zeit eine Rolle. Diese Art von Hindernis stellt nicht nur eine räumliche Gefahr dar, der Spieler muss zusätzlich die Bewegungsbahn des Hindernisses in sein Vorgehen miteinplanen und seine Aktionen zeitlich dementsprechend abstimmen. Als Beispiele für bewegte Hindernisse dienen in der vorliegenden Implementierung einerseits Gegner, die zwischen festgelegten Start- und Endpunkten patrouillieren, und andererseits kugelförmige

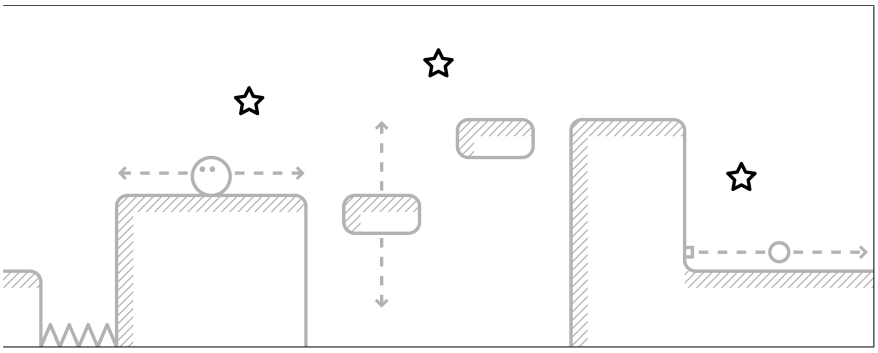


Abbildung 5.5: Einzusammelnde Objekte werden zufällig an erreichbaren Positionen im Level platziert.

Geschosse, die in regelmäßigen Intervallen von Geschützen abgefeuert werden und ihrer vorgegebenen Flugbahn so lange folgen, bis sie auf ein Hindernis treffen. In beiden Fällen sind Sprünge notwendig, um den Bedrohungen auszuweichen. Sie entstehen daher an jenen Stellen im Level, an denen gemäß dem vorgenerierten Lösungsweg ein Sprung vorgesehen ist. Dabei müssen die bewegten Hindernisse wie bereits die bewegten Plattformen nicht nur räumlich, sondern auch zeitlich an die Lösung angepasst werden, sodass es zu keiner Kollision zwischen Spieler und Hindernis kommt und der Lösungsweg nicht beeinträchtigt wird.

5.5.3 Einzusammelnde Objekte

Die Platzierung der einzusammelnden Objekte muss so erfolgen, dass sie vom geplanten Lösungsweg aus erreichbar sind. Gleichzeitig soll eine möglichst ausgewogene Verteilung der Objekte über das gesamte Level gegeben sein (Abbildung 5.5). Dazu werden alle zuvor generierten Plattformen und Hindernisse mit einem Fisher-Yates Shuffle-Algorithmus [12, S. 145–146] durchgemischt. Über die nun zufällig gereihten Elemente wird iteriert und jeweils ein einzusammelndes Objekt pro Element platziert, bis die spezifizierte Anzahl an einzusammelnden Objekten im Level erreicht wurde oder keine weiteren Elemente mehr zur Verfügung stehen. Die Platzierung erfolgt dabei zufällig innerhalb bestimmter Grenzen, die vom jeweiligen Levelelement abhängen. Die maximale Entfernung eines einzusammelnden Objektes von einer Plattform ist beispielsweise durch die maximale Sprunghöhe des Spieleravatars beschränkt. Bei Hindernissen, die mittels Sprung zu überwinden sind, ist hingegen darauf zu achten, dass die Objekte innerhalb der geplanten Sprungkurve platziert werden, damit sie auch garantiert erreicht werden können.

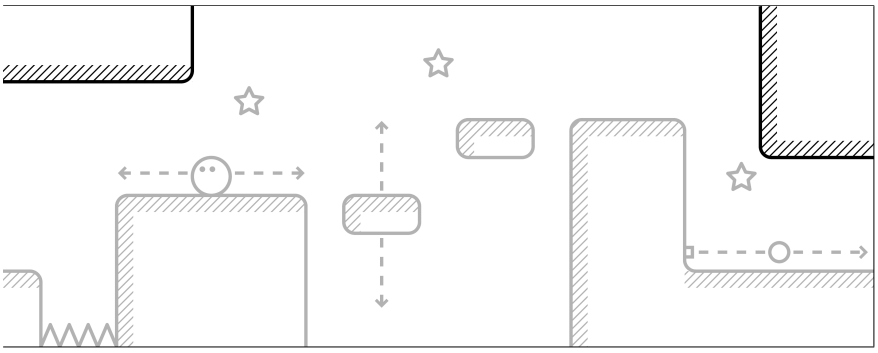


Abbildung 5.6: Die Generierung der Decke bildet den abschließenden Schritt bei der Levelerstellung.

5.5.4 Decke

Nachdem der Untergrund eines Levels in Form von Plattformen und Hindernissen erzeugt und mit einzusammelnden Objekten bestückt wurde, erfolgt im letzten Schritt die Generierung der Decke (Abbildung 5.6). Die verfügbaren Deckenelemente beschränken sich in der vorliegenden Implementierung auf statische Blöcke und freien Himmel, die mit einer einstellbaren Wahrscheinlichkeit ausgewählt und nacheinander in das Level eingepasst werden. Länge und Höhe der einzelnen Abschnitte werden dabei zufällig innerhalb bestimmter Grenzen gesetzt, wobei natürlich darauf geachtet werden muss, dass die Deckenelemente den vorgesehenen Lösungsweg nicht blockieren.

5.6 Gestaltungsraster

Wie bereits mehrfach erwähnt, kommt dem Zufall eine wesentliche Bedeutung bei der prozeduralen Generierung von Spielinhalten zu. Auch das hier vorgestellte System zur Generierung von Leveldesigns setzt in vielen Fällen auf zufällige Entscheidungen bezüglich Anordnung und Ausdehnung einzelner Levelelemente. Während diese Zufälligkeit einerseits für Abwechslung sorgt, kann sie andererseits schnell dazu führen, dass die entstehenden Inhalte visuell ungeordnet und unruhig wirken. Aus diesem Grund kommt in der vorliegenden Implementierung ein simples Gestaltungsraster als Ordnungssystem zum Einsatz, das in ähnlicher Form auch zur visuellen Organisation von Schriftstücken oder Webseiten verwendet wird. Das Raster unterteilt jedes Level in kleine Teilflächen gleicher Größe, an denen sich die Levelgeometrie orientiert. Die Größe jedes Levelelementes entspricht jeweils einem Rasterfeld oder einem Vielfachen davon (Abbildung 5.7). Die Generierung arbeitet daher größtenteils mit ganzzahligen Werten, die sich auf die Anzahl der beanspruchten Rasterfelder beziehen. Erst später erfolgt die Umsetzung in die tatsächlichen Größenangaben in Metern, die reellwertig sind.

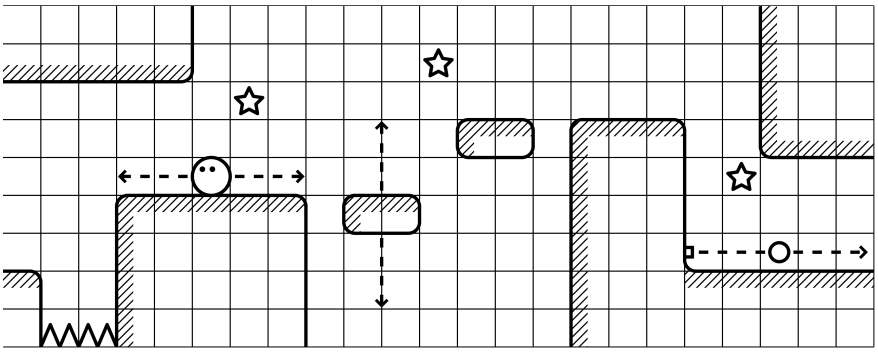


Abbildung 5.7: Alle Levelelemente werden an einem einfachen Gestaltungsraster ausgerichtet.

5.7 Multiple Lösungswege

In den bisherigen Betrachtungen wurde stets von einem einzigen Lösungsweg ausgegangen, der sich vom Anfang bis zum Ende eines Levels durchzieht und die Grundlage für die prozedurale Generierung entsprechender Levelgeometrie bildet. Während dieser Ansatz durchaus funktioniert und spielbare Levels hervorbringt, kristallisiert sich relativ schnell die strenge Linearität der resultierenden Leveldesigns heraus. Durch die Beschränkung auf einen einzigen Lösungsweg, der aus aneinandergereihten Lauf- und Sprungaktionen gebildet wird, besteht auch das darauf basierende Leveldesign aus einer simplen Aneinanderreihung unterschiedlich parametrisierter Levelelemente. Die Wechselwirkungen zwischen den einzelnen Elementen fallen dadurch eher gering aus und es ergeben sich kaum neuartige Konstellationen, die über die manuell konzipierten Grundbausteine hinausgehen. Außerdem verfügt der Spieler über praktisch keine interessanten Wahlmöglichkeiten bei der Suche nach einem Lösungsweg durch ein Level.

Um diesen Problemen entgegenzuwirken, wurde die Generierung des Lösungswegs um die Möglichkeit mehrerer parallel verlaufender Stränge erweitert. Ein zentraler Punkt dabei ist die Entscheidung über Aufspaltung und Zusammenführung der einzelnen Stränge. Das vorliegende System unterteilt das zu generierende Level zunächst in mehrere Teilabschnitte, deren Länge innerhalb eines bestimmten Minimal- und Maximalwertes zufällig gewählt wird. Für jeden Teilabschnitt wird dann aufgrund spezifizierter Wahrscheinlichkeiten entschieden, ob ein einzelner oder zwei parallel verlaufende Lösungsstränge generiert werden. Die Aufspaltung und Zusammenführung erfolgt jeweils an den Grenzen der einzelnen Abschnitte. Die Beschränkung auf maximal zwei Parallelstränge erfolgt aus Platzgründen, da – wie in Abschnitt 5.2 beschrieben – von einer fixen vertikalen Kameraposition ausgegangen wird, sodass das gesamte Level in den vorgegebenen vertikalen Sichtbereich passen muss. Innerhalb der zur Verfügung stehenden Höhe wird

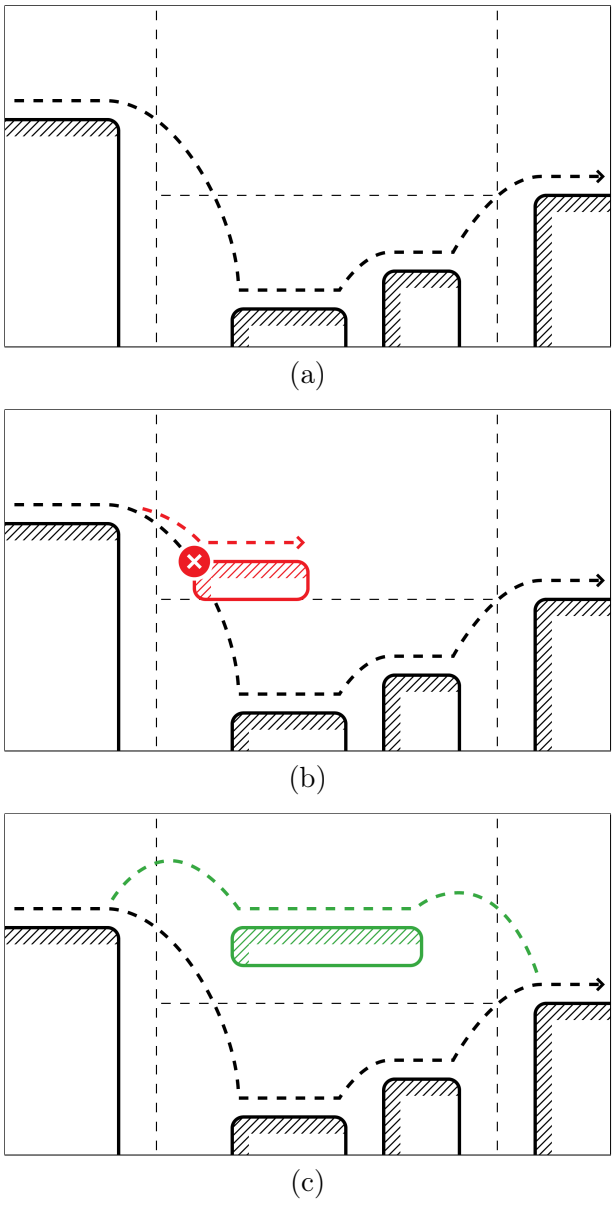


Abbildung 5.8: Vorgehensweise zur Realisierung multipler Lösungswege: Unterteilung des Levels in Abschnitte und Generierung des primären Lösungsweges samt Levelgeometrie (a), Generierung der sekundären Abschnitte inklusive zusätzlicher Kollisionsüberprüfungen, mit deren Hilfe so lange zufällige Lösungen evaluiert werden (b), bis eine gültige gefunden wurde (c).

jedem Lösungsstrang ein eigener Abschnitt zugeordnet, dessen Ausmaß wiederum zufällig bestimmt wird. Diese zusätzliche vertikale Unterteilung stellt sicher, dass sich die Wege nicht schneiden oder überlappen, was zu Konflik-

ten bei der nachfolgenden Generierung der Levelgeometrie führen könnte. Nachdem die horizontalen und vertikalen Grenzen der einzelnen Abschnitte definiert wurden, erfolgt die Generierung des Lösungsweges und der entsprechenden geometrischen Repräsentation individuell pro Abschnitt. Dabei wird zwischen primären und sekundären Wegabschnitten differenziert, für die unterschiedliche Vorgehensweisen notwendig sind.

5.7.1 Generierung des primären Lösungsweges

Sollen an einer Stelle im Level mehrere Lösungswege parallel verlaufen, muss einer davon als primärer Weg definiert werden. Der primäre Lösungsweg wird stets als Erstes generiert, indem wie bereits in Abschnitt 5.4 erläutert zufällig parametrisierte Bewegungsaktionen aneinandergereiht werden. Direkt darauf folgt die Generierung der Levelgeometrie für diesen Weg, die gemäß der in Abschnitt 5.5 beschriebenen Vorgehensweise geeignete Levelelemente auswählt und in das Level einpasst.

5.7.2 Generierung sekundärer Lösungswege

Nachdem der primäre Lösungsweg samt entsprechender Levelgeometrie erzeugt wurde, entstehen die sekundären Lösungsstränge. Hier müssen zusätzliche Einschränkungen berücksichtigt werden, die sich aus den bereits bestehenden Levelelementen des primären Lösungsweges ergeben. Vor allem bei der Abspaltung von und der Zusammenführung mit dem primären Weg kann es zu Konflikten kommen. Daher werden bei der Generierung sekundärer Abschnitte zusätzliche Überprüfungen durchgeführt, um ihre Lösbarkeit sicherzustellen: Bei der Generierung des Lösungsweges wird überprüft, ob eine Kollision mit bestehender Levelgeometrie vorliegt, während bei der Generierung der neuen Levelgeometrie überprüft wird, ob es zu einer Überschneidung mit bestehenden Lösungswegen kommt. Sobald in einem der beiden Fälle eine Kollision erkannt wird, stoppt die Generierung und beginnt mit anderen Zufallsparametern von Neuem. Dieser Prozess wird so lange wiederholt, bis entweder eine gültige Lösung gefunden oder eine maximale Anzahl an Iterationen durchlaufen wurde. Liegt dann noch immer keine gültige Lösung vor, wird der gesamte sekundäre Abschnitt verworfen. Die globale Lösbarkeit des Levels wird dadurch nicht beeinträchtigt, schließlich existiert mit dem primären Lösungsweg immer zumindest ein gültiger Weg durch das gesamte Level. Abbildung 5.8 skizziert die Vorgehensweise anhand eines Beispiels.

5.8 Zeitliche Abhängigkeiten

Der Faktor Zeit spielt eine zentrale Rolle im Leveldesign von Plattformspielen. Bereits das grundlegende Spielprinzip des Springens von einer Plattform zur nächsten verlangt vom Spieler das richtige Timing seiner Aktionen. Für

zusätzliche Komplexität sorgt die Tatsache, dass sich oft nicht nur der Spieler, sondern auch Plattformen und Hindernisse in der Spielwelt bewegen, d. h. neben einer räumlichen auch über eine zeitliche Komponente verfügen. In diesem Fall kommt es leicht zu zeitlichen Abhängigkeiten zwischen den einzelnen Elementen, beispielsweise beim Sprung von einer bewegten Plattform auf eine weitere, ebenfalls bewegte Plattform. Ein solcher Sprung ist nur dann zu schaffen, wenn es mindestens einen Absprungzeitpunkt gibt, zu dem sich die Landeplattform in einer erreichbaren Position befindet, wobei zusätzlich die während dem Sprung vergehende Zeit zu berücksichtigen ist. Abhängigkeiten wie diese müssen natürlich in die prozedurale Levelgenerierung miteinbezogen werden, sodass die Spielbarkeit der entstehenden Levelformationen weiterhin sichergestellt werden kann.

Ein wichtiges Konzept, um zeitliche Abhängigkeiten zwischen Levelelementen aufzulösen, ist die Periodizität der einzelnen Elemente. Darunter ist zu verstehen, dass sich die Position zeitbasierter Elemente periodisch wiederholt: Bewegte Plattformen und Gegner pendeln zwischen festgelegten Wegpunkten hin und her, Kanonen feuern ihre Geschosse in konstanten Intervallen ab. Diese Periodizität sorgt dafür, dass die durch den vorgegebenen Lösungsweg implizierten Plattform- und Hinderniskonstellationen nicht nur zu einem einzigen Zeitpunkt gegeben sind, sondern sich in regelmäßigen Zeitabständen wiederholen. Ohne diese Wiederholung könnten sich für den Spieler Sackgassen ergeben, die ein Weiterkommen unmöglich machen, wenn der vorgesehene Zeitpunkt zur Überwindung einer Levelpassage versäumt wurde.

Allein durch die Tatsache, dass sich bestimmte Konstellationen wiederholen, ist allerdings noch nicht bestimmt, in welchen zeitlichen Abständen die Wiederholung erfolgt. Wenn der Spieler länger als einige Sekunden warten muss, bis sich eine neue Gelegenheit bietet, einen zeitkritischen Levelabschnitt erfolgreich zu absolvieren, wird das Level bereits als unlösbar wahrgenommen. Daher verfügt das vorgestellte Generierungssystem über einen Mechanismus zur Beschränkung der Periodendauer auf einen konfigurierbaren Maximalwert T, der global für das gesamte Level gilt. Die Periodendauer T' jedes zeitbasierten Elementes wird bei der Generierung daran angepasst, indem ein Bruchteil des Maximalwertes angenommen wird, d. h.

$$T' = \frac{T}{n}, \tag{5.3}$$

wobei $n \in \mathbb{N}$. Dadurch wird garantiert, dass alle bewegten Elemente eines Levels einem gemeinsamen Zeittakt folgen und ihre Bewegung innerhalb der maximalen Periodendauer synchron bleibt (Abbildung 5.9).

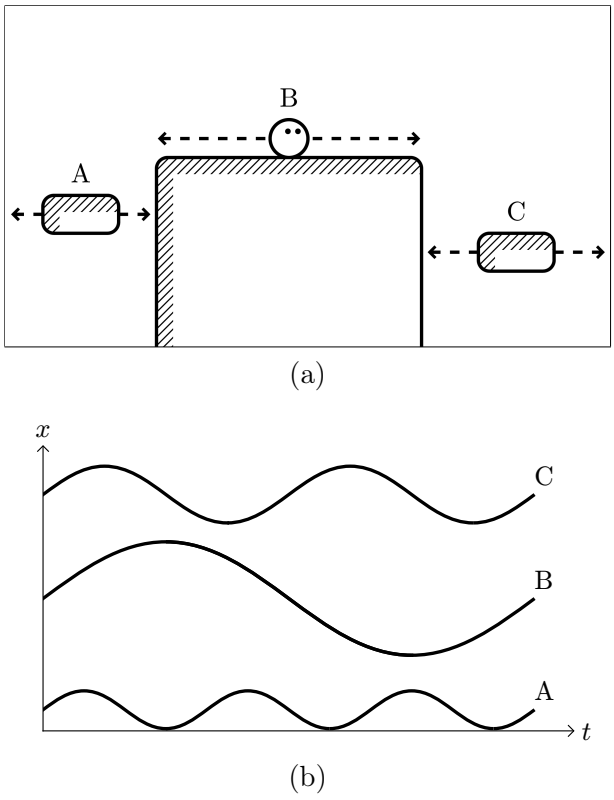


Abbildung 5.9: Drei bewegte Levelelemente (a) und deren Bewegungsfunktionen (b) mit den aufeinander abgestimmten Perioden $3 \cdot T_A = T_B = 2 \cdot T_C$. Die in (a) dargestellte Konstellation wiederholt sich also zu jedem Zeitpunkt $t = n \cdot T_B$ $(n \in \mathbb{N})$.

5.9 Parametrisierung des Schwierigkeitsgrades

Der Schwierigkeitsgrad eines Spiels ist von zentraler Bedeutung für die Spielermotivation. Im Idealfall decken sich das Können des Spielers und die Herausforderung durch das Spiel, sodass sich der Spieler weder unter- noch überfordert fühlt, sondern in den sogenannten Zustand des Flows [5] versetzt wird (Abbildung 5.10). Der Schwierigkeitsgrad eines Plattformspiels leitet sich hauptsächlich aus dem Leveldesign ab, da das primäre Spielziel darin besteht, die Spielwelt in Form unterschiedlich gestalteter Levels möglichst unbeschadet zu durchqueren. Dementsprechend essentiell ist es, bei der Entwicklung eines Systems zur prozeduralen Levelgenerierung für Plattformspiele eine Möglichkeit zur Kontrolle des resultierenden Schwierigkeitsgrades zu schaffen.

In einem ersten Ansatz wurde dazu auf die in Abschnitt 4.3 bereits kurz beschriebene Arbeit von Sorenson und Pasquier [33] zurückgegriffen. Darin

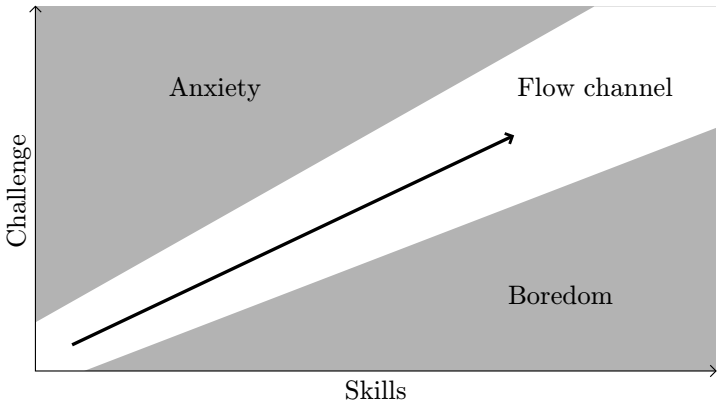


Abbildung 5.10: Die Übereinstimmung von Fähigkeiten und Anforderungen führt zum Flow, einem Zustand optimaler Aufmerksamkeit und Motivation [26, S. 121].

wird ein einfaches mathematisches Modell zur Messung der Schwierigkeit eines Sprunges definiert, das die maximale Sprungweite des Spielers sowie den Abstand und die Ausdehnung von Absprung- und Landeplattform für die Berechnung heranzieht. Je geringer der Abstand zwischen den Plattformen ausfällt und je mehr Raum für Absprung und Landung zur Verfügung stehen, desto niedriger wird die Schwierigkeit des Sprunges bewertet. Während dieses Modell gut geeignet ist, um die Schwierigkeit einzelner Sprünge lokal zu beurteilen, stellte sich bald heraus, dass es nicht flexibel genug ist, um die Schwierigkeit eines gesamten Levels bestehend aus unterschiedlichsten Levelelementen global zu erfassen. Sorenson und Pasquier beschränken sich in ihren Ausführungen außerdem auf statische Elemente, weshalb ihr Schwierigkeitsmodell nur räumliche, nicht aber zeitliche Faktoren berücksichtigt. Gerade diese können den Schwierigkeitsgrad einer Levelpassage aber wesentlich beeinflussen.

Aufgrund der festgestellten Unzulänglichkeiten wurde ein alternativer Ansatz zur Messung und in weiterer Folge Parametrisierung der Levelschwierigkeit entwickelt. Die grundlegende Inspiration dafür stammt aus einem Artikel von Edmund McMillen [50], in dem der Schwierigkeitsgrad von Plattformspielen im Allgemeinen und von *Super Meat Boy* im Speziellen analysiert wird. Der von McMillen mitentwickelte Plattformtitel *Super Meat Boy* ist bekannt für seinen vergleichsweise hohen Schwierigkeitsgrad, der die Spielermotivation trotzdem aufrechterhält und unter Umständen sogar fördern kann. McMillen identifiziert die grundlegende Formel für die Bestimmung der Schwierigkeit als Produkt zweier Faktoren: Einerseits der Wahrscheinlichkeit, im Spiel zu sterben, und andererseits der dafür auferlegten Strafe. In frühen Plattformspielen war der Straffaktor noch sehr hoch: Nach dem

Verlust einer bestimmten Anzahl an Leben verlor der Spieler meist seinen gesamten Fortschritt und wurde an den Beginn des Spiels zurückgesetzt. In *Super Meat Boy* hingegen fällt die Bestrafung nach einer missglückten Aktion wesentlich milder aus: Der Spieler wird nur an den Beginn des aktuellen Levels zurückgesetzt und kann sofort einen neuen Versuch starten. Da die Levels allesamt sehr kurz sind, fällt der verlorene Spielfortschritt nicht merklich ins Gewicht. Im Gegenzug dafür ist allerdings die Wahrscheinlichkeit, im Spiel zu sterben, bedeutend höher. Zahlreiche Hindernisse, Fallen und Gegner erschweren das Erreichen des Levelzieles und begründen den Ruf von *Super Meat Boy* als sehr schwieriges Plattformspiel. Tatsächlich hat sich vor allem der Fokus des Schwierigkeitsgrades weg von der Bestrafung, hin zu einer größeren Herausforderung durch das Leveldesign verschoben.

Genau diese Herausforderung soll nun möglichst präzise gesteuert werden können, wenn Levels prozedural generiert werden. Das vorliegende System bietet daher die Möglichkeit, die gewünschte Schwierigkeit eines Levels in Form eines reellwertigen Parameters von 0 (geringste Schwierigkeit) bis 1 (höchste Schwierigkeit) anzugeben. Als grundlegender Maßstab dient dabei wie in der von McMillen präsentierten Formel die Wahrscheinlichkeit, im Level zu sterben. Bei der Generierung der Levelgeometrie wird jeder Stelle im vorher generierten Lösungsweg entweder ein garantiert sicheres Levelelement (z. B. eine statische Plattform) oder ein potenziell gefährliches Levelelement (z. B. ein Abgrund) zugeordnet. Ein Schwierigkeitsgrad von 0 bedeutet dann, dass für keine Stelle im Lösungsweg ein gefährliches Element erzeugt wird. Wenn der Schwierigkeitsgrad hingegen mit 0,5 spezifiziert wird, führt die Hälfte des Lösungswegs durch potenziell gefährliche Levelabschnitte, während die restliche Hälfte vollkommen sicher ist. Ein Level mit dem Schwierigkeitsgrad 1 beinhaltet in der logischen Konsequenz ausschließlich Elemente, die eine mögliche Gefahr für den Spieler darstellen. Abbildung 5.11 veranschaulicht diese Zusammenhänge.

Die beschriebene Vorgehensweise bietet eine akzeptable Näherung zur Messung und Parametrisierung des Schwierigkeitsgrades eines prozedural generierten Levels. Vorteilhaft ist vor allem, dass die Methode den gesamten Levelverlauf berücksichtigt und nicht auf die lokale Bewertung bestimmter Plattform- und Hinderniskonstellationen beschränkt ist. Außerdem erfolgt die Schwierigkeitsbestimmung unabhängig von den spezifischen Merkmalen einzelner Levelelemente, wichtig ist nur die Unterscheidung zwischen garantiert sicheren und potenziell gefährlichen Stellen. Diese binäre Unterscheidung bietet allerdings noch Raum für Verbesserungen: Allein die Tatsache, *dass* eine Stelle gefährlich ist, sagt nämlich noch nichts darüber aus, *wie* gefährlich sie ist. Hier könnte eine feinere Abstufung vorgenommen werden, um unterschiedliche Gefahren genauer zu differenzieren und in weiterer Folge den Schwierigkeitsgrad exakter bestimmen zu können. Generell hat sich aber gezeigt, dass die Schwierigkeit eines (Plattform-)Spiels ein sehr komplexes Phänomen ist, das von vielen verschiedenen Faktoren abhängt und auch

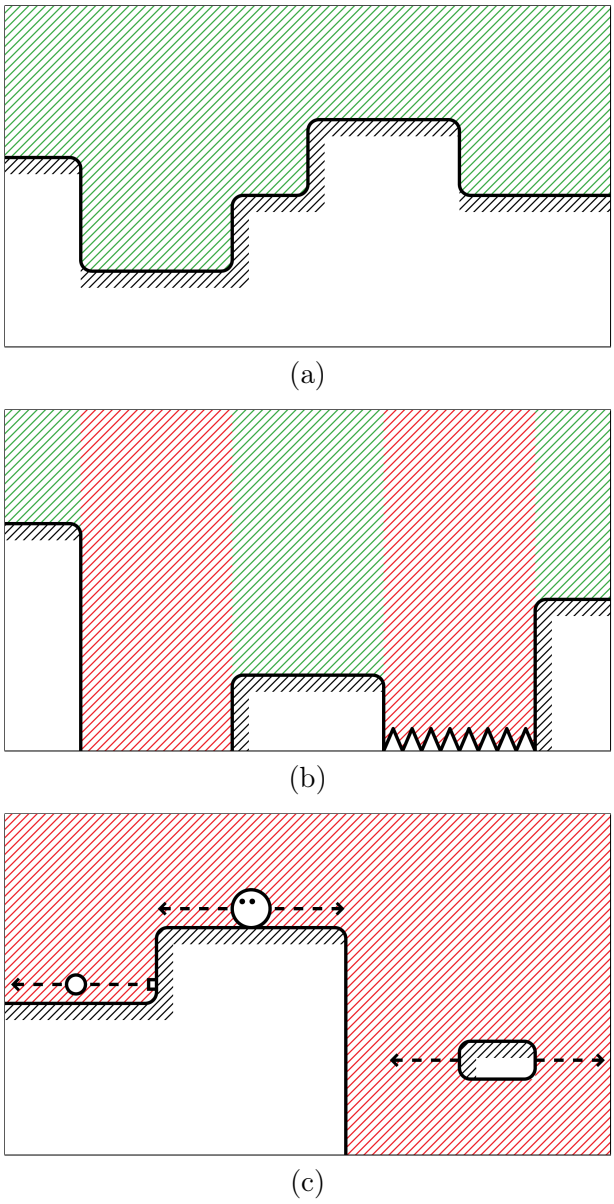


Abbildung 5.11: Unterschiedliche Schwierigkeitsgrade im Überblick: Ein Level mit Schwierigkeit 0 ist vollkommen sicher (a), ein Level mit Schwierigkeit 0,5 beinhaltet zu gleichen Teilen sichere und gefährliche Elemente (b), und in einem Level mit Schwierigkeit 1 stellt schließlich jedes Element eine Gefahrenquelle dar.

von einem Menschen nur ungefähr eingeschätzt werden kann. Deswegen liefert die verwendete Möglichkeit zur näherungsweisen Parametrisierung des Schwierigkeitsgrades bereits zufriedenstellende Ergebnisse.

5.10 Implementierung

Im Folgenden wird ein Überblick über die praktische Implementierung des in den vorhergehenden Abschnitten theoretisch beschriebenen Lösungsansatzes zur Levelgenerierung gegeben.

5.10.1 Verwendete Technologien

Für die technische Umsetzung des entwickelten Ansatzes kam die Spielengine *Unity*[1] zum Einsatz. Der Grund für diese Entscheidung liegt in der einfachen Bedienung, der umfangreichen Funktionalität und dem bestehenden Vorwissen zur Arbeit mit dieser Engine. Obwohl es sich dabei grundsätzlich um eine 3D Engine handelt, können durch die Verwendung orthografischer Kameras auch problemlos 2D Spiele damit entwickelt werden. *Unity* bietet drei unterschiedliche Skriptsprachen zur Implementierung der Programmlogik an, wovon die Wahl auf C# fiel.

5.10.2 Programmarchitektur

Abbildung 5.12 zeigt ein UML-Diagramm der wichtigsten Klassen des Systems, die im Folgenden kurz erläutert werden.

Game

Die Klasse `Game` stellt die zentrale Verwaltungseinheit für das gesamte System dar. Hier erfolgt der Anstoß zur Levelgenerierung, das Zurücksetzen nach einem Tod des Spielers, der Szenenwechsel nach einem erfolgreich absolvierten Level und die Zählung der erreichten Punkte in Abhängigkeit von der Anzahl eingesammelter Objekte.

Level

Die Klasse `Level` beinhaltet die grundlegenden Funktionen für die prozedurale Generierung neuer Levels. Darunter fallen unter anderem die Aufteilung in einzelne Levelabschnitte, das Finden passender Levelelemente für gegebene Lösungswege, die Generierung der Deckengeometrie, die zufällige Platzierung einzusammelnder Objekte sowie die Platzierung von Start- und Zielplattform.

LevelSection

Die kleineren Teilabschnitte, aus denen sich jedes Level zusammensetzt, werden von der Klasse `LevelSection` repräsentiert. Diese ist zuständig für die

[1] http://unity3d.com/

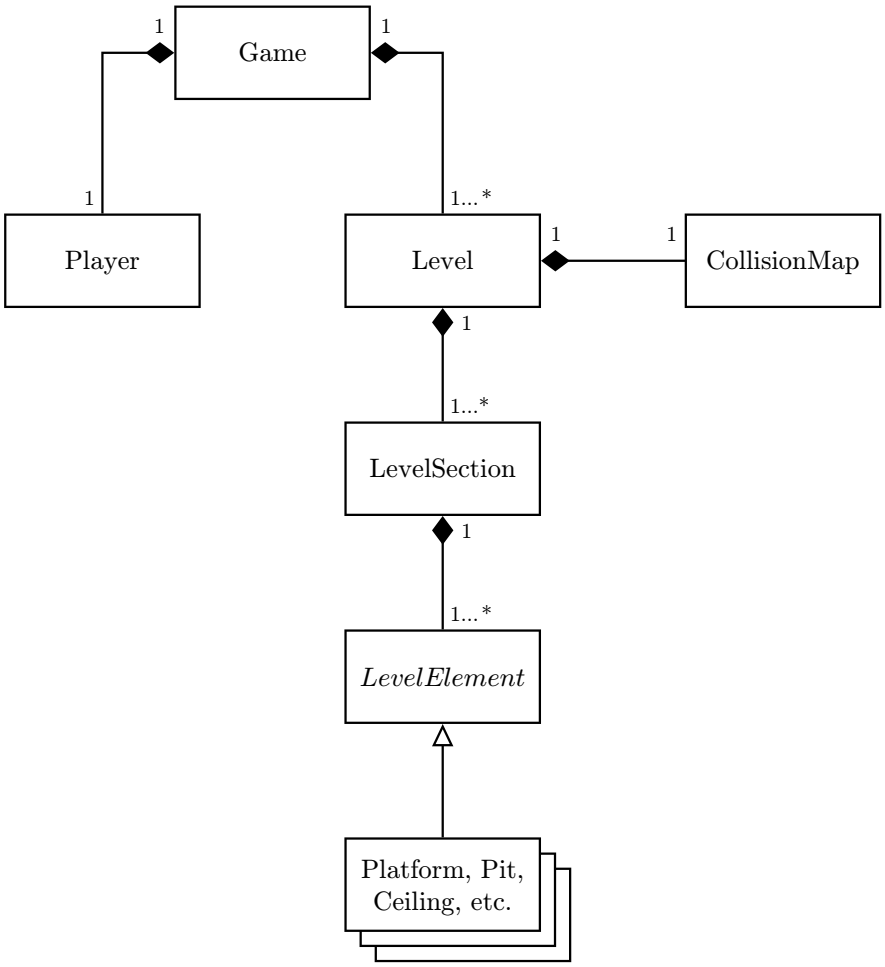


Abbildung 5.12: Die zentralen Klassen des entwickelten Systems und ihre Beziehungen.

abschnittsweise Generierung des Lösungsweges und die Instanziierung geeigneter Levelelemente zur Umsetzung dieses Weges in eigentliche Levelgeometrie. Hier ist auch die spezielle Vorgehensweise zur Generierung mehrerer nebeneinander verlaufender Teilstränge inklusive der Abspaltung von und der Zusammenführung mit dem Hauptstrang implementiert.

LevelElement

`LevelElement` ist eine abstrakte Klasse, die als Basis für die konkrete Implementierung von unterschiedlichen Levelelementen wie Plattformen und Hindernissen dient. Jedem Levelelement wird bei der Erzeugung ein Objekt vom Typ `Constraints` übergeben, das eine Sammlung verschiedener Ein-

schränkungen beinhaltet, an die das Element angepasst werden muss. Diese Einschränkungen definieren beispielsweise Minimal- und Maximalwerte für die horizontale und die vertikale Ausdehnung eines Elementes, die sich aus den Grenzen des übergeordneten Levelabschnittes ableiten. Innerhalb der geltenden Einschränkungen werden die Elemente dann zufällig variiert und in das Level eingefügt.

Player

Die Klasse `Player` definiert das Verhalten und die Steuerung des Spieleravatars. In erster Linie werden dabei die Eingaben des Spielers in entsprechende Bewegungsaktionen umgesetzt und physikalisch simuliert. Außerdem erfolgt hier die Kollisionsüberprüfung mit anderen Objekten der Spielwelt, sodass der Kontakt mit Gefahrenzonen, das Einsammeln von Objekten sowie das Erreichen der Zielposition erkannt und korrekt weiterverarbeitet werden können.

CollisionMap

Die Klasse `CollisionMap` verwaltet ein zweidimensionales Array, das für jede Rasterzelle eines Levels speichert, ob diese leer ist, einen Teil des Lösungsweges darstellt oder von einem Levelelement bzw. einem einzusammelnden Objekt besetzt ist. Diese Informationen werden im Rahmen der Levelgenerierung laufend benötigt, um Überlappungen von Lösungsweg und Levelgeometrie ausschließen und somit die Spielbarkeit des Levels sicherstellen zu können.

5.11 Ergebnisse

Der vorgestellte Lösungsansatz wurde praktisch in einem Plattformspiel mit dem Titel *Night Mary* umgesetzt und erprobt. Das Spiel handelt von einem kleinen Mädchen namens Mary, das einen Albtraum durchlebt, der als Setting für die prozedural generierten Levels dient. Ziel des Spiels ist es, genug Licht in Form von in der Spielwelt verteilten Sternen einzusammeln, um den Albtraum zu beenden. Dafür werden insgesamt hundert Sterne benötigt, wobei in jedem Level maximal zehn zu finden sind. Die Schwierigkeit der Levels wird proportional zur Anzahl der eingesammelten Sterne erhöht, sodass die Herausforderung für den Spieler stetig steigt, je näher er dem Spielziel kommt. Verliert der Spieler in einem Level sein virtuelles Leben, verliert er auch alle Sterne, die in diesem Level eingesammelt wurden, und er wird an den Start eines neu generierten Levels zurückgesetzt. Die Anzahl der Levels ist dank der prozeduralen Generierung unbegrenzt; es werden so lange neue Welten erzeugt, bis das Spielziel von hundert eingesammelten Sternen erreicht wurde. Abbildung 5.13 zeigt verschiedene Szenen aus dem Spiel.

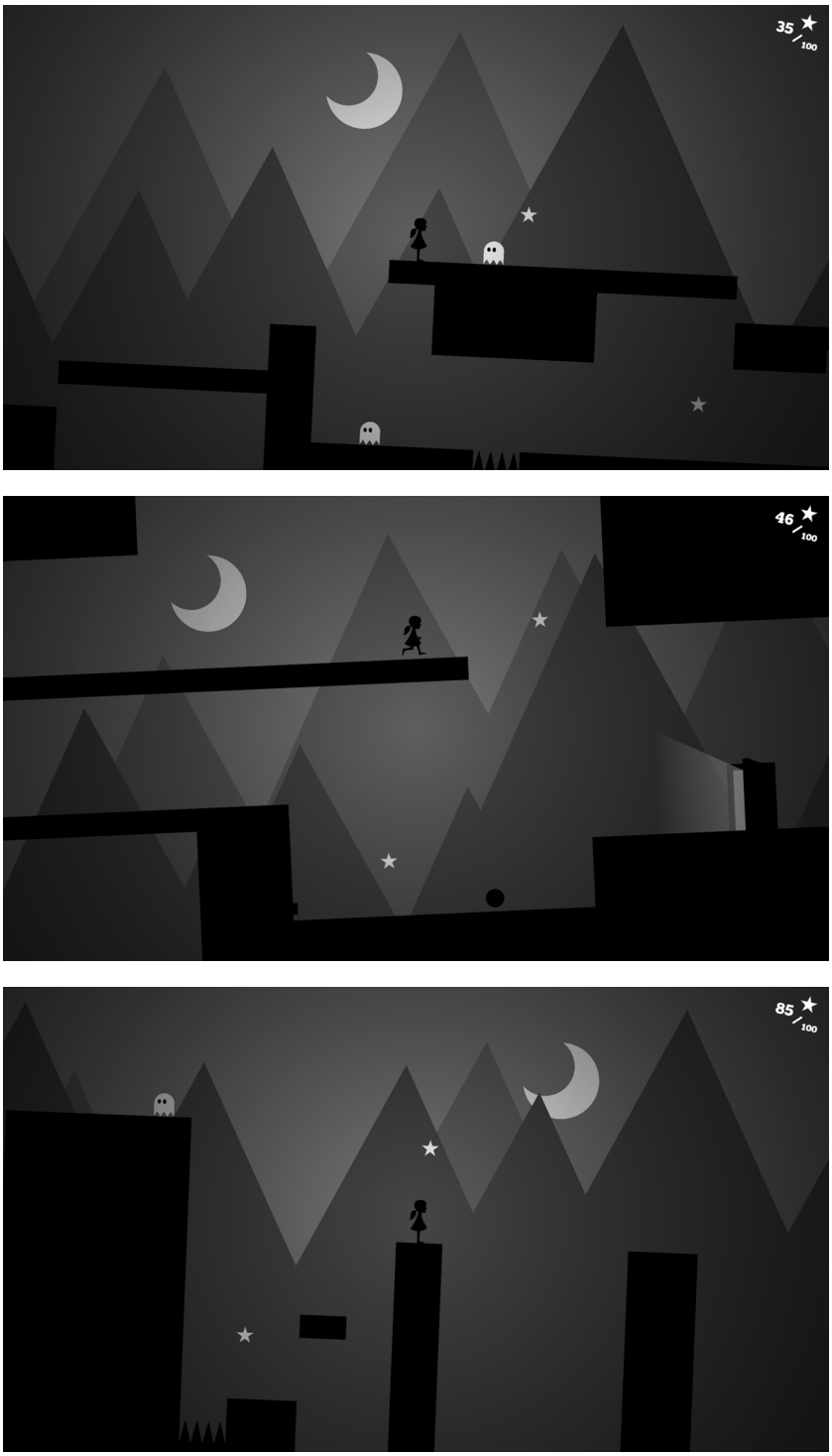


Abbildung 5.13: Spielszenen aus *Night Mary*.

Die Umsetzung hat ergeben, dass mit dem vorgestellten Lösungsansatz alle in Abschnitt 5.1 definierten Anforderungen erfüllt werden konnten. Die Spielbarkeit wird garantiert, indem ein möglicher Lösungsweg als Basis für die Levelgenerierung herangezogen wird. Die notwendige manuelle Arbeit beschränkt sich auf das Erstellen der grundlegenden Levelelemente, ansonsten kann die Generierung sehr einfach durch das Setzen bestimmter Parameter gesteuert werden. Außerdem ist der für die Levelerstellung notwendige Rechenaufwand ausreichend gering, um sie dynamisch zur Laufzeit des Spiels durchführen zu können.

Kapitel 6

Evaluierung

6.1 Methodik

Um die Qualität des entwickelten Lösungsansatzes zur prozeduralen Levelgenerierung für Plattformspiele zu evaluieren, wurde eine Benutzerbefragung mittels Fragebogen durchgeführt. Neben allgemeinen Angaben zu Alter, Geschlecht und Häufigkeit des Computer- bzw. Plattformspielkonsums wurde darin die Meinung der Testteilnehmer zu zentralen Merkmalen der prozedural erstellten Leveldesigns erhoben.

Als Bewertungssystem kam eine Likert-Skala (nach Rensis Likert) zum Einsatz, die den Grad der Zustimmung zu vorgegebenen Aussagen misst. Die abzufragenden Eigenschaften wurden daher als Aussagen formuliert, die von den Testteilnehmern zu beurteilen waren. Bei der Formulierung wurde darauf geachtet, positive und negative Aussagen abzuwechseln, um Zustimmungstendenzen zu vermeiden. Die folgenden acht Aussagen wurden getestet:

- „Die Wartezeit beim Levelwechsel ist gering."
- „Das Leveldesign ist visuell ungeordnet/unruhig."
- „Das Leveldesign ist abwechslungsreich."
- „Das Leveldesign ist zu linear."
- „Das Leveldesign ist übersichtlich/der weitere Weg ist immer gut ersichtlich."
- „Das Leveldesign erzeugt unlösbare Spielsituationen."
- „Das Leveldesign könnte von einem Menschen stammen."
- „Der Schwierigkeitsanstieg im Spielverlauf ist unausgeglichen/weist willkürliche Sprünge auf."

Die Beurteilung dieser Aussagen erfolgte auf einer vierteiligen Skala bestehend aus den Stufen „trifft zu", „trifft eher zu", „trifft eher nicht zu" und „trifft nicht zu". Die Anzahl der Stufen wurde bewusst gerade gewählt, um eine neutrale Mittelposition zu vermeiden und dadurch klarere Tendenzen aus dem Antwortverhalten der Befragten ableiten zu können.

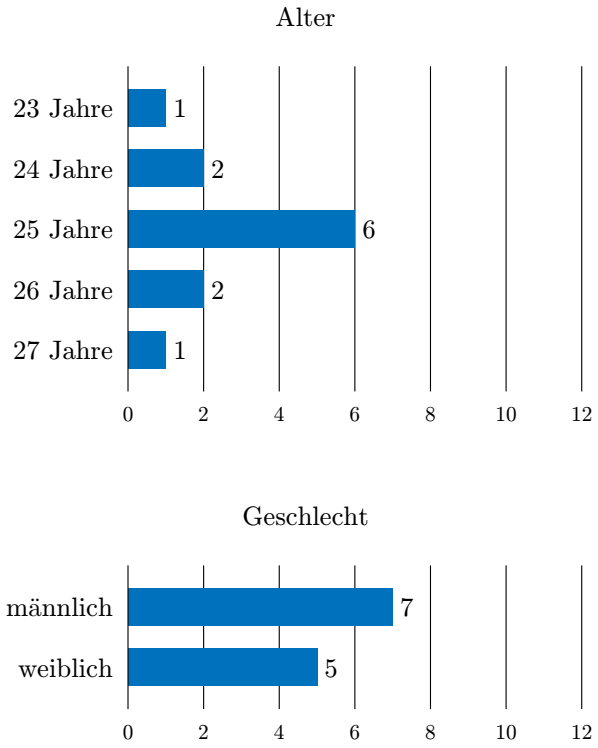


Abbildung 6.1: Alters- und Geschlechtsstruktur der Testteilnehmer.

6.2 Testteilnehmer

Die Gruppe der Testteilnehmer umfasste zwölf Personen, von denen sieben männlich und fünf weiblich waren. Das Alter der Befragten lag zwischen 23 und 27 Jahren, das Durchschnittsalter betrug 25 Jahre (Abbildung 6.1).

Ein Großteil der Befragten spielt regelmäßig Computerspiele: Drei Personen gaben an, mehrmals pro Woche zu spielen, fünf Personen spielen zumindest mehrmals pro Monat. Drei Personen spielen zwar hin und wieder Computerspiele, allerdings seltener als die beiden zuvor genannten Gruppen, und nur eine Person gab an, nie am Computer zu spielen.

Plattformspiele im Speziellen werden von keinem der Befragten mehrmals pro Woche gespielt. Von den elf Personen, die überhaupt Computerspiele nutzen, spielen allerdings sieben mehrmals pro Monat Plattformspiele und vier zumindest gelegentlich, wenn auch seltener (Abbildung 6.2).

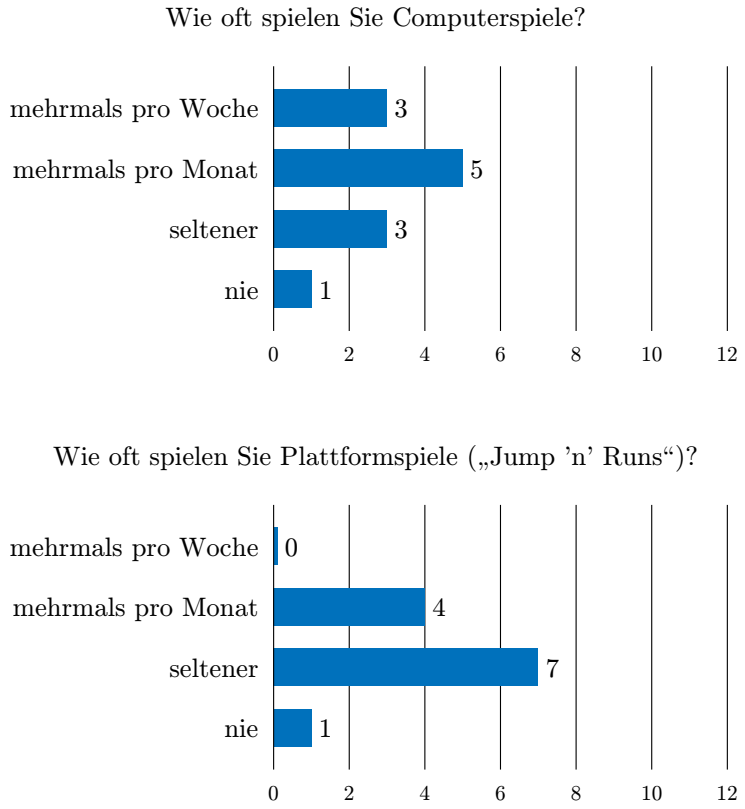
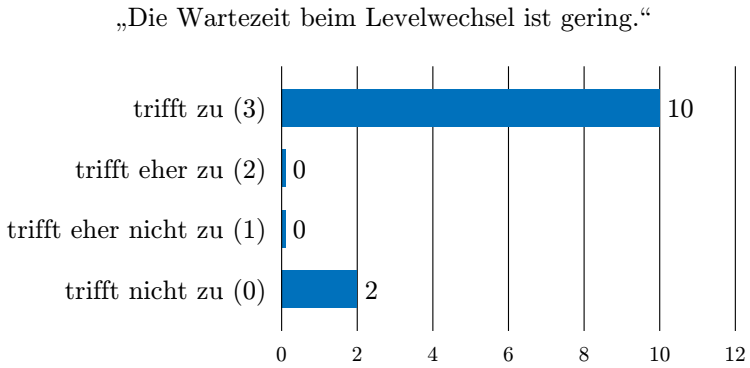


Abbildung 6.2: Häufigkeit des Computer- und Plattformspielkonsums der Testteilnehmer.

6.3 Testablauf

Der Test wurde mit dem im Rahmen dieser Arbeit entwickelten und bereits in Abschnitt 5.11 vorgestellten Plattformspiel *Night Mary* durchgeführt. Das Spiel verwendet den beschriebenen Lösungsansatz zur prozeduralen Levelgenerierung. Die Teilnehmer wurden im Vorfeld über die Bedienung des Spiels und die Tatsache, dass die Levels algorithmisch vom Computer erzeugt werden, informiert. Nach einer kurzen Eingewöhnungsphase zum Kennenlernen der Steuerung bekamen sie dann die Aufgabe, das Spiel zu spielen und durch das Einsammeln der in der Spielwelt verteilten Sterne soweit wie möglich voranzukommen. Die Spielzeit wurde dabei auf fünf Minuten beschränkt, ansonsten folgte das Spiel den Standardregeln. Nach Ablauf der Zeit wurde das Spiel abgebrochen, und den Teilnehmern wurde der vorbereitete Fragebogen zur anonymen Beantwortung vorgelegt.



Durchschnitt: 2,50. Standardabweichung: 1,12.

Abbildung 6.3: Beurteilung der Wartezeit beim Levelwechsel.

6.4 Ergebnisse

Für die Auswertung der Zustimmung zu den untersuchten Aussagen wurde den vier Antwortmöglichkeiten ein Wert von 0 bis 3 zugeordnet:

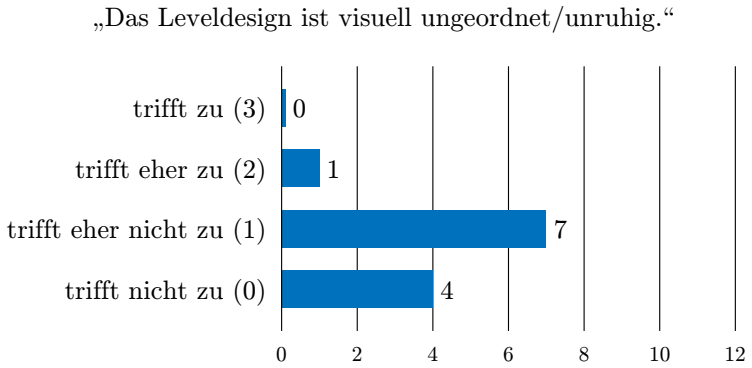
- 0 – „trifft nicht zu“,
- 1 – „trifft eher nicht zu“,
- 2 – „trifft eher zu“,
- 3 – „trifft zu“.

Mit Hilfe dieser Werteverteilung konnte dann die mittlere Zustimmung zu den jeweiligen Aussagen berechnet werden. Im Folgenden werden die Ergebnisse der Auswertung aufgeschlüsselt nach den acht untersuchten Kriterien dargestellt und diskutiert.

6.4.1 Generierungszeit

Die benötigte Rechenzeit ist ein wesentlicher Faktor bei der prozeduralen Generierung von Spielinhalten, ganz besonders wenn die Inhalte dynamisch zur Laufzeit des Spiels erzeugt werden sollen, wie es auch im Rahmen dieser Arbeit der Fall ist. Nicht nur nach dem erfolgreichen Absolvieren eines Levels, sondern auch nach jedem Tod des Spielers wird in *Night Mary* ein neues Level generiert. Die Generierungszeit muss also möglichst gering sein, um den Spieler nicht wiederholt aus dem Spielfluss zu reißen.

Die Testteilnehmer wurden daher befragt, wie sie die Wartezeit beim Levelwechsel empfinden. Das Ergebnis ist relativ eindeutig (Abbildung 6.3): Zehn der zwölf Teilnehmer beurteilen die Aussage „Die Wartezeit beim Levelwechsel ist gering“ als zutreffend. Überraschenderweise sind aber auch zwei



Durchschnitt: 0,75. Standardabweichung: 0,60.

Abbildung 6.4: Beurteilung der visuellen Ordnung der Levels.

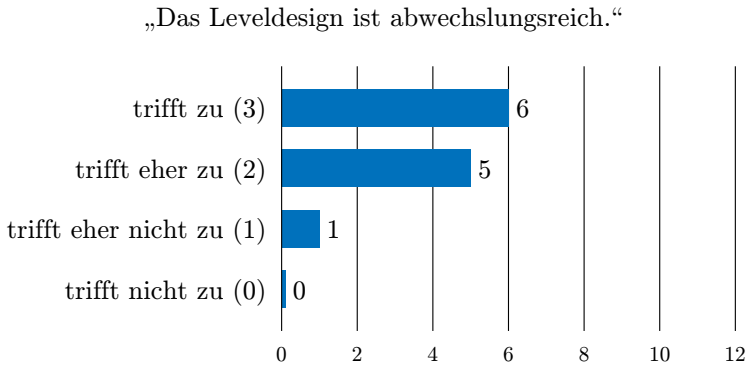
Ausreißer unter den Meinungen, die auf das gegenteilige Extrem der Skala fallen und angeben, die Aussage treffe nicht zu.

Objektiv betrachtet liegt die durchschnittliche Rechenzeit für die Generierung eines neuen Levels im Bereich von unter einer Sekunde, wobei die exakten Werte natürlich von vielen unterschiedlichen Faktoren wie der zur Verfügung stehenden Hardware oder der Länge des zu generierenden Levels abhängen. Dieser Zeitrahmen wird im Allgemeinen als ausreichend gering betrachtet, um die Aufmerksamkeit des Benutzers nicht zu unterbrechen [17].

6.4.2 Visuelle Ordnung

Auf den potenziellen Konflikt zwischen Zufälligkeit und visueller Ordnung eines prozedural generierten Levels wurde bereits in Abschnitt 5.6 hingewiesen. Der Einsatz des dort erläuterten Gestaltungsrasters soll für ein geordnetes Erscheinungsbild der Levels sorgen.

Zur Evaluierung dieser Maßnahme beurteilten die Testteilnehmer die Aussage „Das Leveldesign ist visuell ungeordnet/unruhig“. Die große Mehrheit der Befragten bewertet die Aussage als eher nicht zutreffend (sieben Personen) oder nicht zutreffend (vier Personen). Nur eine Person stimmt der Aussage eher zu (Abbildung 6.4). Die geringe durchschnittliche Zustimmung von 0,75 zeigt also, dass die visuelle Ordnung der generierten Levels grundsätzlich gegeben ist, wenngleich noch etwas Raum für Optimierungen besteht.



Durchschnitt: 2,42. Standardabweichung: 0,64.

Abbildung 6.5: Beurteilung des Abwechslungsreichtums der Levels.

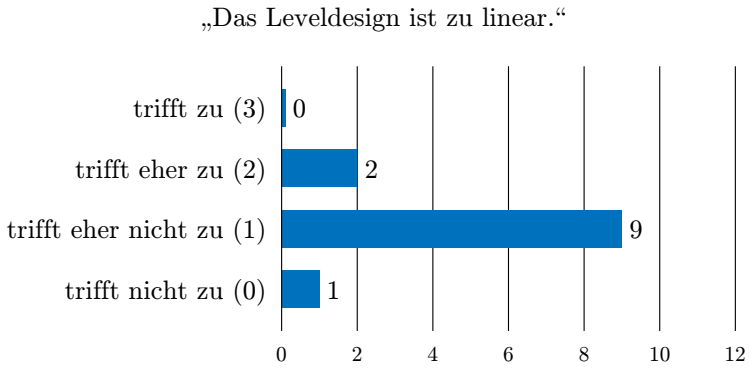
6.4.3 Abwechslungsreichtum

Der Abwechslungsreichtum der erzeugten Levels ist ein weiteres wichtiges Kriterium, um die Qualität eines Ansatzes zur prozeduralen Levelgenerierung beurteilen zu können. Es nützt nichts, wenn das System zwar in der Lage ist, unendlich viele Levels zu generieren, diese aber nicht genügend Variation aufweisen, um das Interesse des Spielers aufrechtzuerhalten.

Aus diesem Grund wurden die Testteilnehmer nach ihrer Zustimmung zu der Aussage „Das Leveldesign ist abwechslungsreich“ befragt. Das Ergebnis fällt verhältnismäßig positiv aus (Abbildung 6.5): Sechs Personen finden die Aussage zutreffend, fünf Personen zumindest eher zutreffend. Nur eine Person ist der Meinung, die Aussage treffe eher nicht zu. Mit einer mittleren Zustimmung von 2,42 stellen die Testteilnehmer also durchaus abwechslungsreiche Levels fest, und dies trotz der Tatsache, dass der Grundstock an verfügbaren Levelelementen sehr klein dimensioniert ist. Das System ist allerdings sehr leicht um neue Elemente erweiterbar, womit der Abwechslungsreichtum sicher noch zusätzlich gesteigert werden könnte.

6.4.4 Linearität

Neben dem Abwechslungsreichtum hat auch die Linearität (bzw. die Nichtlinearität) eines Levels Auswirkungen darauf, ob es vom Spieler als interessant wahrgenommen wird oder nicht. Die Testteilnehmer sollten daher angeben, inwieweit sie der Aussage „Das Leveldesign ist zu linear“ zustimmen. Mit neun Personen beurteilt ein Großteil der Befragten diese Aussage als eher nicht zutreffend, eine Person als nicht zutreffend. Zwei Personen hingegen meinen, die Aussage treffe eher zu (Abbildung 6.6). Insgesamt lässt das Er-



Durchschnitt: 1,08. Standardabweichung: 0,49.

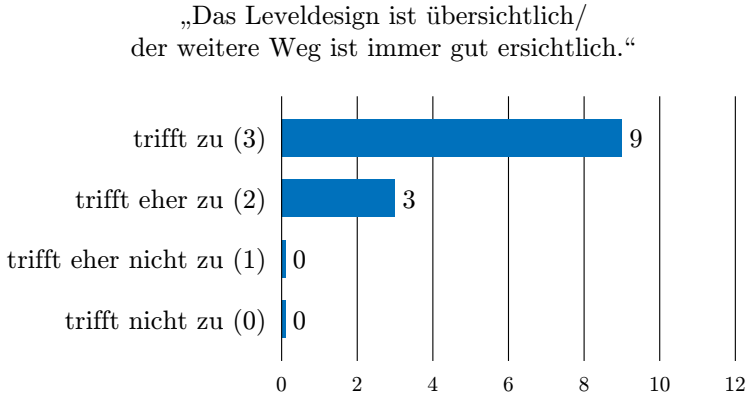
Abbildung 6.6: Beurteilung der Linearität der Levels.

gebnis mit einer Durchschnittszustimmung von 1,08 darauf schließen, dass beim Testen der Levels keine allzu störende Linearität festgestellt werden konnte.

Generell ist anzumerken, dass der Grad der Linearität eines Levels nicht nur eine Frage gewünschter Vielfalt ist, sondern vor allem auch eine Entscheidung, die im Rahmen des grundlegenden Spieldesigns getroffen wird. Viele Plattformspiele legen ihren Fokus auf den Aspekt der Geschicklichkeit, die notwendig ist, um eine Reihe von linear angeordneten Hindernissen zu überwinden, während andere stärker auf das Entdecken einer nichtlinearen Spielwelt oder das Lösen von Rätseln setzen. Der in dieser Arbeit vorgestellte Ansatz zielt vorrangig darauf ab, Levels für die erstgenannte, eher linear ausgerichtete Kategorie von Plattformspielen zu generieren. Mit der in Abschnitt 5.7 erläuterten Vorgehensweise zur Generierung mehrerer parallel verlaufender Levelstränge wurde allerdings versucht, eine – dem Testergebnis zur Folge effektive – Möglichkeit zur Auflockerung der Linearität zu schaffen.

6.4.5 Übersichtlichkeit

Die Übersichtlichkeit eines Levels ist entscheidend für die Orientierung des Spielers und muss – ähnlich wie die visuelle Ordnung eines Levels – trotz der zufälligen Einflüsse auf die prozedurale Generierung sichergestellt werden können. Die Hauptmaßnahme, die dazu getroffen wurde, ist die Beschränkung der Levelhöhe auf den vorgegebenen vertikalen Sichtbereich. Dadurch ist gewährleistet, dass sich der weitere Weg durch ein Level immer im Sichtfeld des Spielers befindet. Natürlich könnte argumentiert werden, dass diese Beschränkung zu Lasten der Levelvielfalt geht. Die Generierung muss also



Durchschnitt: 2,75. Standardabweichung: 0,43.

Abbildung 6.7: Beurteilung der Übersichtlichkeit der Levels.

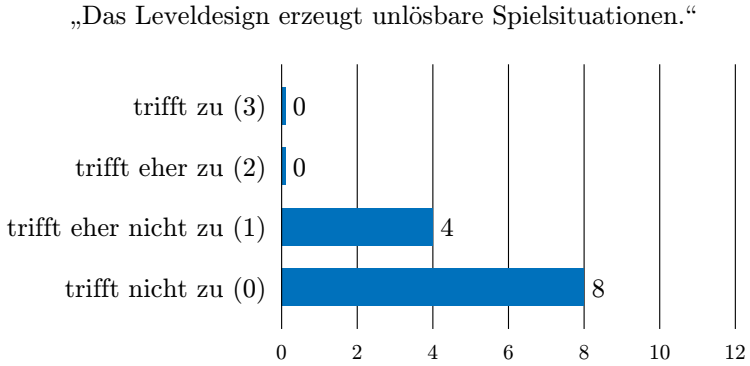
einen Kompromiss zwischen ausreichender Übersicht und genügend Variationsmöglichkeiten finden.

Die Evaluierungsergebnisse bezüglich Abwechslungsreichtum und Linearität belegen, dass ein Großteil der Testteilnehmer mit der Variation der generierten Levels bereits zufrieden ist. Um nun auch die Übersichtlichkeit der Levels zu evaluieren, wurde die Meinung der Testteilnehmer zu der Aussage „Das Leveldesign ist übersichtlich/der weitere Weg ist immer gut ersichtlich“ abgefragt. Die Zustimmung fällt dabei sehr hoch aus (Abbildung 6.7): Neun Personen bewerten die Aussage als zutreffend, die restlichen drei Personen zumindest als eher zutreffend, woraus sich ein Durchschnittswert von 2,75 ergibt. Das Ergebnis zeigt also, dass die Testteilnehmer beim Spielen meist die Übersicht behielten und kaum durch das Leveldesign verursachte Orientierungsprobleme hatten.

6.4.6 Lösbarkeit

Die Garantie der Lösbarkeit jedes prozedural generierten Levels stellte eine zentrale Anforderung bei der Entwicklung des in Kapitel 5 vorgestellten Lösungsansatzes dar. In der Theorie wird diese Anforderung erfüllt, indem die Levelgeometrie wie beschrieben um einen zuvor generierten Lösungsweg herum entsteht. Im praktischen Test wurden die Teilnehmer nun befragt, wie sie die Lösbarkeit der unterschiedlichen Levels tatsächlich wahrgenommen haben.

Dazu wurde ihre Zustimmung zu der Aussage „Das Leveldesign erzeugt unlösbare Spielsituationen“ gemessen. Acht Personen halten diese Aussage für nicht zutreffend, vier Personen für eher nicht zutreffend (Abbildung 6.8).



Durchschnitt: 0,33. Standardabweichung: 0,47.

Abbildung 6.8: Beurteilung der Lösbarkeit der Levels.

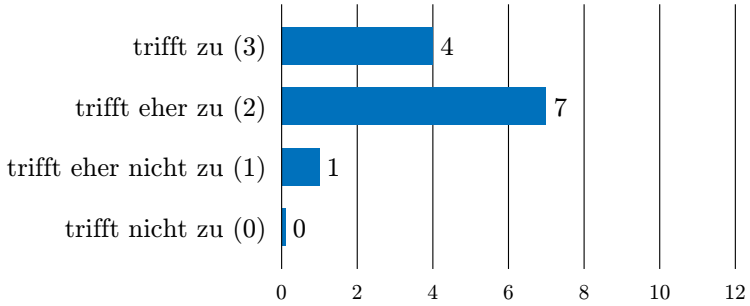
Die durchschnittliche Zustimmung ist mit einem Wert von 0,33 also vergleichsweise gering und zeigt, dass die Testteilnehmer kaum Spielsituationen vorfanden, die sie als unlösbar einschätzten. Dennoch wurde die theoretisch garantierte Lösbarkeit eines Levels in der Praxis nicht hundertprozentig bestätigt, es besteht also noch Potenzial für kleinere Optimierungen zur besseren Anpassung an die Erwartungshaltung der Spieler.

6.4.7 Ähnlichkeit zu manuell erstellten Leveldesigns

Das übergeordnete Ziel eines Ansatzes zur prozeduralen Generierung von Spielinhalten ist in der Regel die bestmögliche Imitation manuell erstellter Inhalte, ohne den damit verbundenen Zeitaufwand tragen zu müssen. Auch in der vorliegenden Arbeit geht es darum, die Qualitäten manuell erstellter Plattformspiellevels möglichst umfassend von einer Computerprozedur nachbilden zu lassen.

Nachdem die Testteilnehmer zu einzelnen dieser Qualitäten bereits befragt wurden, sollten sie nun die allgemeine Aussage „Das Leveldesign könnte von einem Menschen stammen“ bewerten. Vier Personen geben dabei an, die Aussage treffe zu, sieben Personen beurteilen sie als eher zutreffend, und nur eine Person hält sie für eher nicht zutreffend (Abbildung 6.9). Der Durchschnitt der Zustimmung liegt damit bei 2,25. Die Ähnlichkeit zu manuell erstellten Leveldesigns, die als Idealbild und Vorlage für die Generierung dienen, scheint also durchaus gegeben zu sein.

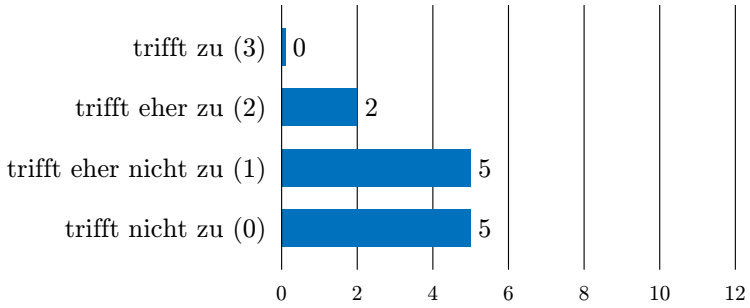
„Das Leveldesign könnte von einem Menschen stammen.“



Durchschnitt: 2,25. Standardabweichung: 0,60.

Abbildung 6.9: Beurteilung der Ähnlichkeit zu manuell erstellten Leveldesigns.

„Der Schwierigkeitsanstieg im Spielverlauf ist unausgeglichen/
weist willkürliche Sprünge auf.“



Durchschnitt: 0,75. Standardabweichung: 0,72.

Abbildung 6.10: Beurteilung des Schwierigkeitsanstieges im Spielverlauf.

6.4.8 Schwierigkeitsanstieg

Die Wichtigkeit eines ausgewogenen Schwierigkeitsgrades für die Spielermotivation wurde bereits in Abschnitt 5.9 erläutert. Die Entwicklung einer möglichst präzisen und gleichzeitig flexiblen Methode zur Parametrisierung des Schwierigkeitsgrades der generierten Levels stellte daher eine der großen Herausforderungen im Rahmen der vorliegenden Arbeit dar.

Um die Eignung der implementierten Schwierigkeitsanpassung zu evaluieren, wurde die Zustimmung der Testteilnehmer zu der Aussage „Der Schwie-

rigkeitsanstieg im Spielverlauf ist unausgeglichen/weist willkürliche Sprünge auf“ erhoben. Die Mehrheit der Befragten beurteilt die Aussage als nicht zutreffend oder eher nicht zutreffend (jeweils fünf Personen). Die restlichen zwei Personen hingegen meinen, die Aussage treffe eher zu (Abbildung 6.10). Insgesamt ergibt sich daraus eine mittlere Zustimmung von 0,75, was darauf schließen lässt, dass die Parametrisierung des Schwierigkeitsgrades grundsätzlich funktioniert, allerdings auch noch etwas Verbesserungspotenzial offen lässt.

Kapitel 7

Schlussbemerkungen

7.1 Zusammenfassung

In Rahmen der vorliegenden Arbeit wurde ein Lösungsansatz zur prozeduralen Generierung von Levels für 2D Plattformspiele entwickelt und vorgestellt. Die grundlegende Vorgehensweise besteht dabei aus zwei aufeinander aufbauenden Schritten: Im ersten Schritt wird ein zufälliger Lösungsweg generiert, der sich aus einer Reihe unterschiedlich parametrisierter Lauf- und Sprungaktionen zusammensetzt. Erst im zweiten Schritt wird die eigentliche Levelgeometrie erzeugt, indem für jeden Abschnitt des zuvor generierten Lösungsweges aus einer Reihe von passenden Levelelementen zufällig eines ausgewählt wird, das den entsprechenden Weg impliziert. Für eine Laufaktion kann beispielsweise eine statische oder eine bewegte Plattform entstehen, während ein Sprung durch einen Abgrund, einen Gegner oder ein anderes Hindernis aufgelöst wird. Unterschiedliche Parameter ermöglichen die Kontrolle des Systems. So können beispielsweise Wahrscheinlichkeiten für die Häufigkeit des Auftretens bestimmter Levelelemente oder der gewünschte Schwierigkeitsgrad eines Levels spezifiziert werden.

Der erarbeitete Lösungsansatz wurde in dem eigens entwickelten Plattformspiel *Night Mary* praktisch umgesetzt und erprobt. Zur Evaluierung des gesamten Systems wurde eine Benutzerbefragung mit zwölf Teilnehmern durchgeführt, die das Spiel jeweils fünf Minuten lang spielten und im Anschluss einen anonymen Fragebogen beantworteten, in dem ihre Meinung zu acht zentralen Qualitätskriterien der prozedural generierten Levels erhoben wurde. Die Auswertung der Fragebögen zeigte, dass die Spieler mit den Ergebnissen der Levelgenerierung grundsätzlich zufrieden waren. Gleichzeitig konnten aber auch einige Aspekte des Systems identifiziert werden, die noch Optimierungsmöglichkeiten bieten – mehr Abwechslungsreichtum, geringere Linearität und eine noch genauere Parametrisierung des Schwierigkeitsgrades sind denkbar.

7.2 Fazit

Die prozedurale Generierung von Spielinhalten – egal welcher Art – ist eine komplexe Angelegenheit. Viele Konzepte, die im Rahmen einer manuellen Produktion selbstverständlich erscheinen, lassen sich nur schwer in Computerprozeduren umsetzen und automatisieren. Die größte Herausforderung besteht darin, interessante Generierungsergebnisse von uninteressanten oder schlicht fehlerhaften zu unterscheiden. Daher wird meist eine Reihe von Einschränkungen definiert, um die Menge potenzieller Lösungen, aus der die Generierung schöpft, auf vielversprechende Teilmengen zu begrenzen. Dabei besteht jedoch die Gefahr, dass neuartige, kreative Lösungen, die außerhalb der vorgesehenen Grenzen liegen, bereits im Vorfeld der Generierung ausgeklammert werden und ihr Potenzial ungenutzt bleibt.

Genau in diesem Spannungsfeld zwischen emergenter Kreativität und notwendiger Kontrolle bewegen sich alle Ansätze zur prozeduralen Inhaltserstellung. Auch im Rahmen der vorliegenden Arbeit wurde schnell klar, dass die ursprüngliche Vision eines Systems, das eine nicht endende Vielfalt an Levels generiert, die selbst den Ersteller des Systems überraschen, nur mit vielen Einschränkungen und Kompromissen zu realisieren sein würde. Vor allem die Sicherstellung der Spielbarkeit und die Parametrisierung des Schwierigkeitsgrades in möglichst geringer Rechenzeit machten Einschnitte in den Entfaltungsspielraum notwendig, die zusätzlich zu vielen ungültigen Levelkonstellationen auch gültige und mitunter interessante Lösungen von der Generierung ausschließen.

Dennoch zeigt das erarbeitete Ergebnis in Form des Plattformspiels *Night Mary* das große Potenzial prozeduraler Ansätze zur Levelerstellung, sowohl für den Entwickler als auch für den Spieler. Neben der Reduktion der benötigten manuellen Ressourcen für bestimmte Aufgaben eröffnen dynamisch generierte Spielumgebungen nämlich die Möglichkeit ganz neuer Spielerfahrungen durch ihre Integration in die Spielmechanik. Im folgenden Ausblick wird unter anderem darauf eingegangen.

7.3 Ausblick

Die prozedurale Generierung von Spielinhalten wird zukünftig sicher eine bedeutende Rolle bei der Entwicklung von Spielen einnehmen. Sowohl für große Projekte mit Budgets in Millionenhöhe als auch für kleine Independent-Produktionen lohnt es sich, den Computer nicht nur als Werkzeug, sondern als zusätzlichen Partner bei der Inhaltserstellung zu betrachten. Das soll allerdings keinesfalls bedeuten, dass prozedurale Generierung manuelle Arbeit obsolet macht – es kommt maximal zu einer Verschiebung der Aufgaben, die im Idealfall mit einer besseren Nutzung der unterschiedlichen Stärken prozeduraler und manueller Techniken einhergeht.

Bereits heute ist der Offline-Einsatz von Programmen wie *SpeedTree* zur prozeduralen Erzeugung und Platzierung von Vegetation ein integraler Bestandteil vieler Spieleproduktionen. Besonders interessant ist allerdings, wie sich die Online-Generierung von Spielinhalten in Zukunft entwickeln wird, denn gerade hier ergeben sich große Chancen für gänzlich neue Spielkonzepte. Auch in der vorliegenden Arbeit wurde versucht, die dynamisch generierten Levels mit der Spielmechanik zu verweben, anstatt sie „nur“ als Ersatz für manuelles Leveldesign zu betrachten. So besteht das Spielziel in *Night Mary* nicht wie in einem herkömmlichen Plattformspiel darin, eine bestimmte Anzahl an Levels erfolgreich zu absolvieren, sondern darin, eine vorgegebene Anzahl an Sternen einzusammeln. Die Anzahl der eingesammelten Sterne beeinflusst aber gleichzeitig den Schwierigkeitsgrad – die prozedurale Levelgenerierung ist also an den Fortschritt des Spielers gekoppelt, was zu interessanten Wechselbeziehungen führt. Natürlich ist dies nur ein sehr simples Beispiel, das aber dennoch zeigt, wie dynamisch generierte Inhalte Spiele bereichern können.

Mögliche Erweiterungen der vorliegenden Arbeit sind also in vielen Bereichen denkbar. Einerseits bieten die Ergebnisse der durchgeführten Benutzerbefragung konkrete Ansatzpunkte für die Verbesserung aufgedeckter Schwachstellen. Andererseits ist auch eine weiterführende Auseinandersetzung mit dem festgestellten Konflikt zwischen Kreativität und Kontrolle im Rahmen der prozeduralen Generierung notwendig, um die bestehenden Lösungsansätze zu verbessern.

Anhang A

Inhalt der CD-ROM

A.1 Masterarbeit

Pfad: /

Handler2012.pdf	Masterarbeit

A.2 Literatur

Pfad: /Literatur/

*.pdf	Kopien der Literatur und Online-Quellen

A.3 Projektdateien

Pfad: /Projekt/

Executables/	Ausführbare Dateien für Windows und Mac OS X
Night Mary/	*Unity*-Projektordner mit sämtlichen Assets und Quellcode-Dateien

Quellenverzeichnis

Literatur

[1] Daniel Ashlock. „Automatic Generation of Game Elements via Evolution“. In: *Proceedings of the 2010 IEEE Symposium on Computational Intelligence and Games*. (Kopenhagen, Dänemark). Piscataway, USA: IEEE, 2010, S. 289–296.

[2] Franz Aurenhammer. „Voronoi Diagrams: A Survey of a Fundamental Geometric Data Structure“. In: *ACM Computing Surveys* 23.3 (Sep. 1991), S. 345–405.

[3] Kate Compton und Michael Mateas. „Procedural Level Design for Platform Games“. In: *Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference*. (Marina del Rey, USA). Palo Alto, USA: AAAI, 2006, S. 109–111.

[4] Chris Crawford. *Chris Crawford on Game Design*. Berkeley, USA: New Riders, 2003.

[5] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. New York, USA: Harper Perennial, 1990.

[6] Oliver Deussen und Bernd Lintermann. *Computergenerierte Pflanzen: Technik und Design digitaler Pflanzenwelten*. Berlin, Deutschland: Springer, 2002.

[7] David S. Ebert. *Texturing & Modeling: A Procedural Approach*. San Francisco, USA: Morgan Kaufmann, 2003.

[8] Michael Edwards. „Algorithmic Composition: Computational Thinking in Music“. In: *Communications of the ACM* 54.7 (Juli 2011), S. 58–67.

[9] Mark Hendrikx u. a. „Procedural Content Generation for Games: A Survey“. In: *ACM Transactions on Multimedia Computing, Communications and Applications* (2012), 1:1–1:24.

[10] Dietmar Jackèl, Stephan Neunreither und Friedrich Wagner. *Methoden der Computeranimation*. Berlin, Deutschland: Springer, 2006.

[11] Christopher Jefferson, Wendy Moncur und Karen E. Petrie. „Combination: Automated Generation of Puzzles with Constraints“. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. (Taichung, Taiwan). New York, USA: ACM, 2011, S. 907–912.

[12] Donald Knuth. *The Art of Computer Programming*. 3. Aufl. Boston, USA: Addison-Wesley, 1998.

[13] Raph Koster. *A Theory of Fun for Game Design*. Scottsdale, USA: Paraglyph Press, 2005.

[14] Michael Mateas und Andrew Stern. „Procedural Authorship: A Case-Study of the Interactive Drama Façade“. In: *Proceedings of the 6th Digital Arts and Culture Conference*. (Kopenhagen, Dänemark). Kopenhagen, Dänemark: IT-Universität Kopenhagen, 2005, S. 1–8.

[15] Peter Mawhorter und Michael Mateas. „Procedural Level Generation Using Occupancy-Regulated Extension“. In: *Proceedings of the 2010 IEEE Symposium on Computational Intelligence and Games*. (Kopenhagen, Dänemark). Piscataway, USA: IEEE, 2010, S. 351–358.

[16] Madjid Merabti u. a. „Interactive Storytelling: Approaches and Techniques to Achieve Dynamic Stories“. In: *Transactions on Edutainment I*. Hrsg. von Abdennour El Rhabili. Berlin, Deutschland: Springer, 2008, S. 118–134.

[17] Robert B. Miller. „Response Time in Man-Computer Conversational Transactions“. In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. (San Francisco, USA). New York, USA: ACM, 1968, S. 267–277.

[18] Pascal Müller. „Design und Implementation einer Preprocessing Pipeline zur Visualisierung prozedural erzeugter Stadtmodelle“. Diplomarbeit. Zürich, Schweiz: Eidgenössische Technische Hochschule Zürich, 2001.

[19] Fausto Mourato, Manuel Próspero dos Santos und Fernando Birra. „Automatic Level Generation for Platform Videogames Using Genetic Algorithms“. In: *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*. (Lissabon, Portugal). New York, USA: ACM, 2011, 8:1–8:8.

[20] Chris Pedersen, Julian Togelius und Georgios N. Yannakakis. „Modeling Player Experience in Super Mario Bros“. In: *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*. (Mailand, Italien). Piscataway, USA: IEEE, 2009, S. 132–139.

[21] Ken Perlin. „An Image Synthesizer“. In: *SIGGRAPH Computer Graphics* 19.3 (Juli 1985), S. 287–296.

[22] Xuexian Pi u. a. „Procedural Terrain Detail Based on Patch-LOD Algorithm“. In: *Proceedings of the 1st International Conference on Technologies for E-Learning and Digital Entertainment*. (Hangzhou, China). Berlin, Deutschland: Springer, 2006, S. 913–920.

[23] Przemyslaw Prusinkiewicz und James Hanan. *Lindenmayer Systems, Fractals and Plants*. Berlin, Deutschland: Springer, 1989.

[24] Przemyslaw Prusinkiewicz und Astrid Lindenmayer. *The Algorithmic Beauty of Plants*. Berlin, Deutschland: Springer, 1996.

[25] Katie Salen und Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. Cambridge, USA: MIT Press, 2004.

[26] Jesse Schell. *The Art of Game Design: A Book of Lenses*. San Francisco, USA: Morgan Kaufmann, 2008.

[27] Noor Shaker, Georgios N. Yannakakis und Julian Togelius. „Towards Automatic Personalized Content Generation for Platform Games“. In: *Proceedings of the 6th Artificial Intelligence and Interactive Digital Entertainment Conference*. (Stanford, USA). Palo Alto, USA: AAAI, 2010, S. 63–68.

[28] Ruben Smelik. „A Declarative Approach to Procedural Generation of Virtual Worlds“. Dissertation. Delft, Niederlande: Technische Universität Delft, 2011.

[29] Adam M. Smith u. a. „A Case Study of Expressively Constrainable Level Design Automation Tools for a Puzzle Game“. In: *Proceedings of the International Conference on the Foundations of Digital Games*. (Raleigh, USA). New York, USA: ACM, 2012, S. 156–163.

[30] Gillian Smith, Mee Cha und Jim Whitehead. „A Framework for Analysis of 2D Platformer Levels“. In: *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games*. (Los Angeles, USA). New York, USA: ACM, 2008, S. 75–80.

[31] Gillian Smith u. a. „Launchpad: A Rhythm-Based Level Generator for 2-D Platformers“. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.1 (März 2011), S. 1–16.

[32] Gillian Smith u. a. „Rhythm-Based Level Generation for 2D Platformers“. In: *Proceedings of the 4th International Conference on Foundations of Digital Games*. (Orlando, USA). New York, USA: ACM, 2009, S. 175–182.

[33] Nathan Sorenson und Philippe Pasquier. „The Evolution of Fun: Automatic Level Design through Challenge Modeling“. In: *Proceedings of the 1st International Conference on Computational Creativity*. (Lissabon, Portugal). New York, USA: ACM, 2010, S. 258–267.

[34] Julian Togelius u. a. „What is Procedural Content Generation? Mario on the Borderline“. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. (Bordeaux, Frankreich). New York, USA: ACM, 2011, 3:1–3:6.

[35] Mark J. P. Wolf. *The Video Game Explosion: A History from PONG to PlayStation and Beyond*. Westport, USA: Greenwood Press, 2008.

[36] Georgios N. Yannakakis und Julian Togelius. „Experience-Driven Procedural Content Generation“. In: *IEEE Transactions on Affective Computing* 2.3 (Juli 2011), S. 147–161.

Online-Quellen

[37] URL: http://www.arcade-museum.com/images/118/1181242103215.png (besucht am 08.07.2012).

[38] URL: http://en.wikipedia.org/wiki/File:NES_Super_Mario_Bros.png (besucht am 08.07.2012).

[39] URL: http://www.dashhacks.com/sites/default/files/psx_crash_bandicoot.jpg (besucht am 08.07.2012).

[40] URL: http://www.littlebigplanet.com/images/assets/shared/screenshots/story_temples_03.jpg (besucht am 08.07.2012).

[41] URL: http://www.bundysoft.com/L3DT/gallery/r25/09Jan07/new_erosion2.jpg (besucht am 26.07.2012).

[42] URL: http://www.speedtree.com/gallery/images/img4e32efca86fed.jpg (besucht am 26.07.2012).

[43] URL: http://www.esri.com/~/media/Images/Content/Software/cityengine/graphics/cs7.jpg (besucht am 26.07.2012).

[44] URL: http://www.geepers.co.uk/media/images/software/solidtextures.jpg (besucht am 26.07.2012).

[45] URL: http://fc07.deviantart.net/fs71/f/2010/077/f/3/Fire_particle_system_by_TheOtherGuy103.jpg (besucht am 26.07.2012).

[46] URL: http://en.wikipedia.org/wiki/File:Rogue_Unix_Screenshot_CAR.PNG (besucht am 27.07.2012).

[47] URL: http://eu.media.blizzard.com/d3/media/screenshots/barb-002-full.jpg (besucht am 27.07.2012).

[48] URL: http://djbarney.files.wordpress.com/2009/11/borderlands-2009-11-12-20-28-17-23.jpg (besucht am 27.07.2012).

[49] *List of Best-Selling Video Games*. Kopie auf CD-ROM (List of Best-Selling Video Games.pdf). URL: http://en.wikipedia.org/wiki/List_of_best-selling_video_games (besucht am 07.07.2012).

[50] Edmund McMillen. *Why Am I So... Hard?* Kopie auf CD-ROM (Why Am I So... Hard.pdf). 2010. URL: http://supermeatboy.com/13/Why_am_I_so____hard_/#b (besucht am 07.09.2012).

[51] Derek Yu. *The Full Spelunky on Spelunky.* Kopie auf CD-ROM (The Full Spelunky on Spelunky.pdf). 2011. URL: http://makegames.tumblr.com/post/4061040007/the-full-spelunky-on-spelunky (besucht am 25. 07.2012).