Dynamische Generierung von Spieleumgebungen
Procedural Content Generation

Seminar Spiele-KI

Eugen Geist
Fachhochschule Wedel
Fachrichtung: Wirtschaftsinformatik
Fach- und Verwaltungssemester: 6
Matrikelnummer: 100175

Inhaltsverzeichnis

1 Einleitung

Dynamische Generierung von Spieleumgebungen, im Englischen als ‚Procedural Content Generation' bezeichnet, im Folgenden PCG genannt, nimmt eine immer größere Rolle in der Entwicklung von Videospielen ein. PCG findet vor allem Anwendung in ‚Indie'-Spielen, wovon es durch die mittlerweile einfach zugänglichen Ressourcen immer mehr gibt. ‚AAA'-Spieleentwickler entdecken PCG jedoch auch immer mehr für sich.
PCG erledigt Aufgaben automatisch, die ansonsten von einem Entwickler oder Designer manuell erledigt werden müssten. Spieleumgebungen meint jedoch nicht nur Spielwelten, sondern jegliche Inhalte. Im Folgenden wird PCG zunächst definiert. Danach werden die Entstehungsgeschichte sowie verschiedene Ansätze zur Umsetzung von PCG beschrieben. Schwerpunkt sollen dabei der suchbasierte Ansatz und Compositional Pattern-Producing Networks sein. Diese sind interessant, da Methoden und Ansätze der KI zum Einsatz kommen.

2 Definition

Für PCG existiert keine eindeutige Definition, jedoch überschneiden sich die meisten Definitionen in den Hauptpunkten.
So definiert [Togelius, 2015c, Seite 1] „[...]PCG refers to computer software that can create game content on its own, or together with one or many human players or designers."
[Wiki, 2013, Abschnit What is Procedural Content Generation?] hingegen definiert PCG als „[...]the programmatic generation of game content using a random or pseudo-random process that results in an unpredictable range of possible game play spaces."
Der zentrale Aspekt beider Definitionen ist, dass es sich bei PCG um die programmatische Erstellung von Spieleinhalten handelt.
[Togelius, 2015c, Seite 1] definiert dabei ‚Content' als das was in den meisten Spielen enthalten ist: Levelkarten, Spielregeln, Geschichten, Aufgaben oder Aufträge, Gegenstände, Waffen, Vehikel und Charaktere.
[Wiki, 2013, Abschnitt: What is Procedural Content Generation?] schränkt PCG auch insofern ein, dass der generierte Content sich auf das Spielverhalten auswirken muss.
Ein weiterer Aspekt ist, dass eine Zufallskomponente enthalten sein muss, um eine möglichst große Vielfalt an Content zu ermöglichen [Wiki, 2013, Abschnitt: What is Procedural Content Generation?].
Zudem wird eine Interaktion mit Designern oder Spielern nicht ausgeschlossen [Togelius, 2015c, Seite 1]. Der Zeitpunkt der Generierung ist dabei nicht näher bestimmt. Manche Spiele generieren Inhalte zur Laufzeit, während bei anderen die Generierung vor dem Spielbeginn stattfindet. Bei wiederum

weiteren Spielen benutzen die Entwickler PCG als unterstützendes Entwicklungswerkzeug.

2.1 Abgrenzung zur normalen Spiele-KI

Normale oder klassische Spiele-KI wird als der Teil eines Spiels bezeichnet der Gegner oder kooperative Mitspieler intelligent erscheinen lässt. Er führt dazu, dass diese bei verschiedenen Entscheidungsmöglichkeiten schlau und nutzenorientiert agieren. [Schwab, 2008, Seite 2]
Hier ist ein klarer Unterschied zu PCG zu ziehen. PCG kann zum einen auch ohne KI zum Einsatz kommen, während das Verhalten von kooperativen oder gegnerischen NPCs immer als KI bezeichnet wird. Zum anderen verändert beziehungsweise erstellt PCG neue Spieleinhalte, während normale Spiele-KI mit dem Spieler oder der Umgebung interagiert.

3 Historischer Hintergrund

PCG tauchte zum ersten Mal 1978 in dem Spiel ‚Beneath Apple Manor' für den ‚Apple II' auf. Bekannter, und oft als die erste Verwendung von PCG genannt, ist ‚Rogue' aus dem Jahr 1980. Beide Spiele sind sogenannte ‚Dungeon Crawler', Adventures, in denen der Spieler rundenbasiert Höhlen erkundet, Gegenstände sammelt und gegnerische Monster bekämpft. In beiden Spielen wird die komplette Spielwelt zu Beginn des Spiels generiert. Die Gründe hierfür sind vor allen Dingen die damals sehr begrenzten Hardware-Ressourcen. Obwohl von Entwicklern erstellte Welten für den Spieler schöner und besser spielbar sind, benötigt der Programmcode für die Erstellung der Welt weniger Speicher, sodass diese Methode gewählt wurde. Ein positiver Nebeneffekt davon ist, dass bei jedem Spiel eine komplett neue Welt entsteht, die der Spieler erkunden muss. Das erhöht den Wiederspielwert und trägt deutlich zur Beliebtheit von Rogue bei. Spiele mit dem gleichen oder einem ähnlichen Spielprinzip wie Rogue werden auch ‚Roguelikes' genannt.
Ein anderes Beispiel ist ‚Elite', eine Weltraum-Flug- und Wirtschaftssimulation aus dem Jahr 1984. Die Spielwelt besteht aus 8 Galaxien á 256 Planeten. Diese Planeten besitzen Eigenschaften wie Bevölkerungsdichte, Sprache, Rohstoffvorkommnisse, diplomatische Einstellung und Handelsabkommen. Da bei 2048 Planeten für die Speicherung der Eigenschaften sehr viel Speicherplatz gebraucht werden würde, werden sie in Elite von einem Algorithmus anhand einer zufälligen Seednummer generiert. Statt der Eigenschaften der Planeten wird nur die Seednummer abgespeichert und die Eigenschaften bei Bedarf generiert. Ein positiver Nebeneffekt dabei ist, dass bei Beginn eines neuen Spiels die Eigenschaften der Planeten verschieden von denen vorher sein können. Ein negativer Nebeneffekt ist, dass die Eigenschaften der Planeten sich nicht ändern können, da sie immer aus der Seednummer generiert und nicht explizit gespeichert werden.

KI kam in den Anfängen des PCG jedoch nicht zum Einsatz. Die meisten Ansätze generierten aus einem Startwert, zum Beispiel einer Seednummer, mithilfe eines fest vorgegebenen Algorithmus Inhalte ohne dabei verschiedene Entscheidungsmöglichkeiten abzuwägen.

4 Ansätze in Procedural Content Generation

4.1 Konstruktiver Ansatz

Der konstruktive Ansatz verfolgt das Ziel, dem Zufall eine Richtung zu geben, damit die generierten Inhalte strukturiert wirken. Dabei werden modulare, vordefinierte Teile zusammengesetzt und bilden so den generierten Inhalt. Zum Beispiel können einzelnen modularen Teilen Nummern zugewiesen werden. Durch einen Zufallszahlengenerator erfolgt dann die Auswahl, wo welche Teile platziert werden.
Algorithmisch ist der konstruktive Ansatz leicht umzusetzen. Die Schwierigkeit besteht darin, die modularen Teile so zu definieren, dass sie in jeglichen Kombinationen zueinander passen und spielbar sind. [Togelius, 2015a, Abschnitt Constructive Methods]

Ein Beispiel für den konstruktiven Ansatz ist ‚Spelunky'. Spelunky ist eine Mischung aus 2D Jump'n Run und Roguelike, bei dem der Spieler eine Spielfigur in Echtzeit durch ein Höhlensystem steuert. [Gamespot, 2013, ab 30 Sekunden] zeigt, wie das Spielprinzip aussieht.
Die einzelnen Level in Spelunky werden mithilfe des konstruktiven Ansatzes zusammengebaut. Ein Level wird in 4 x 4 Blöcke unterteilt und ein Pfad vom Eingang zum Ende des Levels wird eingezeichnet. Abhängig von der Position des Blocks auf dem Weg zum Ausgang wird eine Vorlage ausgewählt. Dieser Vorlage werden dann zufällige Wegelemente, wie zum Beispiel Kisten, die durch einen Sprung überbrückt werden müssen, hinzugefügt. Dies geschieht so, dass sie den Weg vom Ein- zum Ausgang nicht blockieren. Am Ende werden noch zufällige Gegenstände wie Schatztruhen und Goldmünzen, jedoch auch Gegner und Fallen hinzugefügt. Diese sind so modular gestaltet, dass sie in jede Vorlage passen. [Yu, 2011]

Beim konstruktiven Ansatz kommen keine Methoden der KI zum Einsatz. Der Zufall und die vordefinierten einzelnen Teile entscheiden über die generierten Inhalte.

4.2 Grammatik-Ansatz

Der Grammatik Ansatz definiert Symbole, Ontologien, Produktionsregeln und Axiome. Die Produktionsregeln bestimmen, wie Symbole expandiert werden können. Ein Beispiel dafür sind mathematische Lindenmayer-Systeme,

auch L-Systeme genannt, die unter anderem in der Automaten- und formalen Sprachentheorie zum Einsatz kommen.
Grammatiken ermöglichen die Erzeugung von komplexen Strukturen aus kleinen Teilen. Da eine Grammatik immer den gleichen Regeln folgt, ist einer der größten Nachteile, dass die generierten Inhalte zu regelmäßig sind und somit vorhersehbar werden. Andererseits kann genau das auch zum Vorteil genutzt werden, um Level, Missionen oder Gegenstände nach einem bestimmten Schema zu erstellen. [Togelius, 2015a, Abschnitt: Grammars]
Abb. 1 zeigt beispielhaft eine solche Grammatik. Die Produktionsregeln

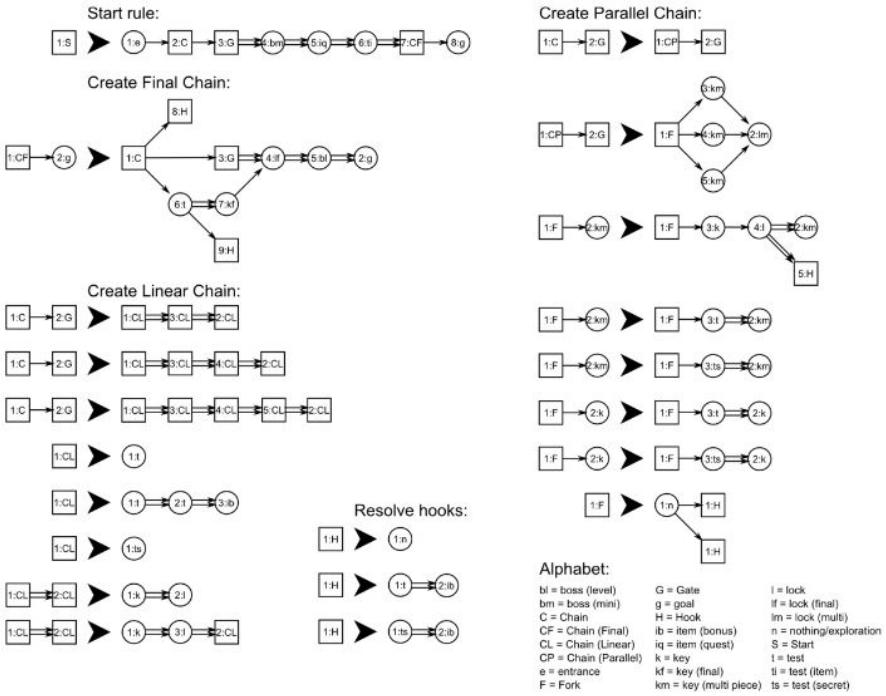

Abbildung 1: Grafische Grammatik, die benutzt wird, um Missionen zu generieren.

sind so aufgebaut, dass die Symbole auf der linken Seite durch die Symbole auf der rechten Seite ersetzt werden. Nichtterminalsymbole sind mit großen Buchstaben bezeichnet. Für jedes Nichtterminalsymbol existiert mindestens eine Produktionsregel, die dieses durch Nichtterminal- oder Terminalsymbole ersetzt. Ein solches Nichtterminalsymbol ist beispielweise eine Gabelung, in der Grammatik als „Fork“ bezeichnet. Terminalsymbole werden mit kleinen Buchstaben bezeichnet. Sie sind nicht weiter ersetzbar, können jedoch in Produktionsregeln benutzt werden, um Symbole zu ersetzen oder Nichtterminalsymbole in Kontext zu setzen. So kann der finale Weg, in der Grammatik als ‚Final Chain’ bezeichnet, nur ersetzt werden, wenn darauf ein Ziel, also ‚Goal’, folgt. Die Mission ist generiert, sobald keine Nichtterminalsymbole

mehr im Graphen existieren.
Die Auswahl der Produktionsregeln kann zufällig oder gesteuert erfolgen. Wenn sie gesteuert erfolgt, kann sie sich unter anderem auf die Aktionen des Spielers beziehen. Falls dieser schon viele Gegner besiegt hat, könnte die generierte Mission mehr Rätsel statt Gegner enthalten, um eine möglichst große Vielfalt zu bieten. [Dormans, 2011, Seite 7]
Axiome können unter anderem die Auswahl der Produktionsregeln einschränken, sodass Regeln öfter oder seltener angewandt werden.

Der Grammatik Ansatz verwendet in der Regel keine Methoden der KI, da die Grammatiken vorher von Menschen definiert und dann expandiert werden. Die Auswahl der Produktionsregeln anhand der Aktionen des Spielers, wie [Dormans, 2011, Seite 7] sie vorschlägt, ist ein Schritt in Richtung KI, da hier nicht nur zufällige, sondern gerichtete, dynamische und nutzenorientierte Entscheidungen getroffen werden.

4.3 Restriktionsbasierte Systeme

Bei restriktionsbasierten Systemen wird eine Domäne definiert, die aus logischen und numerischen Variablen und Restriktionen besteht. Ein Solver löst Probleme in dieser Domäne, indem er die Variablen variiert und dabei die Restriktionen beachtet. Gewünschte Eigenschaften können durch die Restriktionen modelliert und ungewünschte vermieden werden. [Togelius, 2015a, Abschnitt: Constraint-Based Systems]
Restriktionen müssen dabei mit Bedacht gewählt werden, da sie den größten Einfluss darauf haben, was der Solver für Ergebnisse liefert. Den größten Nutzen liefert dieser Ansatz, wenn feste Designentscheidungen vorliegen und diese umgesetzt werden müssen, da sie direkt in Restriktionen umgesetzt werden können. Andererseits muss dabei auch beachtet werden, dass keine Fälle generiert werden, die zu unspielbaren Inhalten führen. [Togelius, 2015a, Abschnitt: Constraint-Based Systems]

Höhlenlevel in einem Roguelike können zum Beispiel mit einem restriktionsbasierten System generiert werden. Die Anzahl der verschiedenen Räume, die maximale Raumgröße, die Anzahl der Schätze und Gegner pro Raum können dabei als Variablen dienen. Die Restriktionen legen fest, wieviele verschiedene Räume, wieviele Räume mit Gegnern oder Schatztruhen existieren und wieviele Gegner maximal und minimal pro Raum existieren dürfen. Außerdem können die Restriktionen noch festlegen, dass am Startpunkt des Spielers keine Gegner erscheinen, die ihn direkt nach dem Start töten, da dieser ansonsten keine Möglichkeit zu reagieren hat.
Die restriktionsbasierten Systeme haben ihren Ursprung in der logischen und Constraintprogrammierung. Dadurch ist es möglich, bereits existierende No-

tationen und Solver zu verwenden. Die restriktionsbasierten Systeme nutzen Methoden der KI, sind dabei in der PCG jedoch noch an ihren Anfängen und müssen weiter erforscht werden.

4.4 Suchbasierter Ansatz

Der suchbasierte Ansatz verwendet Suchmethoden aus der KI, wie evolutionäre Algorithmen, um die gewünschten Inhalte zu finden und wird deswegen auch oft „Optimierung“ genannt. Er besteht aus drei Komponenten: Suchalgorithmus, Inhaltsrepräsentation und Evaluationsfunktion. [Togelius, 2015b, Seite 17]

Der Grundablauf sieht wie folgt aus:

- Ein Suchraum mit möglichen Lösungen wird bestimmt.
- Einer oder mehrere Lösungskandidaten werden iterativ entwickelt.
- Entwicklungen, die Lösungskandidaten von einer möglichen Lösung entfernen, werden verworfen.
- Entwicklungen, die Lösungskandidaten einer möglichen Lösung näherbringen, werden beibehalten.
- Irgendwann wird eine zufriedenstellende Lösung erreicht.

4.4.1 Suchalgorithmus

Der Suchalgorithmus ist für die Suche im definierten Suchraum verantwortlich. Er ist meist simpel und sollte kontextabhängig gewählt werden.
Bei wenigen Einschränkungen für die generierten Inhalte und einfacher Auffindbarkeit von Lösungen kann eine zufällige Suche gewählt werden. Diese ist zum einen durch einen Zufallszahlengenerator leicht zu verwirklichen und führt zum anderen zu einer großen Ergebnisvielfalt.
Sollte der Lösungsraum klein sein oder die Laufzeit bei der Generierung keine relevante Rolle spielen, so können Breiten- oder Tiefensuche zum Einsatz kommen. Breiten- und Tiefensuche expandieren viele Lösungskandidaten, weswegen die Generierung bei einem großen Lösungsraum unter Umständen länger dauern kann.
Eine weitere Möglichkeit wäre der Einsatz von Schwarmintelligenz in Form von Ameisenalgorithmen. Am meisten verbreitet sind jedoch Evolutionäre Algorithmen. [Togelius, 2015b, Seite 18ff.]

Evolutionäre Suchalgorithmen sind eine Gruppe von stochastischen Algorithmen, die an die darwinistische Evolutionslehre angelehnt sind.
Die Grundidee der evolutionären Algorithmen ist, dass eine Menge von Lösungskandidaten pro Generation evaluiert und bewertet wird. Die Kandidaten mit den besten Bewertungen bekommen eine Chance sich zu vermehren beziehungsweise zu mutieren, während die Kandidaten mit den schlechtesten Bewertungen verworfen werden. Nach mehreren Iterationen entstehen Lösungskandidaten, die besser sind als die der Vorgenerationen, da nur die Besten weiterentwickelt werden. Die Suche wird abgebrochen, sobald eine zufriedenstellende Lösung gefunden oder eine vorher definierte Generationenanzahl erreicht wird.
Einer der simpelsten, jedoch voll funktionalen, evolutionären Algorithmen ist die ‚μ + λ evolution strategy(ES)'. μ repräsentiert dabei den Anteil der Population, der zwischen den Generationen erhalten bleibt. λ repräsentiert den Anteil der Population, der zwischen den Generationen neu erstellt wird. Beispielhafter Ablauf mit $\mu = \lambda = 50$:

1. Population der $\mu+\lambda$ Individuen initialisieren (per Hand, aus vorherigen Vorgängen oder per Zufallsgenerator).

2. Alle Individuen evaluieren und eine Bewertung (numerisch, ordinal) hinzufügen.

3. Population absteigend nach Bewertung sortieren.

4. Die λ (also 50) schlechtesten Individuen entfernen.

5. Die λ entfernten Individuen mit Kopien von den verbleibenden μ ersetzen: der Nachwuchs (alle μ Individuen einmal kopieren falls $\mu = \lambda$, ansonsten die Besten auswählen beziehungsweise mehrmals kopieren).

6. Den Nachwuchs mutieren, zum Beispiel durch Zufallsoperationen (beste Mutationsoperationen hängen stark von der Inhaltsrepräsentation und der Evaluationsfunktion ab).

7. Falls eines der Individuen die erforderlichen Qualitätskriterien erfüllt oder die maximale Anzahl Generationen erreicht ist, wird gestoppt, ansonsten wird bei Schritt 2 fortgefahren.

Die μ + λ evolution strategy(ES) ist nur ein evolutionärer Algorithmus. Andere evolutionäre Algorithmen sind zum Beispiel Genetische Algorithmen. Diese setzen mehr auf Rekombinationen der einzelnen Kandidaten statt auf Mutationen. Für Fälle, in denen mehrere Evaluationsfunktionen zum Einsatz kommen, sollten, statt der Summe der einzelnen Evaluationsfunktionen, entsprechende Algorithmen, wie der ‚NSGA-II', benutzt werden. [Togelius, 2015b, Seite 18 f.]

4.4.2 Inhaltsrepräsentation

Die Inhaltsrepräsentation bestimmt wie die Lösungen und der Suchraum abgebildet werden. Sie ist sehr wichtig, da hierdurch bestimmt wird, was für Lösungen möglich sind und wie effizient der Suchalgorithmus arbeitet. Die Wahl der Inhaltsrepräsentation sollte anhand des zu lösenden Problems und der erwünschten Inhalte erfolgen.
Um eine Unterscheidung zwischen der Repräsentation der möglichen Lösungen im Suchraum und den generierten Inhalten zu treffen, werden die Begriffe ‚Genotypen' und ‚Phänotypen' aus der Genetik verwendet.
Genotypen bezeichnen die Elemente in der Menge der möglichen Lösungen. Diese können Instruktionen sein, wie ein Level zu zeichnen ist, oder numerische Arrays mit den Daten eines Levels. Genotypen können auch als „Blaupause" von den zu generierenden Inhalten verstanden werden.
Phänotypen sind die generierten Inhalte. Es findet eine Konvertierung von Genotypen in Phänotypen statt, die von der Art der Geno- und Phänotypen abhängt. Bei einem Genotypen, der beispielsweise Instruktionen zur Platzierung von Gegenständen und Gegnern enthält, werden diese Anweisungen ausgeführt und das fertige Level, der Phänotyp, entsteht. [Togelius, 2015b, Seite 20]
Eine passende Analogie ist in der Objektorientierten Programmierung anzutreffen: Dort können die Klassen, die den Aufbau eines Objektes beschreiben, als Genotypen und die Instanzen der Objekte als Phänotypen bezeichnet werden.
[Togelius, 2015b, Seite 20f] führt ein Beispiel an, das die Auswirkungen der unterschiedlichen Inhaltsrepräsentationsweisen deutlich werden lässt. Dazu werden verschiedene Repräsentationen von Leveln des Spiels „Super Mario Bros." aufgezeigt:

1. Direkte Repräsentation, indem jeder Block im Phänotypen auf einer Variablen im Genotypen abgebildet wird,
2. Liste von Positionen und Eigenschaften der einzelnen Spieleobjekte,
3. Sammlung von wiederverwendbaren Objekten und seperaten Listen, wie die Spielobjekte im Level verteilt sind,
4. Liste von wünschenswerten Eigenschaften, wie Anzahl der Gegner, Münzblöcke, Röhren und Lücken auf dem Weg und
5. Zufallszahl

Diese verschiedenen Ansätze zeigen vor allem auf, wie sehr die Repräsentation den Suchraum beeinflusst. Zunächst wäre die Annahme, dass Repräsentation 1 die beste ist, da sie dem generierten Inhalt am meisten ähnelt. Die hohe Anzahl der Dimensionen, in diesem Fall alle Blöcke im Level, führen

jedoch zu einem sehr großen Suchraum, der es erschwert, eine geeignete Lösung zu finden. Generell kann gesagt werden, je größer ein Suchraum, desto schwieriger ist es eine befriedigende Lösung zu finden. [Togelius, 2015b, Seite 21]
Ein weiterer wichtiger Faktor ist hohe Lokalität. Hohe Lokoalität meint, dass kleine Änderungen im Genotypen zu kleinen Änderungen im Phänotypen und einer kleinen Änderung der Bewertung führen. Wenn keine hohe Lokalität herrscht, können keine zielgerichteten Verbesserungen an den Genotypen vorgenommen werden. Die Suche kann dann nicht effizient erfolgen und ähnelt mehr einer zufälligen Suche. Das führt dazu, dass die Repräsentation 5 ungeeignet ist. Diese macht es nicht möglich abzuschätzen, was für Änderungen am Phänotypen erfolgen, wenn die Genotypen geändert werden. [Togelius, 2015b, Seite 21]
Die Inhaltsrepräsentation bestimmt außerdem, was für Inhalte generiert und welche Eigenschaften dieser beeinflusst werden können. Bei der Repräsentation 4 zum Beispiel können nur Inhalte generiert werden, die aus den beschriebenen Eigenschaften bestehen, da nur diese beeinflussbar sind. Die restlichen Eigenschaften der Level sind statisch. Bei Repräsentation 1 hingegen kann jeder Block frei gewählt werden, sodass nahezu beliebige Inhalte generiert werden können. [Togelius, 2015b, Seite 22]
Schlussendlich lässt sich sagen, dass die Inhaltsrepräsentation immer anhand des zu generierenden Inhaltes und der Rahmenbedingungen, wie Laufzeit und Speicherkapazität, zu wählen ist.

4.4.3 Evaluationsfunktion

Die Evaluationsfunktion betrachtet Lösungskandidaten und bewertet sie. Die Bewertung der Evaluationsfunktion führt zur Selektion der besser geeigneten Lösungskandidaten und schlussendlich zur Lösung. Sie beeinflusst die Effizienz der Suche nach geeigneten Inhalten und die Qualität der Inhalte maßgeblich.
Dementsprechend sollte die Evaluationsfunktion die Eigenschaften der Lösungskandidaten, die in den generierten Inhalten erwünscht sind, besser bewerten als unerwünschte Eigenschaften. Die Bewertung der Eigenschaften ist dabei zum einen stark von den Inhalten selbst, beispielsweise Level, Waffen und Gegenstände sowie zum anderen von den Indikatoren für gute Qualität abhängig. Die Indikatoren für gute Qualität sind schwierig festzustellen, da für jede Person andere Eigenschaften wichtig sind. Allgemein kann dabei festgehalten werden, dass Eigenschaften, die die Spielbarkeit begünstigen, gut und Eigenschaften, die Inhalte unspielbar machen, schlecht bewertet werden sollten. Andere Qualitäten müssen von Designern bestimmt und modelliert werden, da dies weniger eine Entwicklungsfrage sondern mehr eine Frage nach den Eigenschaften der generierten Inhalte ist.[Togelius, 2015b, Seite 22]
Generell können Evaluationsfunktionen in drei verschiedene Arten unterteilt

werden: direkte, simulationsbasierte und interaktive Evaluationsfunktionen.

Direkte Evaluationsfunktionen ordnen Eigenschaften aus den Genotypen Qualitätswerte zu. Das hat den Vorteil, dass die Bewertung wenig Rechenzeit erfordert und leicht zu implementieren ist. Zudem können Eigenschaften, die der Spieler bevorzugt, wie zum Beispiel viele Schatztruhen in einem Level, direkt als positiv eingestuft werden. Andererseits ist es für manche Eigenschaften schwierig Bewertungen festzulegen, da zum Beispiel nicht pauschal gesagt werden kann, ob mehr oder weniger Gegner in einem Level positiv zu bewerten sind. [Togelius, 2015b, Seite 23]

Simulationsbasierte Evaluationsfunktionen benutzen KI-Agenten, um die generierten Inhalte durchzuspielen und anhand dessen zu bewerten. Dabei werden für die Bestimmung des Verhaltens der KI-Agenten und die Durchführung der Bewertung Statistiken benutzt. Die Fähigkeiten der KI-Agenten sind von den Aufgaben bestimmt, die sie zur Bewertung der Inhalte durchführen. Muss ein KI-Agent zum Beispiel Labyrinthe auf Spielbarkeit prüfen, so muss er Wegfindungsmethoden beherrschen. Zudem können KI-Agenten statisch sein, also immer die gleichen Aktionen durchführen, oder dynamisch, sodass sie ihre Aktionen im Laufe der verschiedenen Evaluationen anpassen. [Togelius, 2015b, Seite 23f]

Interaktive Evaluationsfunktionen basieren auf der Interaktion mit Menschen. Die Interaktion kann dabei mit Spielern oder Designern, die PCG-Methoden als Werkzeug benutzen, erfolgen. Die Interaktion geschieht explizit oder implizit. Bei der expliziten Interaktion gibt der Mensch seine Präferenzen an, sodass die Inhalte anhand dessen bewertet werden können. Dabei können die Präferenzen genauer festgehalten werden, es kommt jedoch zu Unterbrechungen im Spielfluss. Bei der impliziten Interaktion werden die Aktionen des Menschen gedeutet und umgesetzt. Beispielsweise wird bei einem Gegenstand, den der Spieler oft benutzt, gedeutet, dass die Eigenschaften dieses Gegenstandes ansprechend sind, sodass andere Gegenstände, die gleiche oder ähnliche Eigenschaften besitzen, positiv bewertet werden. Die Präferenzen des Spielers werden dabei nicht so genau erfasst wie bei der expliziten Methode, jedoch kann die Erfassung im Hintergrund geschehen. Eine Mischung aus beiden Ansätzen ist möglich, um die jeweiligen Nachteile auszugleichen. [Togelius, 2015b, Seite 24]

4.4.4 Beispiel: StarCraft

[Togelius, 2015b, Seite 24f] zeigt anhand von ‚StarCraft' wie die einzelnen Komponenten des suchbasierten Ansatzes implementiert werden können. StarCraft ist ein Echtzeit-Strategiespiel aus dem Jahr 1998. Trotz des hohen Al-

ters ist die Beliebtheit von StarCraft vor allem in der koreanischen Progaming-Szene hoch und nimmt erst in den vergangenen Jahren ab. Die Schwerpunkte des Spiels liegen auf dem Abbau von Rohstoffen, dem Aufbau von Basen, der Produktion von Einheiten und dem Bekriegen des gegnerischen Spielers mithilfe der produzierten Einheiten. Wegen der hohen Beliebtheit im Progaming müssen die StarCraft Level, genannt Maps, ausgeglichen und für alle Spieler fair aufgebaut sein. So darf ein Spieler nicht zu mehr Ressourcen Zugang haben oder eine strategisch bessere Position auf der Karte besitzen als sein Gegenspieler.

Repräsentation Die einzelnen Maps werden als Vektoren mit ungefähr 100 Dezimalziffern repräsentiert. Bei der Umsetzung der Genotypen in die Phänotypen werden die einzelnen Zahlen unterschiedlich interpretiert. Manche werden direkt als Positionen der Basen der einzelnen Spieler oder Ressourcen umgesetzt, während andere Anweisungen für sogenannte ‚Turtle-Zeichenoperationen' darstellen, um unpassierbares Terrain zu füllen. Die Phänotypen werden als zwei dimensionales Array repräsentiert, bei dem jede Zelle einem Block in StarCraft entspricht. Dieses Format lässt sich in das StarCraft Map Format umwandeln. [Togelius, 2015b, Seite 25]

Evaluation Bei der Entwicklung der StarCraft Maps werden acht verschiedene Evaluationsfunktionen verwendet, die die Platzierung der Basen und Ressourcen sowie die Wege zwischen den Basen evaluieren. Die meisten Evaluationsfunktionen basieren vor allem auf A* Wegberechnungen zwischen einzelnen Punkten und Auswertungen von begehbarem, nicht anderweitig belegten, Terrain. So werden unter anderem die Abstände der Basen zueinander und zu den verschiedenen auf den Maps platzierten Ressourcen ausgewertet. Die meisten der Evaluationsfunktionen arbeiten direkt auf den Genotypen-Daten, sind also direkte Evaluationsfunktionen. Eine komplexere Evaluationsfunktion bewertet Engpässe auf den Wegen zwischen Basen als positiv, da diese von taktisch versierten Spielern für Manöver genutzt werden können, um eine gegnerische Übermacht zu bezwingen. [Togelius, 2015b, Seite 25]

Algorithmus Aufgrund der Anzahl der Evaluationsfunktionen ist es sehr schwierig diese in einem einzigen Ziel zu formulieren [Togelius, 2015b, Seite 25]. Der ‚SMS-EMOA', ein evolutionärer Algorithmus, der auf mehrere Ziele hinarbeitet, wird verwendet, um auf zwei bis drei Ziele hinzuarbeiten. Es ist nicht möglich Maps mit komplett maximierten Zielwerten zu generieren, da Konflikte zwischen den einzelnen Zielen existieren. Es können jedoch unterschiedliche, sprich spielbare und interessante, Maps mit verschiedenen Eigenschaften generiert werden. [Togelius, 2015b, Seite 25]

4.5 Compositional Pattern-Producing Networks

Compositional Pattern-Producing Networks, im Folgenden CPPN genannt, ermöglichen das Erstellen von lebensnahen beziehungsweise -ähnlichen Mustern. Lebensnahe Muster sind solche, die vor allem in der Natur vorkommen, wie zum Beispiel Blumen- oder Blättermuster. CPPN sind von den neuronalen Netzen aus der KI abgeleitet und unterscheiden sich nur in der Aktivierungsfunktion. Sie sind dem suchbasierten Ansatz zuzuordnen, da sie eine spezielle Inhaltsrepräsentationsweise darstellen. [Ashlock, 2015, Seite 165]

4.5.1 Neuronale Netze

‚Neuronale Netze' sind verbundene Gruppen von Knoten, die Werte basierend auf externen Eingabesignalen berechnen. Die Berechnung erfolgt, indem die Signale von den Eingabeneuronen durch das Netzwerk zu den Ausgabeneuronen übertragen werden. Neuronen, die keine Eingabe- oder Ausgabeneuronen sind, werden versteckte Neuronen genannt.
Jedes Neuron i berechnet aus allen Eingabesignalen x_j, wobei j die Eingabeknoten sind, gewichtet mit den Gewichten w_{ij} eine Ausgabe y_i, deren Ausmaß durch die Funktion σ bestimmt wird:

$$y_i = \sigma \cdot (\sum_{j}^{N} w_{ij} \cdot x_j)$$

Bei klassischen neuronalen Netzen dient entweder eine sigmoid-:

$$\sigma(x) = \frac{1}{1 + e^{-kx}}$$

oder eine Gauß-Funktion als Aktivierungsfunktion.
Das Verhalten von neuronalen Netzen ist hauptsächlich durch die Verbindungen und Gewichtungen bestimmt.
Abbildung 2 zeigt ein neuronales Netz mit der Gauß-Funktion als σ-Funktion.

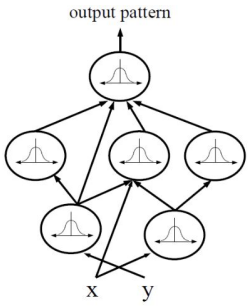

Abbildung 2: Beispielhaftes Neuronales Netz

x und y sind dabei die Signale, die an den Eingabeneuronen in das Netz eingegeben werden, der ‚output pattern' ist die Ausgabe.

Neuronale Netze werden in der Spiele-KI normalerweise verwendet, um NPC-Verhalten zu modellieren und zu steuern. Sie können jedoch auch als Evaluationsfunktion für den suchbasierten Ansatz benutzt werden. [Ashlock, 2015, Seite 162f]

4.5.2 Compositional Pattern-Producing Networks

CPPN sind modifizierte neuronale Netzte, die als Mustergeneratoren verwendet werden. CPPN benutzen im Gegensatz zu neuronalen Netzen oftmals Varianten aus gaußschen, sigmoiden, sinus und vielen anderen Funktionen als Aktivierungsfunktion σ. Die Aktivierung und Funktionsweise sind dieselben wie bei neuronalen Netzen.
Typischerweise werden periodische Funktionen verwendet, um sich wiederholende und symmetrische Muster, wie sie in der Natur vorkommen, zu erzeugen.

Abbildung 3 zeigt ein CPPN mit verschiedenen Aktivierungsfunktionen an

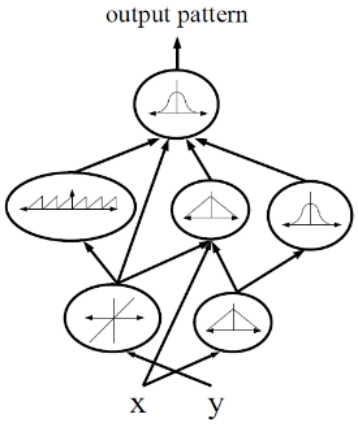

Abbildung 3: Beispielhaftes CPPN

den verschiedenen Knoten. Die Benutzung erfolgt, wie beim neuronalen Netz, durch die Eingabe der Signale x und y.
CPPN haben einen weitaus größeren Anwendungsbereich als neuronale Netze, da sie durch die Vielfalt der Aktivierungsfunktionen frei modellierbar sind. Ein beliebter Anwendungszweck sind Bildgeneratoren und -mutatoren. Abbildung 4 zeigt ein CPPN, das als Input x- und y-Koordinaten eines Bildes entgegennimmt und als Ausgabe die Farbe an dem Punkt liefert. Das später vorgestellte Spiel ‚Petalz' nutzt ein komplexeres CPPN, um das Aussehen von Blumen anhand der Koordinaten zu generieren. [Ashlock, 2015, Seite 162ff]

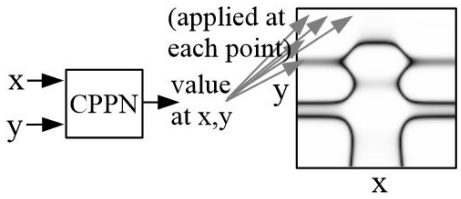

Abbildung 4: CPPN für Erstellung von Bildern

4.5.3 Beispiel: Galactic Arms Race

‚Galactic Arms Race' ist ein Weltraumshooter aus dem Jahr 2010, in dem der Spieler ein Raumschiff steuert und gegnerische Raumschiffe mit verschiedenen Waffen besiegen muss. Das Verhalten der Waffen des Spielers wird durch CPPN umgesetzt.
Jede Waffe in Galactic Arms Race wird durch ein eigenes CPPN repräsentiert. Je öfter ein Spieler eine bestimmte Waffe benutzt, desto mehr nimmt das Spiel an, dass der Spieler die Eigenschaften der Waffe favorisiert und generiert Waffen mit ähnlichen Eigenschaften.
Ein CPPN in Galactic Arms Race berechnet anhand der aktuellen Position eines Projektils einer Waffe die Bewegungsrichtung und Farbe des Projektils, also wie die Flugbahn und Farbe der einzelnen Projektile sich verhalten. Es hat vier Eingabesignale: p_x, p_z, d_c und *bias* sowie vier Ausgabesignale: v_x, v_z, r, g und b (siehe Abbildung 5).

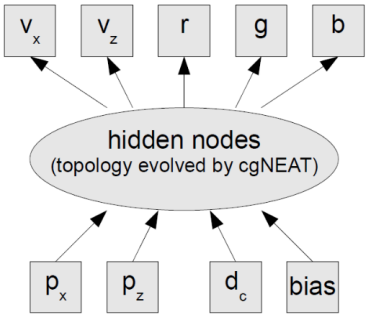

Abbildung 5: Galactic Arms Race CPPN

Eingabesignale

- p_x und p_z sind die Koordinaten der aktuellen Position des Projektils.
- d_c, gibt an wie weit sich das Projektil bereits vom Abschusspunkt entfernt hat.
- *bias* ist eine Zufallszahl um das Verhalten der Projektile unvorhersehbarer zu machen, zum Beispiel durch eine Streuung.

Ausgabe:

- v_x und v_y sind Bewegungsvektoren, die die Richtung und Geschwindigkeit des Projektils angeben.
- r, g und b sind die einzelnen Komponenten der Farbe, die das Projektil hat.

Für jedes Waffenprojektil wird das zuständige CPPN einmal pro Bild mit den aktuellen Signalen aufgerufen, sodass Bewegungsrichtung, -geschwindigkeit und Farbe des Projektils für das nächste Bild berechnet werden. [Ashlock, 2015, Seite 167f]
[SplatterCatGaming, 2013] zeigt verschiedene Waffen in Galactic Arms Race. Ab Sekunde 25 ist eine Waffe zu sehen, deren CPPN die Projektile in Bögen weg vom Abschusspunkt bewegt, während die Waffe zu Anfang nur Projektile mit einer linearen Schussbahn abfeuert.

4.5.4 Beispiel: Petalz

Petalz ist ein soziales Netzwerkspiel auf Facebook, in dem Spieler Pflanzen züchten und diese auf ihrem virtuellen Balkon ausstellen können. Durch Bestäubung der Pflanzen oder Kombination verschiedener Pollen können Spieler neue Pflanzen züchten und diese an andere Spieler verkaufen.
Die Pflanzen in Petalz werden durch CPPN generiert. Jeder Pflanze liegt ein anderes, durch NEAT entwickeltes, CPPN zugrunde. NEAT, Neuroevolution of Augmenting Topologies, ist eine Methode um neuronale Netze und CPPN durch inkrementielles Hinzufügen von versteckten Knoten zu entwickeln. [Ashlock, 2015, Seite 164f]
Die Grundidee bei der Generierung der Pflanzen ist es, einen Kreis zu formen und diesen mit verschiedenen Farben zu füllen. Dazu wird mithilfe des CPPN zunächst für jeden Winkel der maximale Radius ausgerechnet. Danach werden die Farben aller Punkte berechnet, indem das CPPN mit jedem Winkel und Radius zwischen 0 und dem berechneten maximalen Radius aktiviert wird. Außerdem ermöglicht die Änderung eines Eingabesignals die Berechnung verschiedener Höhenschichten der Blume. Abbildung 6 zeigt eine Blume aus Petalz und das dahinterliegende CPPN.

Eingabesignale

- r gibt den Radius des Punktes von der Mitte der Blume aus an, für den die Farben berechnet werden.
- θ gibt den Winkel des Punktes an.

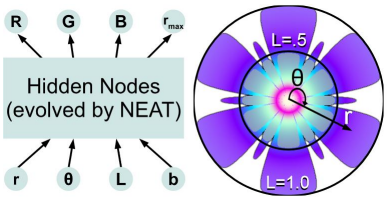

Abbildung 6: Petalz CPPN und Blume

- L gibt die Schicht an, auf der der Punkt, für den die Farbe berechnet wird, sich befindet, sodass ein CPPN für mehrschichtige Blumen verwendet werden kann.

Ausgabe:

- R, G und B sind die einzelnen Komponenten der Farbe des Punktes.
- r_{max} wird beim ersten Durchgang benötigt, um die maximalen Radien der einzelnen Winkel zu berechnen.

Bevor die Farben aller Punkte berechnet werden, wird das CPPN für jeden Winkel mit dem Radius 0 durchlaufen, um die maximalen Radien der Winkel zu bestimmen. r_{max} wird dabei ausgegeben.
Danach werden im 2. Durchgang alle Winkel mit allen Radien von 0 bis zum maximal berechneten Radius r_{max} für den jeweiligen Winkel und die gewünschte Schicht L in das CPPN eingegeben. Die Ausgabe von R, G und B bestimmt die Farbe für den angefragten Punkt. Statt dem Winkel wird jedoch tatsächlich $\sin(P * \theta)$ eingegeben, um eine Kreisform zu garantieren. P ist dabei die maximale Anzahl der Blütenblätter. [Ashlock, 2015, Seite 165ff]

5 Fazit und Ausblick

PCG wurde in ihren Anfängen vor allem wegen der eingeschränkten Rechenressourcen benutzt. Mittlerweile stellen diese jedoch keine Einschränkung mehr dar, sodass sogar noch mehr Ressourcen für PCG aufgewandt werden können.
Während Hersteller aus der Indie-Branche PCG vor allem für sich entdecken, weil sie nicht genug Entwickler und Designer haben, um alle ihre Inhalte manuell zu erstellen, hat das bei AAA-Herstellern andere Gründe. Diese sehen vor allem in der automatischen Erstellung von Inhalten eine Möglichkeit Kosten zu sparen oder weiteren Umsatz durch den Verkauf zusätzlicher Inhalte zu erwirtschaften.

Durch die Anwendung von KI in PCG können nicht nur neue, zufällige Inhalte generiert, sondern diese auch in eine für den Spieler ansprechende Richtung

gelenkt werden. PCG mithilfe von KI ist jedoch noch neu und muss weiter erforscht werden. Der suchbasierte Ansatz und die CPPN stellen da bereits einen guten Anfang dar und können vielfältig eingesetzt werden. Zu erwähnen ist hier vor allem, dass [Shaker, 2015] vielfältige Ideen bietet und den aktuellen Stand der Forschung umfassend darstellt.
Trotz des immer größeren Interesses an PCG gibt es auch Inhalte, bei denen stark gezweifelt wird, ob diese erstellt werden können. Ein Beispiel sind komplexe 3D-Welten. Komplex meint hier, dass diese nicht nur aus flachem Land und Bergen bestehen, sondern auch aus Formen wie Brücken und Bögen. Das Problem dabei ist, dass die Spielbarkeit der Welt garantiert werden muss und keine Punkte existieren dürfen, die das Weiterspielen verhindern. Die Konnektivität der Welt muss bei der Generierung überprüft werden, was ohne weitere Einschränkungen nicht möglich ist. [Doull, 2008, Abschnitt „Runtime random level generation"]
Das 2015 erscheinende ‚No Man's Sky' soll eine dynamische Generierung von 3D-Welten und Universen zur Laufzeit vornehmen. Ob die 3D-Welten in dem Sinne komplex sind, lässt sich erst beurteilen, sobald das Spiel erschienen ist, da der Hersteller nur wenig Material veröffentlicht. [IGN, 2015, Ab Minute 1:10] zeigt bisher die umfassendsten Einblicke, anhand denen sich dies nicht beurteilen lässt.

PCG bietet in Kombination mit KI viel Potenzial. Daher wäre es denkbar, dass bald nicht nur die Generierung der Spieleinhalte, sondern auch der Metadaten, wie Spielregeln und Geschichte erfolgt, sodass ganze Spiele-Generatoren existieren, die mit jedem Durchgang ein neues Spiel ausgeben.

Quellenverzeichnis

[Ashlock, 2015] Ashlock, Dan und Risi, S. u. T. J. (2015). Representations for search-based methods. In Ashlock, Dan und Risi, S. u. T. J., editor, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. Online erhältlich unter http://pcgbook.com/wp-content/uploads/chapter09.pdf, abgerufen am 28. Mai 2015.

[Dormans, 2011] Dormans, J. (2011). Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. Paper. Online erhältlich unter http://www.jorisdormans.nl/pdf/dormans2010_AdventuresInLevelDesign.pdf, abgerufen am 18. Juni 2015.

[Doull, 2008] Doull, A. (2008). The death of the level designer: Procedural content generation in games - part one. Blog. Online erhältlich unter http://roguelikedeveloper.blogspot.de/2008/01/death-of-level-designer-procedural.html, abgerufen am 28. März 2015.

[Gamespot, 2013] Gamespot (2013). Gamespot reviews - spelunky! Video. Online erhältlich unter https://www.youtube.com/watch?v=UyMCo7Kv9k4, abgerufen am 23. Juni 2015.

[IGN, 2015] IGN (2015). No man's sky: 18 minute gameplay demo - ign first. Video. Online erhältlich unter https://www.youtube.com/watch?v=CLcjvIQJns0, abgerufen am 08. Juli 2015.

[Schwab, 2008] Schwab, B. (2008). *AI Game Engine Programming*. Course Technology.

[Shaker, 2015] Shaker, Noor und Togelius, J. u. N. M. J. (2015). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. Online erhältlich unter http://pcgbook.com, abgerufen am 01. Mai 2015.

[SplatterCatGaming, 2013] SplatterCatGaming (2013). Splattercat's indie shorts - galactic arms race. Video. Online erhältlich unter https://www.youtube.com/watch?v=MKJTu51G6bE, abgerufen am 06. Juni 2015.

[Togelius, 2015a] Togelius, Julian und Smith, G. (2015a). The Power and Peril of PCG. Vortrag. Online erhältlich unter http://www.gdcvault.com/play/1022134/Making-Things-Up-The-Power, abgerufen am 18. Juni 2015.

[Togelius, 2015b] Togelius, Julian und Shaker, N. (2015b). The search-based approach. In Shaker, Noor und Togelius, J., editor, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. Online erhältlich unter http://pcgbook.com/wp-content/uploads/chapter02.pdf, abgerufen am 20. Mai 2015.

[Togelius, 2015c] Togelius, Julian und Shaker, N. u. N. M. J. (2015c). Introduction. In Shaker, Noor und Togelius, J. u. N. M. J., editor, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. Online erhältlich unter http://pcgbook.com/wp-content/uploads/chapter01.pdf, abgerufen am 01. Mai 2015.

[Wiki, 2013] Wiki (2013). PCG Wiki. Website. Online erhältlich unter http://pcg.wikidot.com, abgerufen am 06. Juni 2015.

[Yu, 2011] Yu, D. (2011). The full spelunky on spelunky. Website. Online erhältlich unter http://makegames.tumblr.com/post/4061040007/the-full-spelunky-on-spelunky, abgerufen am 06. Juni 2015.

Abbildungsverzeichnis