Procedural Content Generation in Games via Machine Learning

In vielen Spielen kommen bereits prozedural generierte Spielinhalte zum Einsatz. Dies können aneinandergereihte Levelsegmente oder auch sich wiederholende Texturen sein, die mithilfe von Algorithmen beschrieben werden. Dank dem anhaltenden Forschungsfokus im Bereich künstlicher Intelligenz stehen stetig wachsende Möglichkeiten unter der Verwendung neuronaler Netze (NNs) zur Verfügung. Mit diesen kommt auch der Wunsch diese zur Generierung von Spielinhalten zu nutzen.

Doch warum nutzt man Procedural Content Generation (PCG) überhaupt?

Nach Summerville et al. [6] sprechen aus Sicht eines Spieleentwicklers mehrere Faktoren für eine Verwendung von PCG. So entsteht durch die Verwendung generierter Inhalte ein Wiederspielwert und dadurch eine Bindung des Spielers an das Produkt. Außerdem reduzieren sich Entwicklungskosten und -aufwand für weitere Inhalte. Zusätzlich verringert sich deren Speicherbedarf aufgrund der Wiederverwertung vorhandener Elemente, wie beispielsweise sich wiederholender Kartensegmente. Zu guter Letzt kann der Einsatz von PCG auch einen rein ästhetischen Hintergrund haben.

Ein weiterer Aspekt liegt laut Summerville et al. im Bereich der Spieleforschung. Auf der einen Seite möchte man hier verschiedene Arten von Spielerlebnissen erforschen, um beispielsweise herauszufinden, wie ein Spiel vom Spieler angenommen wird, dessen Inhalte sich ihm anpassen. Auf der anderen bedient PCG das Forschungsfeld der computergestützten Kreativität, bei dem es u. a. darum geht Kreativität zu simulieren. Hinzu kommt, dass die für PCG notwendigen formalen Modelle von spezifischen Inhalten für ein allgemein verbessertes Verständnis des Spieldesigns sorgen.

Was bringt PCG via Machine Learning (PCGML)?

Bei der PCGML steht ebenfalls der Wunsch zur Kreation von Inhalten durch die Verwendung von Algorithmen im Vordergrund. Hierbei sollen die neuronalen Netze Modelle aus vorhandenen Spielelementen erlernen, mit deren Hilfe sie später gänzlich neue Inhalte schaffen können. Nach Summerville et al. wird der so erzielte Output eines trainierten NNs selbst als Spielinhalt gesehen, was für (searched based) PCG nicht zutrifft.

Da es sich bei der Erschaffung von Spielinhalten stets um einen kreativen Prozess handelt, steht hier ebenfalls die Zusammenarbeit zwischen Mensch und Maschine im Fokus.

Betrachtet man im Zusammenhang von PCGML für Videospiele die klassische Deep Learning Forschung stellt man gewisse Synergien fest. Aus Sicht der Spieleforschung verstärkt die Deep Learning Forschung die Kapazitäten in der Content Generierung. Wiederum aus Sicht der Deep Learning Forschung stellen Spiele ein herausforderndes Problem an diesen Bereich dar. Liu et al. [3] sagt hierzu:

„[…] deep learning may serve as a content generator, as a content evaluator, as a gameplay outcome predictor, as a driver of search, and as a pattern recognizer for repair and style transfer."

Abgeleitet aus dieser Aussage erkennt man eine vielfältige Anwendungsmöglichkeit für maschinelles Lernen im Bereich der Spieleentwicklung.

Wie ist der Stand der Forschung?

Hierzu haben Liu et al. [3] einen guten Überblick über in der Forschung verwendete Trainingsmethoden gegeben, welcher in diesem Abschnitt zusammengefasst wird.

Für die Generierung unterschiedlicher Inhalte wie Level, Texturen oder Sound Effekte können unterschiedliche neurale Architekturen aufgebaut werden. Ein wesentlicher Unterschied dabei ist die verwendete Trainingsmethode, mit der das neuronale Netz die Modelle erlernt. In der Arbeit von Liu et al. werden die folgenden näher betrachtet:

Supervised Learning (SL)
Überwachtes Erlernen eines Modells anhand von bekannten Trainingsdaten. Die Datensätze sind oft sehr groß und vorab mit großem Aufwand händisch selektiert bzw. kategorisiert, daher „weiß" das neuronale Netz, ob eine Entscheidung korrekt war oder nicht.

Unsupervised Learning (USL)
Unüberwachtes Erlernen eines Modells anhand von Trainingsdaten. Im Gegensatz zum SL erlernt das neuronale Netz dabei selbst die Kategorien anstatt diese vorgegeben zu bekommen.

Reinforcement Learning (RL)
Erlernen einer Strategie, zur Maximierung der Belohnung. Das neuronale Netz kann hier bestimmte Aktionen ausführen und lernt anhand der Höhe der Belohnung die beste Aktion für bestimmte Situationen anzustoßen.

Adversarial Learning (AL)
Training durch „rival model" zur stetigen Verbesserung des Ergebnisses. Hier werden zwei neuronale Netze gegeneinander ausgespielt und so stetig verbessert, beispielsweise Kunstfälscher und -kenner.

Evolutionary Computation (EC)
Training durch an die Biologie angelehnte Methodik. Dabei werden anhand von definierten Parametern Varianten eines Objekts geschaffen, deren individuelle „Überlebensfähigkeit" bewertet wird. Fällt diese größer aus, hat das Individuum eine höhere Chance seine Parameter weiterzugeben. Auch Mutationen werden simuliert.

In der Analyse von insgesamt 72 Arbeiten wurde dabei eine Verteilung der genannten Trainings Methoden wie in Abbildung 1 dargestellt, beobachtet.

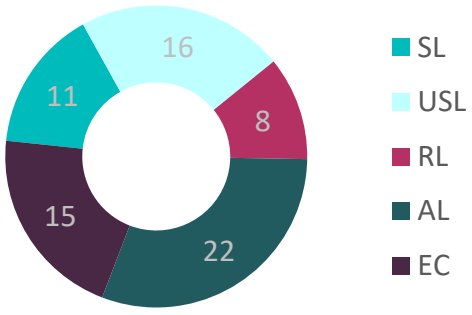

Abbildung 1: Verteilung der verwendeten Trainingsmethoden. Abbildung basierend auf Daten nach Liu et al. [3]

Hier werden schon erste Tendenzen sichtbar. So wird RL eher selten verwendet,

um Spielinhalte zu generieren. Ein Grund hierfür ist, dass es schwer ist die Belohnung festzulegen, die von der Methode gefordert wird. Wie bewertet man z. B. die Qualität eines generierten Levels?

Auch SL wird aufgrund des hohen Aufwands bei der Aufbereitung der Trainingsdaten seltener verwendet. Je nachdem was man generieren möchte sind die Daten bei Spielen auch schwer aufzubereiten (z. B. 3D-Level) oder es sind nur wenige Beispiele vorhanden (Feuer-Effekte). In der Praxis braucht SL (ebenso USL) oft Millionen Datensätze, um z. B. effektiv zwischen Bildern von Katzen und Hunden unterscheiden zu können.

Ebenfalls interessant sind die in den Arbeiten generierten Inhaltstypen welche in Abbildung 2 dargestellt werden. Die Kategorie „Other“ beinhaltet durch eine Zusammenfassung u. a. auch generierte Karten und Decks. „Game Art“ beinhaltet sowohl Charaktere als auch Gesichter und Texturen.

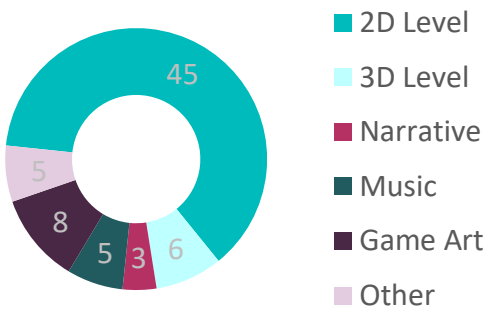

Abbildung 2: Verteilung der Arbeiten auf Content Types. Abbildung basierend auf Daten nach Liu et al. [3], einzelne Kategorien zusammengefasst.

Auf dieser Basis lässt sich feststellen, dass 62,50 % der Arbeiten auf der Generierung von 2D-Leveln basieren. Bezieht man die 3D-Level mit ein, handeln ~70 % der Arbeiten allgemein von der PCGML gestützten Levelbildung. Dabei verteilen sich die generierten 2D-Level auf alle Trainings Methoden, während 3D-Level nur mit AL und EC erforscht wurden.

An was genau wird geforscht?

Um diese Frage zu beantworten, wird im Folgenden ein genauerer Blick auf zwei Arbeiten geworfen.

PCG via Reinforcement Learning (PCGRL)

Ein Einsatzgebiet von RL ist das Erlernen des Spiels an sich, um diesen selbstständig durchlaufen zu können. Hier kann die Belohnung für das neuronale Netz beispielsweise direkt an die Punktzahl geknüpft werden. Zudem stehen dem KI-Agenten die gleichen definierten Aktionen wie dem Spieler zu.

In der Arbeit von Khalifa et al. [2] wird RL hingegen zur Generierung verschiedener 2D-Level verwendet. Laut den Autoren liegt dabei der größte konzeptionelle Unterschied zu anderen dargestellten Lernmethoden in der Betrachtung des „content generator space“ anstelle des „content space“ sowie in der Ansicht die Generierung von Inhalten als iteratives Verbesserungsproblem zu betrachten. Mit anderen Worten wird, anstatt den Inhalt selbst zu betrachten, dessen zugrundeliegender Generator erlernt.

Zudem findet bei dieser Lernmethode im Vergleich zur Gruppe der „Search Based Methods“ eine Verlagerung des Zeitaufwands von der Berechnung hin zum Training statt. Weiterhin sind im Vergleich zu SL keine Trainingsdaten notwendig.

Khalifa et al. [2] umgehen dabei die bereits vorab erwähnte Frage nach der Messbarkeit der Qualität eines Levels, indem Sie auf der Ebene des Levelgenerators auf die Einhaltung definierter Richtlinien prüfen. Diese „Policies“ können aufeinander aufbauen und ermöglichen dadurch eine leichtere Zusammenarbeit mit Designern.

Wie sehen die Policies und deren Ergebnisse aus?

Es wurden in der Arbeit drei angewandte Probleme behandelt:

Binary
Modifizierung eines 2D-Levels mit „festen“ und „leeren“ Flächen, mit folgenden Regeln:

1. Längster kürzester Weg zwischen jeglichem Punktpaar wird um *x* (20)[1] erhöht
2. Leere Flächen müssen miteinander verbunden sein

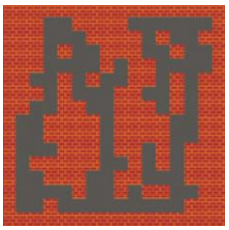

Abbildung 3: Mögliche Lösung des Binary Problems. [2]

Zelda
Modifizierung eines 2D-Levels anhand der Regeln eines „Zelda“ Dungeons:

1. Je 1 Spieler, Tür und Schlüssel
2. Spieler muss Schlüssel und Tür in *x* (16) Schritten erreichen können
3. Gegner „spawnen“ mindestens *x* (3) Tiles entfernt vom Spieler

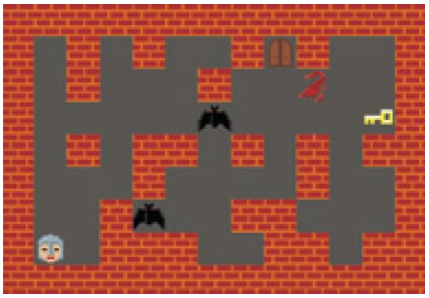

Abbildung 4: Mögliche Lösung des Zelda-Problems. [2]

Sokoban
Modifizierung eines 2D Levels anhand der Regeln des japanischen Puzzle Games „Sokoban“:

1. 1 Spieler
2. Gleiche Zahl Kisten und Ziele, welche durch den Spieler erreichbar sein müssen
3. Puzzle muss in *x* (18) Zügen lösbar sein[2]

Abbildung 5: Mögliche Lösung des Sokoban-Problems. [2]

Zusätzlich zu den Richtlinien verwendete das Team verschiedene Repräsentationen, um den Agenten das 2D-Level modifizieren zu lassen:

Narrow
In jedem Iterationsschritt erhält der Agent eine Koordinate zugewiesen, deren Zustand er modifizieren darf.

Turtle
In jedem Iterationsschritt bewegt sich der Agent ein Feld weiter (nicht diagonal) und darf Felder auf seinem Weg modifizieren.

Wide
In jedem Iterationsschritt hat der Agent die volle Kontrolle über das Feld und darf frei wählen, wo er Modifikationen vornimmt.

In Abbildung 6 wird eine beispielhafte Lösungsreihe für das Binary-Problem in verschiedenen Repräsentationen anhand

[1] Zahl in Klammern entspricht den in der Arbeit von Khalifa et al. Verwendeten Zahl.

[2] Sicherstellung durch Baumsuche (BFS & A*, 5000 Nodes)

einer gemeinsamen Ausganglage „Initial“ dargestellt.

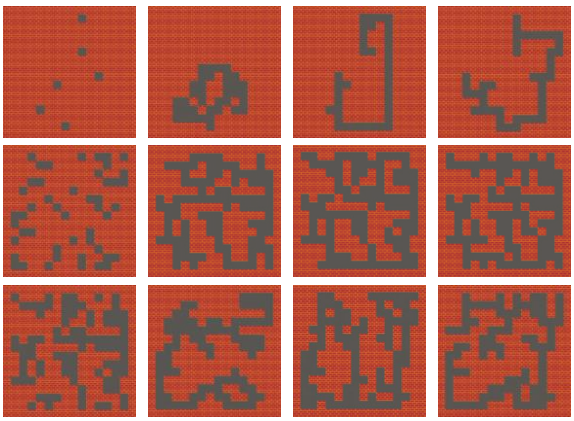

Abbildung 6: Mögliche Lösungen des Binary-Problems mit unterschiedlichen Repräsentationen. „Initial“ gibt die Ausgangslage der Reihe an. [2]

NeuroEvolution of Augmenting Topologies (NEAT)

Nach dem ersten Beispiel für die Verwendung von RL zeigt der folgende Abschnitt die Forschung mit EC am Beispiel der 2002 von Stanley et al. [5] erfundenen NEAT-Methode auf.

Papavasileiou et al. [4] fassen in ihrer Arbeit die Geschichte hinter dieser Methode gut zusammen. So wurden genetische Algorithmen (GAs) zu Beginn dazu verwendet, Artifical Neural Networks (ANNs) mit fester Topologie zu optimieren. Da die Topologie eines neuronalen Netzes aber einen signifikanten Einfluss auf die Performance aufweist, entwickelten sich sogenannte Topological and Weight Evolving ANNs (TWEANNs). Bei dieser Methode wurde die Topologie und die Gewichte zwischen den Neuronen mit dem Training modifiziert.

Im Jahr 2002 wurde mit NEAT von Kenneth O. Stanley und Risto Miikulainen eine neue TWEANN Methode vorgestellt, die für bekannte Probleme der NeuroEvolution (NE)[3] Lösungen bot. Davon sind die wichtigsten:

1. NEAT erleichtert die Kreuzung von Individuen unterschiedlicher Länge.
2. NEAT entwickelt durch das Hinzufügen von Strukturen selbst Netzwerke.
3. NEAT schützt strukturelle Innovationen des NN durch die Einteilung in unterschiedliche Spezies.

Papavasileiou et al. [4] identifizierten 60 Algorithmen, die auf NEAT basieren und bestimmte Qualitätsmerkmale aufweisen. Die Methode findet in vielen Bereichen Anwendungsfälle, so auch in der Spieleforschung. Insgesamt neun dieser Nachfolger wurden entweder an Spielen angewandt oder für die Verwendung mit Spielen entwickelt.

content-generating NEAT (cgNEAT)

Im Besonderen sticht hier der für die Generierung von Spielinhalten in Echtzeit entwickelte cgNEAT hervor. Dieser wurde 2009 von Hastings et al. [1] vorgestellt. Auch Kenneth O. Stanley war dabei einer der Autoren.

Beim cgNEAT werden dabei die Präferenzen des Spielers in die Generierung von neuen Inhalten eingebaut. Hastings et al. zeigen diese Funktionsweise am Beispiel des dafür entwickelten Spieles „Galactic Arms Race“. Bei diesem 2,5-D Multiplayer Space Shooter nehmen die Spieler die Rolle eines Piloten ein und begegnen ihren Gegnern

[3] NE ist dabei als Lernmethode zu verstehen, die evolutionäre Algorithmen (EAs) verwendet, um ANNs zu optimieren.

dabei mit den Partikelwaffen ihres Raumschiffs.

Das neuronale Netz, welches durch cgNEAT modifiziert wird, evolviert dabei genau diese Partikelwaffen. So ändert sich beispielsweise die Form, Streuung, Farbe oder Geschwindigkeit der Projektile. Dabei ist die Nutzung durch die gesamte Spielerschafft ausschlaggebend für die Wahrscheinlichkeit der iterativen Weiterentwicklung einer bestimmten Waffe. In Abbildung 7 sind ein paar Beispiele dargestellt.

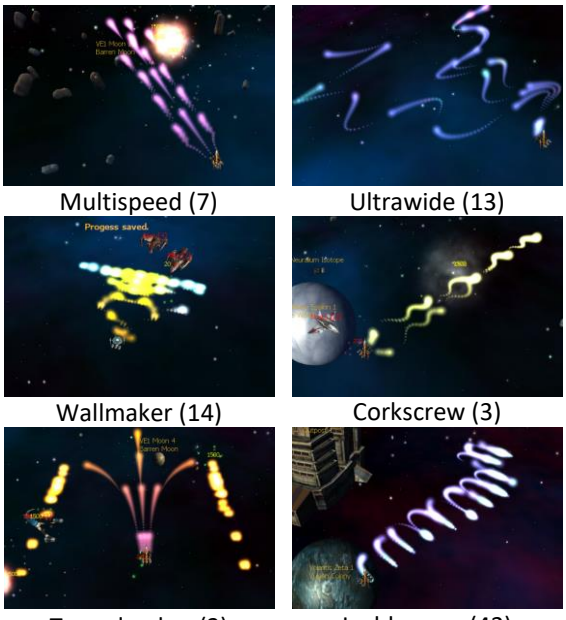

Abbildung 7: Mit cgNEAT evolvierte Partikelwaffen aus GAR. Die Zahl in Klammern gibt die Anzahl der benötigten Generation an. [1]

Wie entwickelt sich PCGML weiter?

Die bisherige Forschung ist zu unterschiedlichen Erkenntnissen gekommen. So stellen Liu et al. [3] fest, dass in der aktuellen Forschung Plattformer und „dungeon-like“ Spiele wie Zelda oder Super Mario überrepräsentiert sind.

Eine weitere wichtige Erkenntnis ihrer Arbeit lautet, dass eine gewünschte Zielqualität oder -eigenschaft schwer über reine PCG zu erreichen ist. Vielmehr müssten Prozesse die Zusammenarbeit von Mensch und Maschine beachten. Hier wird der Begriff „Mixed-initiative PCG“ angeführt, bei dem es darum geht, den Designer oder Spieler den Prozess steuern zu lassen.

Für die Zukunft stehen verschiedene Arbeits- und Forschungsfelder bereit. So sehen sowohl Liu et al. [3] als auch Summerville et al. [6] ein Problem mit der limitierten Anzahl an Trainingsdaten in Videospielen. Liu et al. betrachten es als wünschenswert, dass ein Generator im Training nur mit einer Hand voll handgemachter Inhalte (z. B. Items, Level oder Charaktere) auskommt. Für Spiele, die sich noch in der Entwicklung befinden, könnten bereits veröffentlichte Spiele desselben Genres als Datensatz dienen.

Als Folge der angesprochenen Überrepräsentation bei der Generierung von 2D-Leveln, können sich zukünftige Arbeiten mit der Adaption der Erkenntnisse auf weitere Inhalte, wie beispielsweise Rulesets, Events, Charaktere, Fähigkeiten oder vermehrt mit 3D-Leveln beschäftigen. Hier können auch aktuelle Entwicklungen aus dem Bereich Deep Learning mit eingewoben werden.

Die in Hastings et al. [1] bereits durchgeführte Personalisierung in Echtzeit an das Verhalten und die Bedürfnisse des Spielers können für künftige Arbeiten ebenfalls eine Basis bilden.

Allgemein kann an der Orchestrierung des gemeinsamen Entwicklungsprozesses zwischen Mensch und Maschine gearbeitet werden, um so künftig standardisierte und etablierte Prozesse und Abläufe zu erhalten.

Nach Liu et al. [3] ist ein nicht ausgeschöpftes Feld ebenfalls der Style Transfer, das Breeding und Blending zur Generierung neuer Spielinhalte.

Was lässt sich abschließend sagen?

Rund um die prozedurale Generierung von Spielinhalten existieren viele interessante Themen und Forschungsfelder mit zahlreichen Facetten und Ausprägungen. Allgemein ist die KI-Forschung ein sehr relevanter Bereich in der Forschung. Hierbei können auch Videospiele einen wichtigen Teil zur Forschungslandschaft beitragen und z. B. als komplexe Problemstellung in der Deep Learning Forschung behilflich sein.

Aktuell hält sich der reale Nutzen von PCGML für die Spieleindustrie aber scheinbar in Grenzen. Langfristig kann PCGML aber einen Mehrwert für Designer und Entwickler während der Entwicklung neuer Spiele bieten und den Kreativprozess positiv unterstützen.

Quellenverzeichnis

[1] E.J. Hastings, R.K. Guha, and K.O. Stanley. 2009. Automatic Content Generation in the Galactic Arms Race Video Game. IEEE Transactions on Computational Intelligence and AI in Games1, 4 (Dec. 2009), 245–263. https://doi.org/10.1109/tciaig.2009.2038365

[2] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. 2020. PCGRL: Procedural Content Generation via Reinforcement Learning. CoRR abs/2001.09212 (2020). arXiv:2001.09212 https://arxiv.org/abs/2001.09212

[3] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, and Julian Togelius. 2020. Deep learning for procedural content generation. Neural Computing and Applications 33, 1 (Oct. 2020), 19–37. https://doi.org/10.1007/s00521-020-05383-8

[4] Evgenia Papavasileiou, Jan Cornelis, and Bart Jansen. 2021. A Systematic Literature Review of the Successors of “NeuroEvolution of Augmenting Topologies”. Evolutionary Computation 29, 1 (March 2021), 1–73. https://doi.org/10.1162/evco_a_00282

[5] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies.Evolutionary Computation 10, 2 (June 2002), 99–127. https://doi.org/10.1162/106365602320169811

[6] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2017. Procedural Content Generation via Machine Learning (PCGML). CoRR abs/1702.00539 (2017). arXiv:1702.00539 http://arxiv.org/abs/1702.00539