SoNNic

Neuroevolution in Sonic the Hedgehog

Kyrill Hux

mng rämibühl

Betreuung: Lukas Fässler

Einleitung

SoNNic ist ein Computerprogramm, das mithilfe von künstlichen neuronalen Netzen selbstständig lernt, den Spieleklassiker „*Sonic the Hedgehog*“, der 1991 auf der Spielekonsole „*Sega Genesis*“ erschien, zu spielen und das erste Level erfolgreich abzuschliessen.
Sonic the Hedgehog eignet sich besonders gut für das gesetzte Ziel, da das Spiel eine lineare Spielweise besitzt, wo der Charakter von links nach rechts zum Ziel laufen und dabei verschiedensten Gegnern und Hindernissen ausweichen muss, um das Spiel zu meistern. Besonders das erste Level zeichnet sich dadurch aus, dass es ohne die Richtung zu wechseln durchlaufen werden kann (siehe Abbildung 1).
Da *Sonic the Hedgehog* heutzutage als Klassiker gilt, existiert eine grosse Community um dieses Spiel, die bis heute aktiv zahlreiche Modifikationen und Erweiterungen für das Spiel entwickelt. Aus diesem Grund ist eine umfangreiche Dokumentation verschiedenster Spielmechaniken- und Techniken verfügbar, die Informationsbeschaffung aus dem Arbeitsspeicher, wie sie für SoNNics Zwecke notwendig ist, massiv erleichtert. Andere Spiele, wie „*Excitebike*“ aus dem Jahre 1984 (Abbildung 2), erfüllen diese Vorgaben nicht.

Abbildung 1: Das Layout des ersten Levels in *Sonic the Hedgehog*

Abbildung 2: „*Excitebike*“, ein populäres NES-Spiel, das im Jahre 1984 veröffentlicht wurde.

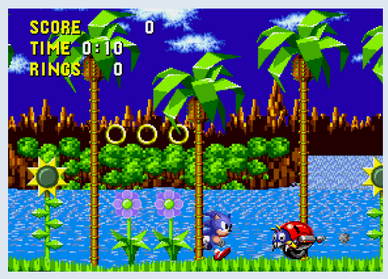


Abbildung 3: Der 1991 auf der Spielekonsole „*Sega Genesis*“ veröffentlichte Titel „*Sonic the Hedgehog*“

Neuronale Netze & NEAT

Um das Ziel umzusetzen, nutzt SoNNic sogenannte „künstliche neuronale Netze“, kurz KNNs. Diese bestehen aus vielen künstlichen Neuronen, die in mehreren Schichten hintereinander geschaltet werden (siehe Abbildung 5). Dabei gibt es prinzipiell 3 Schichtarten: Eine Schicht vorher festgelegter *Inputs*, wo Daten hineingegeben werden (grün), beliebig viele Schichten Neuronen („*hidden nodes*“, blau) und eine Schicht vorher festgelegter *Outputs*, wo die Ergebnisse abgerufen werden können (gelb). Die künstlichen Neuronen, aus denen die Netzwerke bestehen, nehmen alle hereinkommenden Signale $i_1 \dots i_n$ und multiplizieren sie mit den entsprechenden Gewichtungen $w_1 \dots w_n$. Auf die Summe dieser Werte wird dann eine Aktivierungsfunktion angewendet und das Ergebnis über den Output weitergeleitet (Abbildung 4).

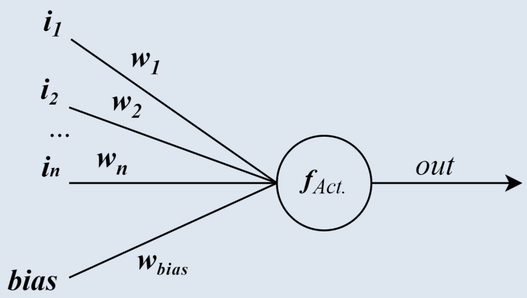


Abbildung 4: Schema eines künstlichen Neurons

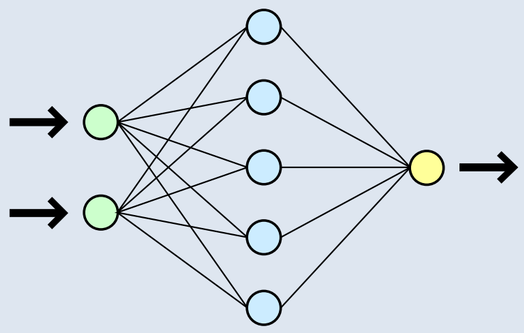


Abbildung 5: Schema eines KNNs

KNNs alleine haben kaum Nutzen - sie müssen ‚trainiert‘ werden, um sinnvolle Werte zu produzieren. Dies ist ein Vorgang, bei dem die einzelnen Gewichtungen der künstlichen Neuronen angepasst werden. Um seine Netzwerke zu trainieren, nutzt SoNNic einen Algorithmus, der unter dem Akronym „*NEAT*“ bekannt ist. *NEAT* ist besonders, da hier eine Evolution emuliert wird, wie sie auch in der Natur vorhanden ist. Dazu wird eine Menge KNNs generiert, von welchen jedes ein eigenes Genom hat. Dieses Genom enthält einen genauen Bauplan, mit dem das zugehörige KNN rekonstruiert werden kann. Jedes KNN wird dann der Aufgabe ausgesetzt, die es erfüllen soll und je nach dem, wie nahe es dem Ziel gekommen ist, besser oder schlechter bewertet. Diese Bewertung wird *Fitness* genannt. Wenn alle KNNs geprüft wurden, wird aus diesen eine neue Generation erstellt, wo die neu entstandenen KNNs zufällige Mutationen aufweisen können. Die Netzwerke, die besser abgeschnitten hatten, dürfen mehr Nachkommen zeugen als solche, die schlechter abschnitten. Die neue Generation wird anschliessend auf die gleiche, oben beschriebene Art und Weise geprüft und der Kreislauf so fortgesetzt.

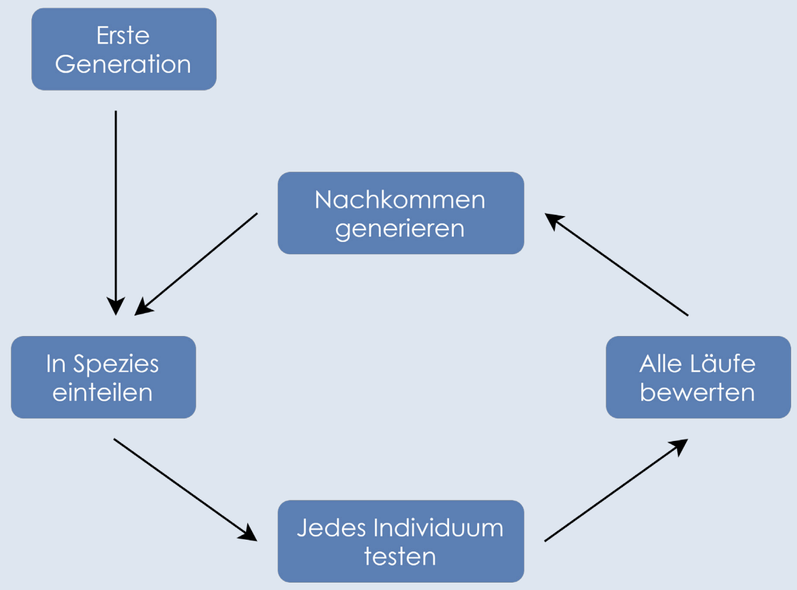
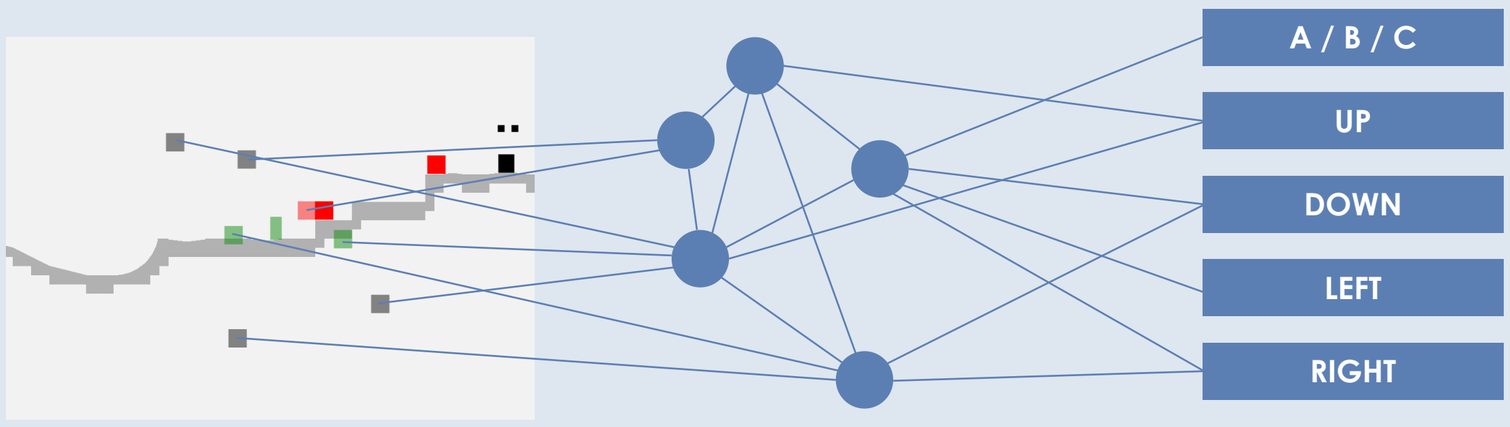


Abbildung 6: Schema eines *NEAT*-Zykluses

Funktionsweise



Inputs

SoNNic legt entgegen den Konventionen keine *Inputs* von Anfang an fest. Stattdessen werden neue *Inputs* als zufällige Mutationen im Laufe der Evolution hinzugefügt. Diese *Inputs* erhalten dann jeweils eine zufällig gewählte Position relativ zur Spielfigur. Die Werte für die *Inputs* werden in jedem Frame des Spielgeschehens aus einer von SoNNic generierten vereinfachten Spielansicht entnommen, wie sie über diesem Abschnitt dargestellt ist. Die *Inputs* werden dort als transparente Quadrate angezeigt. Je nach dem, was diese berühren, ändern sie ihre Farbe und damit auch den Input-Wert, der an das KNN weitergegeben wird:

- Grün bei festen Gegenständen
- Rot bei gefährlichen Objekten
- Grau bei keiner Berührung

Auf diese Art und Weise kann SoNNic auf die Umgebung reagieren und ohne grössere Schwierigkeiten zwischen verschiedenen Hindernisstypen unterscheiden.

Abbildung 7: Spielansicht des oben dargestellten Frames

Hidden Nodes

Hidden nodes arbeiten in SoNNics KNNs genau so, wie man es von ihnen erwarten würde. Es gibt beliebig viele Schichten, die durch Mutationen entstehen oder verloren gehen können. Die Input-Werte werden summiert und auf die Summe wird eine Aktivierungsfunktion angewendet. In SoNNics Fall ist dies ein Sigmoid (Abbildung 8), der einen fliessenden Schwellwert simulieren soll. Oft werden bei KNNs auch Step-Funktionen (Abbildung 9) verwendet, für SoNNics Zwecke eignete sich allerdings ein Sigmoid dank den möglichen Zwischenwerten um einiges besser. Die Ergebnisse werden zur jeweils nächsten Schicht weitergeleitet und schliesslich an die *Outputs* ausgegeben.

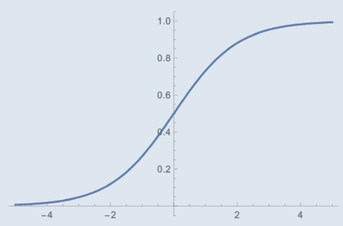


Abbildung 8: Ein Sigmoid

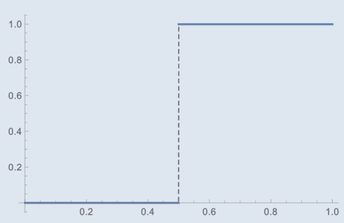


Abbildung 9: Eine Step-Funktion

Outputs

Die *Outputs* sind bei SoNNic relativ simpel definiert: Es sind die Knöpfe auf dem Controller, die zum jeweiligen Zeitpunkt gedrückt werden sollten. Das sind also das Steuerkreuz mit allen Himmelsrichtungen, das die Bewegung des Spielcharakters angibt mit jeweils einem *Output*, sowie die drei Knöpfe, die alle nur zum Springen benutzt werden können. Diese erhalten zusammen nur einen *Output*, da die Funktion bei allen die gleiche ist und mehrere *Outputs* keinen Vorteil, sogar einen Nachteil beim Lernprozess einbringen würden.

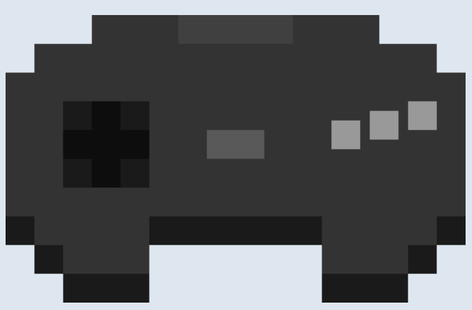


Abbildung 10: Vereinfachte Darstellung eines „*Sega Genesis*“ Controllers

Ergebnisse

Gesamtlaufzeit: 1029 Generationen: 289 Stunden ≈ 12 Tage
Aufgabe: Erfolgreich abgeschlossen

Fitness = grösste X-Koordinate eines Laufs im Level