Ökosystemsimulation

Jonas Berggren, Jacob Maxton

December 20, 2019

Abstract

In diesem Dokument erklären wir, wie wir eine numerische Simulation eines Ökosystems entwickelt haben, und was für Erkenntnisse sich daraus ziehen lassen. Die Simulation basiert auf Objektorientierter Programmierung und simuliert die Wechselwirkung zwischen Kaninchen und Fuechsen.

Contents

1 Die Simulation

1.1 Objektorientierte Programmierung und numerischen Simulationen(Jonas)

Objektorientierte Programmierung bezeichnet das Programmieren mit Hilfe so genannter Klassen und Objekte. Ein Objekt oder eine Instanz einer Klasse ist beispielsweise ein Individuum des Typs Kaninchen. In dem Beispiel waere die Klasse die Kaninchen im allgemeinen. Klassen kann man sich vorstellen wie Methodenkarten, in denen eine Reihe von Informationen gespeichert sind. Diese koennen Instanzvariablen oder Methoden sein. Instanzvariablen sind Variablen die jede Instanz traegt. Methoden sind Funktionen oder Anweisungen die jede Instanz der Klasse ausfuehren kann. Klassen haben ausserdem Hierarchien. Eine Instanz einer sogenannten Unterklasse hat automatisch alle Methoden der uubergeordneten Klasse. Im unserer Simulation sind die Klassen `Rabbit` und `Fox` Unterklassen zu `Animal`. Somit erben sie alle Methoden und variablen von `Animal`.

Bei numerischen Simulationen wird ein Szenario aus der Reellen Welt durch eine Computersimulation nachgestellt. Dabei wird das System zum Zeitpunkt t betrachtet. Auf Grundlage dessen wird der Zustand des Systems zum Zeitpunkt t + h berechnet. Dabei gilt je kleiner h desto genauer ist die Simulation. Alle aenderungen werden angewand und der Prozess wird wiederholt bis der gesamte Zeitraum Simuliert wurde den es zu betrachten gilt. Dies ist nuetzlich um die Gultigkeit von Modellen zu pruefen oder z.b. die Stabilitaet des betrachteten Systems zu testen.

1.2 Konzept der Simulation(Jonas)

In der Simulation werden Pflanzen Kaninchen und Fuechse simuliert. Jedes Tier ist entweder eine Instanz der Klasse `Fox` oder `Rabbit`, die jeweils Unterklassen der Klasse `Animal` sind. In Animal sind alle Methoden gespeichert, die fuer Kaninchen und Fuechse identisch ausgefuehrt werden, wie der Konstruktor `__init__` oder `movetargeted` zu gezielten bewegen. Methoden die fuer Fuechse und Kaninchen unterschiedlich sind, sind in den jeweiligen Unterklassen gespeichert. Diese sind z.b. findtarget zum finden aller potentiellen Ziele. Auf die unterschiedlcihen Instanzvariablen und Methoden wird aber in Kapitel 1.3 weiter eingegangen. Alle Tiere haben eine kreisförmiges Sichtfeld, was nicht die komplette Karte abdeckt. Sie haben alle ein, mit der Zeit zunehmendes, Bedürfnis sich fortzupflanzen und zu essen. Alle Bedürfnis werden bei deren Erfüllung verringert. Wird der Hunger zu stark kann das Tier verhungern und sterben. Ausserdem kann ein Tier zu jedem Zeitpunkt sterben, mit einer Wahrscheinlichkeit sterben die vom alter abhaengig ist. Kaninchen essen pflanzen und Füchse essen Kaninchen, wobei gegessen Kaninchen sterben. Kaninchen koennen vor Fuechsen fluechten die sich innerhalb des Sichtfelds befinden.

1.3 Methoden(Jonas)

1.3.1 Aussuchen eines Ziels

Dies wird durch die Methode `findtarget` geregelt, die aus der Hauptschleife aufgerufen wird. `findtarget` ist sowohl eine Methode der Klasse `Fox` als auch der Klasse `Rabbit`. Dort sind sie aber unterschiedlich definiert dar Fuechse sich anders verhalten sollen als Kaninchen. Jedes Tier speichert alle Objekte die sich innerhalb des Sichtradius `self.sens` befinden in einer Liste ab.

Anschliessend wird geprueft welches Ziel angesteuert werden soll. Dazu wird aus jeder Liste das naechste Element gesucht. Wenn ein Kaninchen ein Fuchs sieht hat die Flucht immer oberste Priorität. Der Fuchs wir anvisiert und der Bewegungsvektor wird mit -1 multipliziert. Danach wird geprueft ob Hunger oder Libodo staerker wirkt und demnach wir entschieden ob das naechste essen oder der naechste Partner anvisiert wird.

1.3.2 Bewegung

Die Bewegung der Tiere wird durch die Methoden `movetargeted` und `moverandom` definiert. Jedes Tier hat zwei Arten wie es sich fortbewegen kann. Wenn es ein bestimmtes Ziel hat, wird `movetargeted` aufgerufen, und das Tier bewegt sich entlang des Vektors von der eigenen Position zum Ziel. Dabei ist der Bewegungsvektor auf die Bewegungsgeschwindigkeit normiert. Wenn kein Ziel in Sicht ist, wird `moverandom` aufgerufen, und sie bewegen sich zufällig. Dabei wird die Methode `collision` jedes mal aufgerufen. Diese verhindert das Tiere aus der Karte raus laufen.

1.3.3 Fortpflanzung und Mutation

Haben sich zwei gefunden wird der Liste der Tiere eine neue Instanz der Klasse hinzugefügt. Bei den mutierbaren Eigenschaften wird der Durchschnitt aus den jeweiligen Werten der Eltern als Mittelwert angenommen. Der Wert des Kindes weicht weicht um x von diesen Durchschnitt ab, wobei x eine zufallige Zahl zwischen u und v ist

Die Eltern speichern sich gegenseitig als ehemalige Partner ab und betrachten sich anschließen für eine festgelegte Frist nicht mehr als mögliche potentielle Partner. Ausserdem koennen sich die Eltern prinzipiell fuer eine Festgelegte Frist nicht weiter fortpflanzen.

1.3.4 Tod

Tiere koennen auf drei untertschiedlichen Arten sterben. Sie koennen verhungern, sie koennen durch die Altersabhaengige Funktion sterben oder Kaninchen koennen gefressen werden. Dies wird durch die Methoden `starve`, `die` und `eat` geregelt. Die Methoden `starve` und `die` werden bei jeder Iteration aufgerufen, und `eat` wenn ein Kaninchen gefressen wird.

Die Todeswahrscheinlichkeitsdichte wird durch eine Funktion beschrieben nach dem Muster:

$$a(e^{-t+b} \cdot c \cdot t + d) \tag{1}$$

t entspricht dem Alter fuer $t \in \mathbb{R}_{>0}$ und a bis d sind Konstanten. Durch so eine Funktion ist P fuer ein kleines t relativ gross. P erreicht im positiven Wertebereich ein Minimum und waechst dann approximativ linear an. Dies fasst Kindersterblichkeit und Alterschwaeche in ein Funktion zusammen.

Beim sterben durch gegessen werden loescht der Fuchs die gegessene Instanz aus der Liste der Tiere.

1.3.5 Andere Methoden

Hinzu kommen noch weiter Hifsmethoden wie z.b. `distance`, die den Abstand zwischen self und einem beliebigen Punkt berechnet. Diese sind jedoch fuer das allgemeine Verstaendniss unsere Abreit nicht essentiell.

1.4 Frontend (Jacob)

1.5 Tarieren der Werte(Jacob)

2 Die Analyse

3 Verbesserungsmoeglichkeiten (Jonas)

Eine Moeglichkeit das Programm zu verbessern waere ein sogenanntes Gridsystem zu implementieren. Dabei wird das Feld in mehrere Subfelder unterteilt. Die Objekte werden in einer Matrix gespeichert wobei jeder Index der Matrix eine Subfeld zugeordnet ist. Das bietet den Vorteil, dass die Tiere bei Jeder Iteration nicht mehr die Die Position aller Objekte abfragen muessen, sondern nur die der Objekte die sich im selben bzw. in einem der Felder befinden, die mit dem Sichtkreis ueberlappen. Somit kann die Laufzeit pro Iteration stark reduziert werden.

Asserdem kann eine andere, schnellere Programmiersprache verwendet werden. Fuer Umfangreiche wissenschaftliche Anwendungen kann eine erweiterte Version eines solchen Programms auf staerkeren Computern ausgefuehrt werden.

4 Anwendungen

Eine Umfangreichere Simulation dieser Art koennte nuetzlich sein um die Auswirkungen von Menschlichem Eingriff in oekosysteme wie Klimaerwaermung, Lebensraum veraendungen oder austerben einer Art abzuschaetzen. Dies koennte nuetzlich sein um Notwendigkeit und Dringlichkeit von Umwelt Massnahmem abzuschaetzen.

5 Fazit

Wir haben mit Hilfe Objektientierter Programmierung eine Vereinfachte Simulation eines OEkosysteme entwickelt indem wir uns auf die Wechselwirkung zwischen zwei Tierarten konzentriert haben. Dabei sind wir auf unterschiedliche Schwierigkeiten gestossen. Zuerst mussten wir entscheiden wie genau unsere Simulation sein soll. Anschliesend war es sehr schwierig all Willkuerrlich gewaehlten Parameter so zu beziffern, dass keine der Arten zu schnell ausstrirbt. Dabei haben wir festgestellt wie fragil auch so ein einfaches OEkosystem sein kann.