Eine Betrachtung von Valentin Bürgler und Lucas Schnüriger

1. Einleitung

1.1. Geschichte

Die erste öffentliche Version von Scratch wurde 2007 von MIT-Professor Mitchel Resnick veröffentlicht. Sie wurde in Squeak, einer Version von Smalltalk, implementiert. 2013 erschien der offizielle Release von Scratch 2.0, eine komplette Neuimplementation in Adobe Flash.

Für 2019 ist der Release von Scratch 3.0 angesetzt, wieder eine Neuimplementation – diesmal in HTML5, also mit neuesten HTML- und CSS-Markups und Javascript-Version. Scratch 3.0 verwendet hauptsächlich WebGL, Web Workers und Web Audio Javascript Libraries. Javascript wurde wegen seiner weit verbreiteten Unterstützung durch gängige Webbrowser gewählt (Internet Explorer wird explizit auch in Zukunft nicht unterstützt). WebGL wurde für seine Geschwindigkeit und die Fähigkeit, Operationen wie "touching color", auserkoren.

Release	Datum	Verfügbar auf
Public Alpha	Januar 2018	https://preview.scratch.mit.edu/
Beta	01. August 2018	https://beta.scratch.mit.edu/
Offiziell	02. Januar 2019	https://scratch.mit.edu/

Tabelle 1: Übersicht der offiziellen Release-Termine von Scratch 3.0

Scratch-Programme werden in einer browserbasierten Entwicklungsumgebung auf https://scratch.mit.edu erstellt. Eine offline-Version ist zum Download verfügbar, allerdings zum Zeitpunkt dieses Teamprojektes noch nicht für Version 3.0.

1.2. Vision

> "*We wanted to develop an approach to programming that would appeal to people who hadn't previously imagined themselves as programmers. We wanted to make it easy for everyone, of all ages, backgrounds, and interests, to program their own interactive stories, games, animations, and simulations, and share their creations with one another.*" - **Resnick, Mitchel, et al.** "Scratch: programming for all.", in: *Communications of the ACM* 52.11 (2009): 60-67.

Die Programmiersprache Scratch soll die Grundkonzepte der Programmierung für Neueinsteiger verständlich vermitteln. Die Zielgruppe sind primär Kinder und Jugendliche. Nach eigenen Aussagen hilft Scratch:

> "*[...] jungen Leuten, kreativ zu denken, systematisch zu schlussfolgern und miteinander zusammenzuarbeiten [...]*" - ***Lifelong Kindergarten Group:*** *Über Scratch.* URL: https://scratch.mit.edu/about [17.12.2018].

Weiter sammelt das MIT Daten über die Nutzung und betreibt Forschung darüber, wie die Benutzer mit Scratch lernen und arbeiten.

1.3. Verbreitung

Scratch wird in über 150 Ländern verwendet und ist in mehr als 40 Sprachen verfügbar. In Schulen ist es nicht selten Teil des Informatikunterrichts geworden. In den meist nach Themenbereich gruppierten, sogenannten "Scratch Studios" (vor 2.0 noch "Galleries") auf der offiziellen Seite (https://scratch.mit.edu/) können Benutzer ihre Projekte veröffentlichen. Nach dem Veröffentlichen ist es jedermann gestattet, einen sogenannten "Remix" des Projektes zu machen. Sämtliche in "Scratch Studios" veröffentlichte Projekte unterliegen der **Creative Commons Attribution Share-Alike license 2.0**.

2. Die Sprache

2.1. Allgemeines

In einem Scratch Projekt agieren **Spielfiguren** (Sprites) auf einer **Bühne** (Stage). Sie können unterschiedliche Kostüme (Grafiken) anziehen und Geräusche abspielen. Alle Bestandteile können selber entworfen und zur Laufzeit verändert werden. Die Figuren können bewegt werden, man kann sie denken, sprechen oder Geräusche machen lassen und sie können auf verschiedene Ereignisse reagieren.

Scratch ist eine **visuelle** Programmiersprache. Anweisungen werden hierbei mit grafischen Elementen statt mit reinem Text dargestellt und per Drag & Drop zusammengebaut. Auch wenn es keine Konstrukte für Klassen und Vererbung oder Prototypen gibt, besitzt Scratch **objektorientierte** Aspekte. Spielfiguren sind Objekte mit Eigenschaften (Datenkapselung) und können geklont werden um mehrere Instanzen mit gleichem Verhalten und unterschiedlichen Eigenschaften zu erhalten. Die Kommunikation zwischen verschiedenen Instanzen ist über den Austausch von Nachrichten möglich. Die einzelnen Skripte sind **imperativ**, die Bausteine bilden eine schrittweise Folge von konkreten Befehlen. Jedes Scratch-Skript startet, sobald dessen "Kopf-Bedingung" erfüllt ist, bzw. dessen Ereignis eintritt. So kann Scratch auch als **ereignisorientiert** angesehen werden. Variablen sind **dynamisch typisiert**, deren Datentyp kann sich während der Ausführung ändern. Oberflächlich wird auch nur zwischen Zeichenketten und Zahlen unterschieden. Als Datenstruktur kennt Scratch **Listen** mit dynamischer Länge und beliebigen Datentypen.

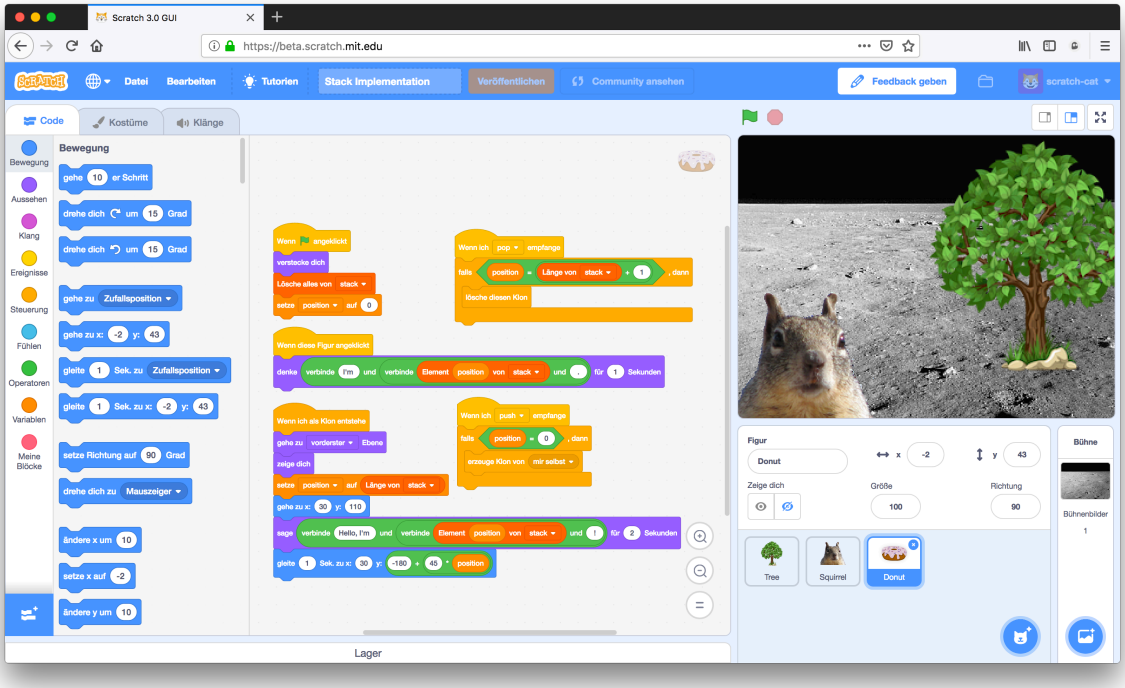

Abbildung 1: Oberfläche von Scratch 3.0

2.2. Blöcke

Scratch-Programme bestehen aus einem oder mehreren Skripten, welche aus mehreren aufeinander gestapelten Blöcken aufgebaut wird. Ein Skript besteht mindestens aus einem Kopf-Block, der das Ereignis festlegt unter dem es gestartet wird. Jedes Skript ist entweder Teil einer Figur oder der Bühne.

Blöcke sind wie farbige Puzzleteile, die sich miteinander verbinden lassen. Durch deren Form sind nur syntaktisch korrekte Kombinationen möglich, Syntaxfehler werden damit vermieden. Es gibt weit über 100 verschiedene Blöcke in den folgenden 6 Formen:

- **Hat:** Die Hut, bzw. Kopf-Blöcke, starten ein Skript als Reaktion auf ein Ereignis.
- **Stack:** Sie führen wesentliche Befehle aus und haben Anschlüsse oben und unten.
- **Reporter:** Sie stehen für Werte und geben entweder eine Zahl oder eine Zeichenkette zurück.
- **Boolean:** Spezielle Reporter, die entweder mit Wahr/1 oder Falsch/0 antworten.
- **C:** Die Klammer-Blöcke sind Steuerungsblöcke, die andere Blöcke enthalten, die entweder mehrmals oder nur unter einer Bedingung ausgeführt werden sollen. Sie sind das Pendant für While, For und If.
- **Cap:** Mit den Abschluss-Blöcken werden Skripte beendet.

Nebst der Form können die Blöcke in ihrer Farbe unterschieden werden. In Scratch 3.0 gibt es neun offizielle Blockkategorien, die jeweils eine andere Farbe haben.

Farbe	Kategorie	Bedeutung
Blau	Motion (Bewegung)	Steuern die Bewegungen von Figuren.
Violett	Looks (Aussehen)	Steuern das Aussehen von Figuren und der Bühne. Die am häufigsten verwendeten Blöcke.
Pink	Sound (Klang)	9 Blöcke um Töne abzuspielen und zu steuern.
Gelb	Events (Ereignisse)	Erkennen Ereignisse und lösen Skripte aus. Fast alle Hat Blöcke sind Event Blöcke.
Gold	Control (Steuerung)	Steuern den Kontrollfluss der Skripte mit Schleifen und Bedingungen.
Türkis	Sensing (Fühlen)	Werden verwendet um verschiedene Faktoren zu erkennen: darunter Berührungen von Farben und Benutzereingaben.
Grün	Operators (Operatoren)	Mathematische und logische Operationen.
Orange/ Rot	Data (Daten)	Variablen um Zahlen oder Zeichenketten zu speichern (orange) und Listenoperationen (rot).
Rosa	Custom Blocks (Meine Blöcke)	Vom Benutzer erstellte Hat oder Stack Blöcke. Erlaubt der Aufruf von Skripten mit Parametern. Dies ermöglicht auch Rekursion.

Tabelle 2: Übersicht der verfügbaren Blockkategorien in Scratch 3.0

Nebst den wesentlichen Kategorien gibt es Erweiterungen (Extensions) die unter anderem die Verwendung von Kamerabild, Text-to-Speech und weiterem erlaubt.

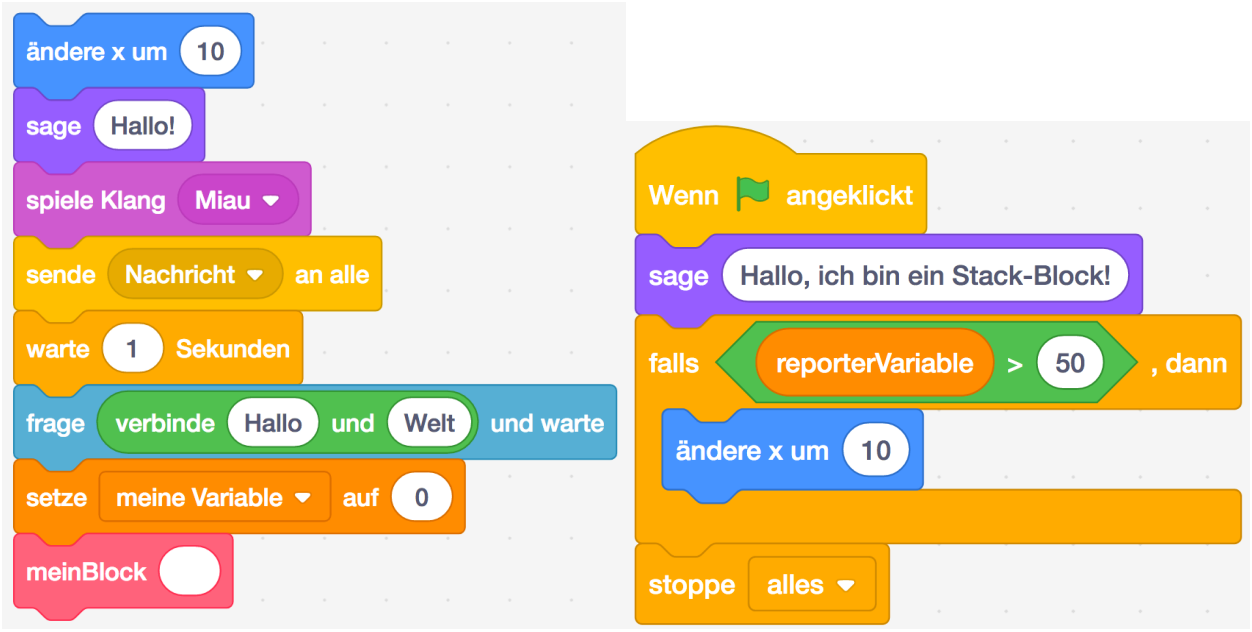

Abbildung 2: Beispielblöcke aus jeder Farbe (links) und Form (r) von Scratch 3.0

2.3. Nachrichten & Nebenläufigkeit

Zwei der Ereignis-Blöcke erlauben es Nachrichten als **Broadcast** an alle Figuren des Scratch-Programms zu senden. Eine Nachricht besteht nur aus einem einzelnen Bezeichner und enthält keine weiteren Informationen oder Daten. Bei jeder Figur kann nun der entsprechende Hat-Block hinzugefügt werden, dessen darunter stehende Skript als Reaktion auf eine erhaltene Nachricht ausgeführt wird. Über dieses Senden und Empfangen wird die Kommunikation zwischen Figuren ermöglicht. Sie können so interagieren und aufeinander reagieren.

Abbildung 3: "Push"-Nachricht wird gesendet (links) und empfangen (r)

Da beliebig viele Figuren und Skripte auf eine Nachricht aktiviert werden können, lässt sich Code auf sehr einfache Art parallel ausführen. Dies auch schon ohne wirklichen Kenntnissen von Parallelisierung und Synchronisierung. Mit "Sende an alle und warte" kann man sogar abwarten, bis alle aufgerufenen Skripte abgeschlossen sind. Zum Beispiel bei einem Dialog zwischen zwei Figuren, die so aufeinander warten können.

2.4. Klone (erlauben OOP)

Es gibt in Scratch keine Klassen oder Vererbung. Auch die Datenkapselung ist auf lokale und globale Variablen begrenzt. Dennoch kann man objektorientierte Ansätze umsetzen dank der Möglichkeit des Klonens.

Figuren werden grundsätzlich direkt, quasi als Instanzen, auf der Bühne hinzugefügt. Sie haben ihre eigenen Skripte und können private Variablen besitzen. Zur Laufzeit können existierende Figuren aber geklont werden (siehe Abbildung 3 rechts: "erzeuge Klon von mir selbst"). Dies ist quasi die Instanziierung weiterer Objekte in einem Prototype-basierten Verfahren. Daraufhin agieren diese Klone eigenständig, aber mit den selben Skripten. Ein entsprechendes Ereignis erlaubt es Klonen auf ihre Entstehung zu reagieren. Sie können sich auch selber wieder löschen.

Geklonte Figuren haben keine besonderen Bezug zueinander oder zu anderen Figuren. Auch sie können nur über Broadcasts und globalen Variablen kommunizieren.

3. Fazit

3.1. Team

Handelt es sich bei Scratch um eine "Gute" Sprache?

Ja. Kaum eine Sprache schafft es, genau das zu halten, was sie verspricht. Scratch schafft dies mit Bravour. Sie will den Einsteig in die Welt des Programmieren verständlich und intuitiv machen, unbeachtet des Vorwissens. Dies macht Scratch nach unserer Meinung hervorragend.

Pro: Es ist die Sprache, um jemanden an das Programmieren heran zu führen. Man braucht keinen Syntax zu lernen, da dieser in Form von visuellen Blöcken völlig selbsterklärend ist. Ausserdem lassen sich keine Syntaxfehler machen, weil man die Blöcke nicht falsch zusammenstecken kann. Die Blöcke ergeben letztendlich auch sehr sprechenden "Code", den man dann in seiner eigenen Muttersprache lesen kann.

Contra: Als Entwickler ist die Programmierung mit praktisch ausschliesslich Drag & Drop sehr ungewohnt und eher ein Hindernis zum schnellen Arbeiten mit der Tastatur.

Würden wir sie regelmässig einsetzten wollen?

Als Softwareentwickler: Nein, nicht produktiv. Projekte können nicht offiziell als Stand-Alone-Applikation exportiert werden, sondern sind von der Scratch-Webseite abhängig. Da alles im Rahmen von Scratch unter der Creative Commons Lizenz steht, darf man keine proprietäre Software mit geschlossenem Quelltext erstellen. Scratch könnte allerdings als Werkzeug für rasche Proof-of-Concept Prototypen hilfreich sein.

Als Lehrer: Definitiv. Die grafische Oberfläche und visuelle Darstellung von Code ermöglicht einen unkomplizierten Start in die Programmierung.

3.2. Persönliches Fazit Lucas Schnüriger

Mir war Scratch zwar durchaus ein Begriff, ich wusste aber nicht was wirklich dahinter steckt. Ich war schon nach sehr kurzer Zeit völlig überrascht über die Möglichkeiten, die Scratch bietet. Es ist nicht nur eine triviale Spielerei für Kinder, sondern hat einen erstaunlich grossen Funktionsumfang. Mit etwas Ausdauer lassen sich komplexere Anwendungen und Spiele erstellen, was man bei den vielen veröffentlichen Projekten auf der Webseite auch sehen kann.

Besonders interessant finde ich die Verwendung des Broadcast Prinzips. Damit kann sehr intuitiv Code bei beliebig vielen anderen Figuren aufgerufen werden und sogar Parallelisierung umgesetzt werden. Aus meiner Sicht ist dies eine extreme Art der vielzitierten Entkopplung von Komponenten. Sender und Empfänger kennen sich nicht und beeinflussen sich nicht direkt. Etwas schade ist die Tatsache, dass keine Daten mitgesendet werden können, sondern der Weg über globale Variablen gemacht werden muss. Was zwar zu Scratch passt, aber als Entwickler zunächst nicht intuitiv ist.

Die exzessive "Klickerei", die Scratch bedingt, ist für mich persönlich nach etwas Zeit eher mühsam. Während das für Neueinsteiger und Kinder einfacher sein mag, bevorzuge ich die Arbeit mit der Tastatur und würde den Code lieber schreiben statt auf dem Bildschirm herumziehen.

Um Kindern und Jugendlichen, aber auch nicht-technikaffinen Erwachsenen, das Programmieren auf spielerische, verständliche Art beizubringen kann ich Scratch nur empfehlen. Besonders da grundlegende Konzepte imperativer (und teils auch der objektorientierten) Programmierung erhalten sind. Man kann eigene Funktionsblöcke mit Namen aufrufen, globale und lokale Variablen haben, es hat alle üblichen Schleifen und Bedingungen. Möchten Interessierte nach Scratch weiterfahren, müssen sie nicht alle gelernten Konzepte wieder über den Haufen werfen.

3.3. Persönliches Fazit Valentin Bürgler

Von Scratch hatte ich zuvor nur wenig gehört. Wie man allerdings beim Blick auf die Community erkennen kann, erfreut es sich gewaltiger Beliebtheit bei klein und gross. Es ist erstaunlich, welche "Artenvielfalt" und welche Kuriositäten sich auf dem Scratch Hub finden und "remixen" lassen. Neben allen anderen Stärken, welche die Sprache besitzt, vermute ich auch *darin* wenigstens zum Teil den Ursprung ihres Erfolgs. Jeder kann jeden inspirieren.

Mich hat die mit ein paar Ausnahmen sehr intuitive Benutzeroberfläche überzeugt. Innerhalb von wenigen Minuten findet sich auch jemand, der noch nie mit Scratch gearbeitet hat, mühelos zurecht. Man muss nicht etwa erst an der Hand gehalten durch die Entwicklungsumgebung geführt werden. Oder, wie in anderen Sprachen etwa, um ein Codesample überhaupt verstehen zu können, erst den kompletten Syntax erlernen. Alles lässt sich sofort begreifen und ist absolut selbsterklärend. Nicht umsonst ist die Firma Lego eine der Hauptunterstützerinnen des Scratch-Projektes.

Und doch stiess ich immer wieder an die Grenzen der Möglichkeiten, welche mir Scratch geboten hat. Besonders was die sehr eingeschränkte Parameterübergabe betrifft oder das komplette Fehlen von Polymorphismus. Ebensowenig lassen sich Referenzen auf Klone übergeben ohne ein Gebilde aus globalen Flags und Event Blöcken. Was vielversprechend und voller bunter Begeisterung beginnt, endet früher oder später nicht selten in einem overengineeretem Gebastel aus Kaugummi und Ducttape. Und trotzdem hatte ich Spass!