

IT2School

Gemeinsam IT entdecken



Modul KI-A3 – Große Gesten

Mit KI Geschichten erzählen

Eine Entwicklung in Kooperation von:

Inhalt

| | | |
|-------|---|----|
| 1 | Große Gesten – Mit KI Geschichten erzählen..... | 3 |
| 2 | Warum gibt es das Modul? | 4 |
| 3 | Ziele des Moduls..... | 4 |
| 4 | Die Rolle der Unternehmensvertreterin/des Unternehmensvertreters..... | 4 |
| 5 | Inhalte des Moduls..... | 5 |
| 5.1 | Gestenerkennung..... | 5 |
| 5.2 | Blockbasiertes Programmieren in der Entwicklungsumgebung Snap!..... | 7 |
| 6 | Unterrichtliche Umsetzung..... | 8 |
| 6.1 | Grober Unterrichtsplan..... | 9 |
| 6.2 | Stundenverlaufsskizzen | 10 |
| 6.2.1 | Doppelstunde 1 | 10 |
| 6.2.2 | Doppelstunde 2 | 12 |
| 6.2.3 | Doppelstunde 3 | 13 |
| 6.3 | Hinweise zur unterrichtlichen Umsetzung..... | 15 |
| 7 | Einbettung in verschiedene Fächer und Themen | 15 |
| 8 | Anschlussthemen..... | 15 |
| 9 | Literatur und Links | 15 |
| 10 | Arbeitsmaterialien | 16 |
| 11 | Glossar..... | 17 |
| 12 | Fragen, Feedback, Anregungen..... | 18 |
| 13 | Bildverzeichnis | 18 |

1 Große Gesten – Mit KI Geschichten erzählen

Gestenerkennung und verwandte Algorithmen verwenden Schülerinnen und Schüler in ihrem Alltag regelmäßig beispielsweise zur Steuerung von Touchscreens. Gesten wie das sogenannte „Swipen“ (nach links oder rechts wischen) oder der „Pinch-Zoom“ (Daumen und Zeigefinger einer Hand zusammen-/auseinanderbewegen, um auf dem Bildschirm angezeigte Inhalte zu vergrößern oder zu verkleinern) werden häufig und unbewusst angewandt und können geräteübergreifend eingesetzt werden. Nach einem spielerischen Einstieg über das Spiel Pictionary (auch bekannt als Montagsmaler) programmieren Schülerinnen und Schüler in diesem Modul eine Gestenerkennung selbst und lernen dabei, künstliche Intelligenz kreativ einzusetzen, um eigene Geschichten zu animieren.



| | |
|--|---|
| Lernfeld/Cluster: | Künstliche Intelligenz |
| Zielgruppe/Klassenstufe: | 4. bis 5. Klasse |
| | 6. bis 7. Klasse |
| | X 8. bis 10. Klasse |
| | X 11. bis 12. Klasse |
| Geschätzter Zeitaufwand: | 3 Doppelstunden (zusätzliche Stunden zur Vertiefung werden empfohlen) |
| Lernziele: | <ul style="list-style-type: none"> • Einsatz von Gestenerkennung im Alltag erkennen und beschreiben. • Den gesamten Prozess einer Künstlichen Intelligenz (Maschinelles Lernen) anhand eines Beispiels zur Gestenerkennung nachvollziehen und aktiv gestalten: Programmierung – Training – Anwendung • Kreative Möglichkeiten entwickeln, die erstellte KI zum Erzählen einer Geschichte zu verwenden. • Schwächen des erstellten Programms analysieren und Lösungsvorschläge erarbeiten. |
| Vorkenntnisse der Schülerinnen und Schüler: | Erforderlich: <ul style="list-style-type: none"> • Grundkenntnisse im Programmieren mit Snap! oder Scratch |
| Vorkenntnisse der/des Lehrenden: | Erforderlich: <ul style="list-style-type: none"> • Kenntnisse im Programmieren mit Snap! oder Scratch |
| Vorkenntnisse der Unternehmensvertreterin/des Unternehmensvertreters: | Empfohlen: <ul style="list-style-type: none"> • Kenntnisse im Programmieren mit Snap! oder Scratch |
| Sonstige Voraussetzungen: | <ul style="list-style-type: none"> • Pro 1-2 SuS sollte ein Computer oder Tablet zur Verfügung stehen • Ein Beamer zur Präsentation der Ergebnisse ist von Vorteil |



2 Warum gibt es das Modul?

Gestenerkennung gehört zu den ältesten Anwendungsbereichen künstlicher Intelligenz, die bereits seit den 1960er Jahren als Schnittstelle zwischen Mensch und Computer (Human Computer Interaction – HCI, siehe Glossar) dient. Sie bietet interaktive und kreative Steuerungsmöglichkeiten für elektronische Geräte, die über die Vielseitigkeit üblicher Peripheriegeräte hinausgehen. Den Schülerinnen und Schülern sind vielfältige Formen der Anwendung bereits bekannt, etwa durch das Entsperren des Smartphones, dem Swipen oder dem Spielen von Videospiele mit Bewegungssteuerung (Kinect/ Wii). In diesem Modul haben die Schülerinnen und Schüler die Möglichkeit, sich mit diesem Alltagsphänomen auseinanderzusetzen. Sie können ihre eigene Gesten-/ Zeichenerkennung programmieren, um so Phänomene im Zusammenhang mit Gestenerkennung im Alltag und deren Vorteile und Schwächen besser deuten und hinterfragen zu können.

3 Ziele des Moduls

Die Schülerinnen und Schüler...

- befassen sich mit Anwendungen künstlicher Intelligenz anhand eines alltäglichen Beispiels
- lernen die algorithmischen Grundbausteine zur Programmierung einer Gestenerkennung in Snap! kennen und wenden diese an
- implementieren eine Gestenerkennung basierend auf einem sogenannten Nearest Neighbor (nächster Nachbar) – Algorithmus
- setzen sich kreativ und handlungsorientiert mit Gestenerkennung auseinander
- reflektieren die Grenzen und Herausforderungen der Gestenerkennung in Bezug auf Bias und entwickeln entsprechende Lösungsansätze

4 Die Rolle der Unternehmensvertreterin/des Unternehmensvertreters

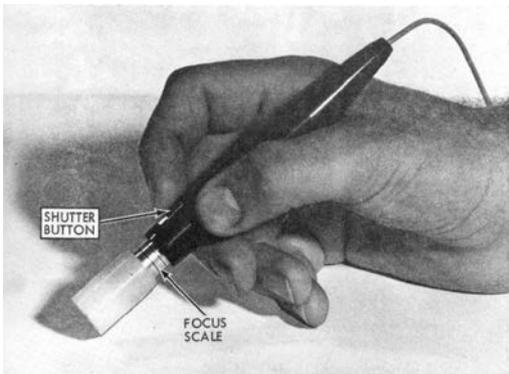
- Unterstützung der Lehrkraft bei der Durchführung des Unterrichts
- Erfahrungsberichte über den Einsatz von Gestenerkennung (oder ähnlichem) im Unternehmensumfeld, falls vorhanden.

5 Inhalte des Moduls

5.1 Gestenerkennung

Die Gestenerkennung ist ein Teilbereich der Informatik, der sich mit der Interpretation menschlicher Bewegungen mithilfe von Algorithmen befasst. Als Geste werden physische Bewegungen bezeichnet, die von einem Computer wahrgenommen werden können, wie etwa Hand- oder Kopfgesten. Ziel ist es, mithilfe der Gesten den Computer zu steuern und in Anwendungsprogrammen zu agieren.

Gestenerkennungen sind vergleichsweise frühe Formen der Mensch-Computer-Interaktion (HCI). Zahlreiche Grundlagen der HCI, unter anderem zur Gestenerkennung, wurden mit Ivan Sutherlands Sketchpad bereits in den 1960er Jahren gelegt (Sutherland, 1963). Dieses erste Programm mit einer grafischen Oberfläche konnte mithilfe von Bewegungen mit einem sogenannten Lichtgriffel gesteuert werden. Es wurde 1963 im Rahmen der Doktorarbeit von Ivan Sutherland entwickelt und damit ca. zeitgleich zum Bau des ersten Maus-Prototypen. (In Kapitel 9 *Literatur und Links* befindet sich ein Video mit einer Demonstration von Sketchpad.)



Lichtgriffel (CC BY SA 3.0, Kerry Rodden)



Ivan Sutherland demonstriert Sketchpad (CC BY SA 3.0, Kerry Rodden)

Bereits seit 1964 gibt es Programme, die in der Lage sind, Gesten automatisiert zu erkennen. So konnte beispielsweise das RAND Tablet mittels GRAIL (Graphical Input Language) gesteuert werden und 53 verschiedene handgeschriebene Ziffern, Symbole und Buchstaben erkennen und unterscheiden (Link zum Video in Kapitel 9). Ebenfalls 1964 wurde das erste trainierbare Programm zur Gestenerkennung entwickelt (Myers, 1998).

Heutzutage müssen Gesten nicht mit einem Stift auf einen Bildschirm gezeichnet werden. Sie können im zwei- oder dreidimensionalen Raum (z.B. Steuerung einer Kinect oder Wii) stattfinden und mit oder ohne Hilfsmittel ausgeführt werden. Dafür wird meist eine sogenannte TOF (Time of Flight)-Kamera verwendet. Diese sendet einen Lichtimpuls aus und misst für jeden Bildpunkt die Zeit, die ein Lichtstrahl zum Objekt und von dort wieder zurück zur Kamera benötigt. Dadurch kann für jeden Bildpunkt eine Entfernung zum nächsten Objekt berechnet werden (nähere Beschreibung von TOF-Kameras siehe Kapitel 9).

Typische Gesten, die heute zur Steuerung von Computern, Tablets oder Mobiltelefonen verwendet werden, sind beispielsweise Pinch-to-Zoom, das Zusammen- oder Auseinanderbewegen von zwei Fingern zum Zoomen, oder Swipen, um nach vorne oder zurückzublättern.

Als Gestenerkennungsalgorithmus in diesem Modul verwenden wir eine vereinfachte Form des \$1 Gesture Recognizers, der 2007 von Forschenden an der University of Washington und bei

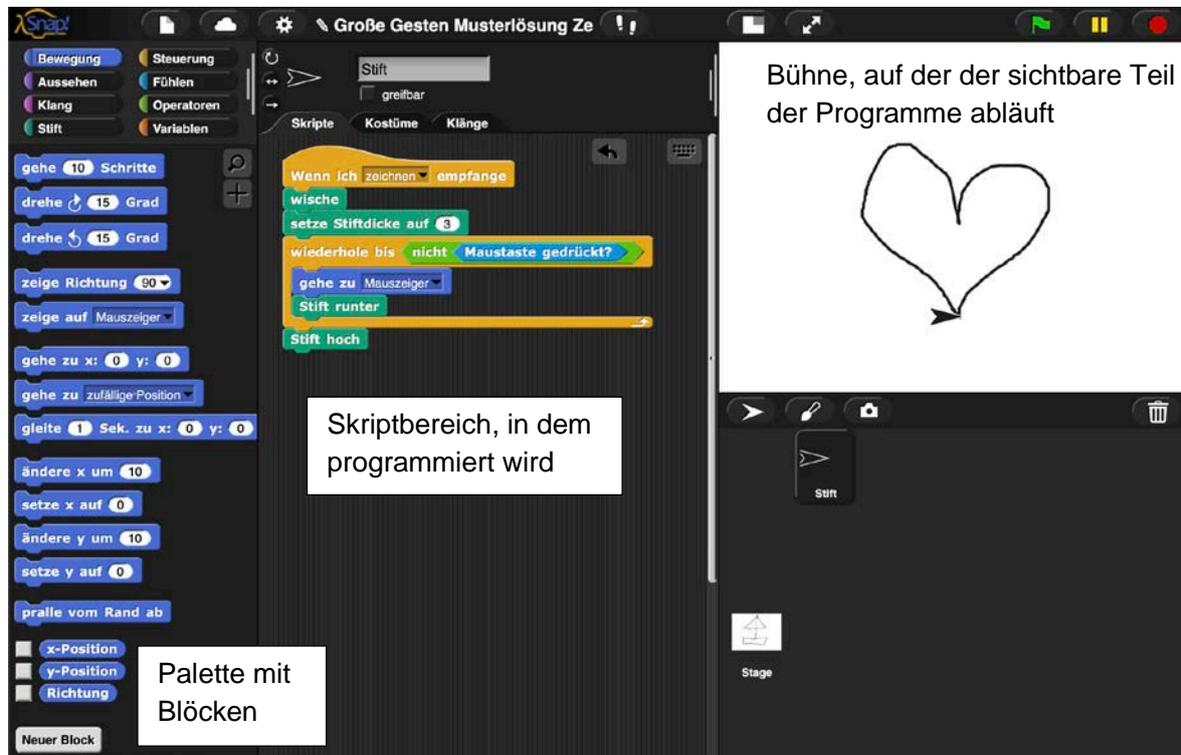
Microsoft veröffentlicht wurde (Wobbrock et al., 2007). Er dient der Erkennung von 2D Einstrichgesten (Unistroke Gestures), d.h. Gesten, die ohne Absetzen am Stück gezeichnet werden. Für dieses Modul ist er besonders geeignet, da er niedrigschwellig das eigene Prototyping im HCI-Design ermöglicht. Das gesamte Programm umfasst nur 100 Zeilen Code. Der \$1-Gesture Recognizer verwendet einen Nächste-Nachbar-Algorithmus (Nearest Neighbour). Diese Art von Programmen wird auch im Modul „Von Daten und Bäumen“ verwendet. In diesem Fall wird dabei der Pfad einer Geste mit einer Liste von Beispielpfaden verglichen. Der Pfad, der am ehesten mit der Zeichnung übereinstimmt, der nächste Nachbar, wird als korrekt erkannte Geste angenommen. Bei dieser simplen Form des Maschinellen Lernens handelt es sich um überwachtes Lernen (supervised learning), d.h. das Programm lernt anhand beschrifteter Eingaben, die als Beispiele übergeben werden.

Der eingesetzte Erkennungsalgorithmus (Nächster Nachbar) ist relativ leicht verständlich und kann von den Schülerinnen und Schülern selbstständig erarbeitet und programmiert werden.

Das Thema Gestenerkennung wurde für dieses Modul aus mehreren Gründen gewählt. Bereits erwähnt wurde die Tatsache, dass die Schülerinnen und Schüler Gestensteuerungen aus ihrem Alltag kennen. Außerdem handelt es sich dabei um eine Form der Künstliche Intelligenz, die verhältnismäßig wenige Beispiele zum Trainieren des Programms benötigt, um ein Ergebnis erzielen zu können. So könnte eine einfache Form des Programmes schon anhand zweier Beispiele zwischen zwei Gesten unterscheiden. Das Programm kann während der Erstellung immer wieder leicht getestet werden. Der eingesetzte Erkennungsalgorithmus (Nächster Nachbar) ist relativ leicht verständlich und kann von den Schülerinnen und Schülern selbstständig erarbeitet und programmiert werden. Aus einer Liste von Pfaden, die Punkt für Punkt verglichen werden, wird jeweils der ausgewählt, bei dem die insgesamt Differenz zum Pfad der aktuellen Geste am geringsten ist, also der „nächste Nachbar“. Mit den erkannten Gesten lassen sich hervorragend eigene Programme steuern. So bietet sich die Möglichkeit, künstliche Intelligenz kreativ einzusetzen, beispielsweise zum Storytelling. Wie bereits erwähnt eignet sich die Gestenerkennung auch dazu, problematische Bereiche der Künstlichen Intelligenz, wie Data Bias, erfahrbar zu machen.

5.2 Blockbasiertes Programmieren in der Entwicklungsumgebung Snap!

Für die Umsetzung des Moduls verwenden wir die browserbasierte Programmiersprache Snap! (<https://snap.berkeley.edu>). Snap! ist eine visuelle Programmiersprache, bei der Programmeinheiten, sogenannte Blöcke, durch Drag and Drop zu größeren Programmen zusammengesteckt werden können. Die Ergebnisse dieser Programme werden auf der sogenannten Bühne im rechten Bereich des Snap!-Fensters angezeigt (Einführung in Snap! – siehe Kapitel 9).



Die Snap! Entwicklungsumgebung

In diesem Modul werden besonders die über Scratch hinausgehenden Funktionen von Snap! verwendet, die es erlauben Blöcke zu erstellen, die Werte zurückgeben (Funktionen). Ebenfalls setzen wir benutzerdefinierte Blöcke ein, an die andere Blöcke als Parameter (Eingabe) übergeben werden können. Weitere Informationen darüber, wie benutzerdefinierte Blöcke in Snap! erstellt werden können, finden sich im Material für Lehrkräfte und Unternehmensvertreter*innen (A3-2.1-Blöcke in Snap!).

Beim ersten Öffnen ist Snap! standardmäßig auf englische Sprache eingestellt. Bei Wunsch kann die Sprache über das Einstellungsmenü (Zahnrad – Language) auf Deutsch geändert werden.

6 Unterrichtliche Umsetzung

Die Materialien im Zusammenhang mit diesem Modul gliedern sich in drei Bereiche, die im Rahmen von 3 Doppelstunden durchlaufen werden können.

Einführung

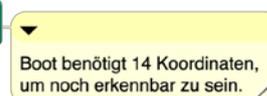
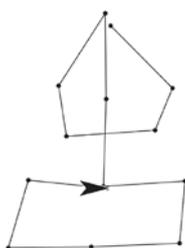
In der ersten Doppelstunde starten die Schülerinnen und Schüler mit einer Runde Pictionary (Montagsmaler), um sich auf das Thema der Stunde – Gestenerkennung und Erstellung eines Zeichenprogramms einzustimmen. Alternativ kann zum Einstieg der ersten Stunde auch einfach eine animierte Beispielgeschichte präsentiert werden (siehe: <https://tinyurl.com/GroßeGestenSchneewittchen>) Danach erarbeiten die Schülerinnen und Schüler, was Gestenerkennung ist und wie sie im Alltag eingesetzt wird. Darüber hinaus diskutieren sie, wo dabei Vor- und Nachteile liegen können. Nach dieser Einführung erstellen sie in Snap! eine erste Programmierung mit der sie Einstrichgesten aufzeichnen können. Hierfür werden Programmierblöcke verwendet, die das Zeichnen auf der Bühne ermöglichen.



Zeichenprogramm mit Ergebnis

Vertiefung

In der zweiten Doppelstunde wird die Gestenerkennung programmiert. Dafür müssen die Gesten aus dem Zeichenprogramm vergleichbar gemacht werden. Dies geschieht über Pfade, d.h. Listen von x- und y-Koordinaten, die während des Zeichnens gespeichert werden. Die Schülerinnen und Schüler erweitern ihr Zeichenprogramm, um damit Pfade aufzuzeichnen und wieder abzuspielen. Im Anschluss werden diese Pfade miteinander verglichen. Dafür ist es nötig, dass die Zeichnungen die gleiche Anzahl an Punkten aufweisen. Mithilfe eines „resample“-Blocks kann ausgewählt werden, auf wie viele Punkte sich ein gezeichneter Pfad verteilen soll.



Zeichnung eines Bootes normiert auf 14 Punkte mit dem "resample" Block

Nun kann die Differenz zwischen zwei Pfaden berechnet werden. Die Schülerinnen und Schüler trainieren nun ihr Programm mit mehreren Beispielgesten.

Im Anschluss daran haben die Schülerinnen und Schüler die Möglichkeit im Rahmen eines Abschlussprojekts Gesten zu verwenden und damit eine kurze Geschichte zu illustrieren.

Diese kann entweder selbstverfasst oder Bestandteil einer größeren Geschichte sein, die von der Lehrkraft vorgegeben wird. So können beispielsweise Kinderbücher oder Märchen als Grundlage für eine Geschichte dienen, die von der Klasse gemeinsam erarbeitet wird.

Präsentation und Abschluss

In der dritten Doppelstunde präsentieren die Schülerinnen und Schüler ihre Ergebnisse vor der Klasse. Dies kann entweder live mit einem Snap!-Projekt oder über ein vorher aufgezeichnetes Video erfolgen. Zum Abschluss des Moduls testen Schülerinnen und Schüler die Schwächen ihres Programms. Es gibt Einschränkungen in der Funktionalität dadurch, dass absolute Koordinaten zum Vergleich verwendet werden. So können beispielsweise anders positionierte oder stark in der Größe variierende Gesten nicht erkannt werden. Gleiches gilt, wenn die Geste in eine andere Richtung oder auf dem Kopf ausgeführt wird. Gemeinsam erarbeiten die Schülerinnen und Schüler Lösungsansätze, mit denen das Programm verbessert werden könnte, wie beispielsweise weitere Normierungsfunktionen (translatieren, rotieren, skalieren) und die Verwendung diverserer Beispiele.

6.1 Grober Unterrichtsplan

| Inhalte | Kurze Zusammenfassung | Dauer | Material |
|---|---|--------|-----------------------|
| Einführung in Gestenerkennung, Erstellung eines Zeichenprogramms, Animation der Zeichnungen | Die Schülerinnen und Schüler beschäftigen sich über eine Beispielgeschichte, ein Spiel und/oder eine Diskussion mit Gestenerkennung. Im Anschluss programmieren sie ein Zeichenprogramm für Einstrichgesten in Snap! Diese Gesten können in einer Erweiterung des erstellten Programms animiert werden. | 90 min | KI-A3.1.1 – KI-A3.1.3 |
| Speicherung von Pfaden, Implementierung des Nächster-Nachbar Algorithmus zur Erkennung der Gesten, Animation der erkannten Gesten | Um Gesten erkennen zu können, müssen diese vergleichbar gemacht werden, beispielsweise über Pfade. SuS erweitern ihr Programm, um Pfade zu speichern und zu vergleichen/erkennen. Als Hausaufgabe oder in einer weiteren Unterrichtsstunde konzipieren die SuS eine Geschichte, die sie mithilfe der Gesten steuern und illustrieren. | 90 min | KI-A3.1.4 – KI-A3.1.6 |
| Präsentation der Geschichten, Schwächen der Gestenerkennung und Verbesserungsmöglichkeiten. Diskussion über weitere Einsatzmöglichkeiten. | Die SuS präsentieren ihre Geschichten in der Klasse. Die Gestenerkennung wird reflektiert – Schwächen und ihre Lösungsansätze, Vor- und Nachteile sowie Einsatzmöglichkeiten werden diskutiert. | 90 min | KI-A3.1.7 |

6.2 Stundenverlaufsskizzen

Abkürzungen/Legende

AB = Arbeitsblatt/Arbeitsblätter; L = Lehrkraft; MuM = Mitschülerinnen und Mitschüler; SuS = Schülerinnen und Schüler;

UV = Unternehmensvertreterin/Unternehmensvertreter

6.2.1 Doppelstunde 1

| Zeit | Phase | Sozialform/ Lehrerimpuls | Inhalt/Unterrichtsgeschehen | Material |
|----------|--------------|-----------------------------|---|-----------------------------------|
| | Vorbereitung | | Ausdrucken bzw. Vorbereiten des Versands der Arbeitsmaterialien Falls Einstieg über Pictionary/Montagsmaler geplant ist, müssen Beispielbegriffe auf Zettelchen vorbereitet werden. Falls Einstieg über Geschichte geplant ist, muss diese vorbereitet werden. | Arbeitsmaterial KI-A3.1.1- 1.7 |
| 5-20 min | Einstieg | Plenum | <p>Variante 1: Stundeneinstieg über Pictionary/ Montagsmaler: Die Klasse bildet zwei Gruppen und lost aus, welche Gruppe beginnt. Die beginnende Gruppe bestimmt ein Mitglied, das als erstes malt. Eine Zeitdauer (z.B. 2 Minuten) wird festgelegt. Die malende Person zieht dann verdeckt einen der Zettel und versucht den darauf stehenden Begriff zu malen. Dabei darf sie nicht sprechen, "Ja" und "Nein" sind aber erlaubt. Die anderen Gruppenmitglieder versuchen anhand der Zeichnung den Begriff zu erraten. Ist der Begriff korrekt, zieht die malende Person sofort den nächsten Zettel und malt weiter. Das wiederholt sich so lange, bis die vorgegebene Zeit abgelaufen ist. Jeder erratene Begriff bringt der Gruppe einen Punkt. Ist die Zeit abgelaufen, darf die andere Gruppe malen und raten. Danach wird wieder gewechselt. Das Spiel endet, wenn alle mit malen dran waren, wenn eine bestimmte Punktzahl erreicht ist oder wenn es keine Zettel mehr gibt. Wer am Ende die meisten Punkte hat gewinnt.</p> <p>Variante 2: Stundeneinstieg über eine Beispielgeschichte: Die Lehrkraft bereitet eine Geschichte vor und trägt diese mit dem fertigen Gestenerkennungsprogramm der Klasse vor. Alternativ kann auch die hier zur Verfügung gestellte Beispiel-Geschichte verwendet werden.</p> | |

| | | | | |
|----------|-------------------------|-------------------------|--|---------------------------|
| 10 min | Hinführung | Plenum | <p>Das Thema der Unterrichtseinheit, Gestenerkennung, wird vorgestellt. Es wird kurz erläutert, was Gestenerkennung bedeutet. Dabei kann ein kurzer Rückblick auf Pictionary/Montagsmaler erfolgen (es gibt entscheidende und vernachlässigbare Details, gut zu malen ist keine Voraussetzung zum Erkennen)</p> <p>Im Anschluss überlegen die SuS wo ihnen Gestenerkennung im Alltag begegnet und tragen dies ins Arbeitsmaterial KI-A3-1.1 ein und teilen ihre Ergebnisse in der Klasse.</p> <p>Danach diskutieren die SuS kurz darüber, was die Vor- und Nachteile von Gestenerkennung (z.B. gegenüber Schrift oder Ton) sein könnten, und wo sie Anwendungsgebiete sehen.</p> | Arbeitsmaterial KI-A3.1.1 |
| 25 min | Erarbeitung (Zeichnen) | Einzel- oder Paararbeit | Programmierung eines Zeichenprogramms anhand des Arbeitsmaterials KI-A3-1.2 | Arbeitsmaterial KI-A3.1.2 |
| 5-10 min | Sicherung (Zeichnen) | Plenum | Gesamtlösung für das Zeichenprogramm im Plenum besprechen und sicherstellen, dass alle SuS für den weiteren Verlauf ein Snap! Projekt mit diesem Stand haben. | Musterlösung KI-A3 |
| 5 min | Hinführung (Animieren) | Plenum | <p>Eine mögliche Animation einer Geste wird von der Lehrkraft vorgestellt. Hierfür eignet sich beispielsweise das schlagende Herz. Das passende Animationsskript kann von der Lehrkraft selbst programmiert werden.</p> <p>Alternativ kann als Grundlage für die Vorstellung dieses Programm verwendet werden: https://tinyurl.com/GestenAnimieren</p> <p>Falls gewünscht, kann die Funktion des „animiere“-Blocks basierend auf Material KI-A3-2.1 genauer ausgeführt werden.</p> | Arbeitsmaterial KI-A3.2.1 |
| 25 min | Erarbeitung (Animieren) | Einzel- oder Paararbeit | Animation einiger Einstrichgesten anhand des Arbeitsmaterials KI-A3-1.3. Die SuS überlegen einige Beispiele für Einstrichgesten mit passenden Animationen und setzen diese in Snap! um. | Arbeitsmaterial KI-A3.1.3 |
| 5-10 min | Sicherung (Animieren) | Plenum | Einige mögliche Lösungen für die Animation von Gesten werden besprochen. Wenn die Zeit ausreicht, können einzelne SuS ihre Programme zeigen, ansonsten können einige Animationen kurz beschrieben werden. | Musterlösung KI-A3 |

6.2.2 Doppelstunde 2

| Zeit | Phase | Sozialform/ Lehrerimpuls | Inhalt/Unterrichtsgeschehen | Material |
|----------|------------------------------------|-----------------------------|---|---------------------------|
| 5-10 min | Hinführung | Plenum | Zusammenfassung, was bisher erarbeitet wurde. Erläuterung der Ziele dieser Doppelstunde <ul style="list-style-type: none"> - Gesten miteinander vergleichbar machen - Ähnlichste Geste ermitteln und ausgeben | |
| 5 min | Erarbeitung | Plenum | Einführung zu Pfaden: Gesten können als Bilder nicht direkt verglichen werden, deshalb müssen Zusatzinformationen mitgespeichert werden. Wo werden Pfade noch verwendet (z.B. Vektorgrafiken) und wie können sie in Snap! gespeichert werden. | |
| 15 min | Erarbeitung (Pfade erzeugen) | Einzel- oder Paararbeit | Speicherung von Pfaden (Koordinaten) anhand des Arbeitsmaterials KI-A3-1.4 Erweiterung des Snap! Projekts um eine Variable, in der beim Zeichnen der jeweilige Pfad gespeichert wird. | Arbeitsmaterial KI-A3.1.4 |
| 45 min | Erarbeitung (Pfade vergleichen) | Einzel- oder Paararbeit | Vergleichen von Pfaden anhand des Arbeitsmaterials KI-A3-1.5 Anzahl an Punkten verändern mit „resample“ (ca. 15 min): Die SuS testen die Funktion des „resample“ Blocks, um Zeichen auf bestimmte Anzahl von Punkten zu normieren. Unterschied zwischen Pfaden messen (5-10 min): Die SuS testen mit dem „Unterschied“ Block, wie verschieden zwei Pfade sind Beispiele Speichern (10 min): Die SuS trainieren ihr Programm mit mehreren Beispielgesten mit passender Beschriftung, die in einer Variable gespeichert werden. Ähnlichste Geste ermitteln (15-20 min): Die SuS erstellen mithilfe eines Parson's Puzzles einen „erkenne“-Block, der die eigentliche Gestenerkennung ausführt. Dieser gibt den Namen des erkannten Pfads (dem nächsten Nachbarn) aus der Liste der Beispiele aus. | Arbeitsmaterial KI-A3.1.5 |

| | | | | |
|---|----------------------------|-------------------------|---|---------------------------|
| | | | Es empfiehlt sich, nach jeder der Arbeitsphasen eine kurze Sicherungsphase einzuplanen, um sicherzustellen, dass alle SuS mit demselben Stand ihrer Snap!-Projekte weitermachen. An dieser Stelle sollte auch darauf hingewiesen werden, dass es sich hierbei um einen Algorithmus des überwachten maschinellen Lernens handelt. | |
| - | Erarbeitung (Storytelling) | Einzel- oder Paararbeit | Erstellen einer animierten Geschichte anhand des Arbeitsmaterials KI-A3-1.6 Je nach verfügbarer Zeit ist dieser Teil des Moduls als Hausaufgabe, Extraeinheit oder den Rest der Stunde denkbar. SuS können eigene Kurzgeschichten entwickeln und animieren. Alternativ kann die L. mehrere Kapitel/Abschnitte eines Buches oder einer längeren Geschichte zur Verfügung stellen, die die SuS animieren sollen, um eine zusammenhängende längere Geschichte zu erzeugen. Optional kann die Anzahl der verwendbaren Gesten oder die Dauer der Abschlusspräsentation vorgegeben werden. | Arbeitsmaterial KI-A3.1.6 |

6.2.3 Doppelstunde 3

| Zeit | Phase | Sozialform/ Lehrerimpuls | Inhalt/Unterrichtsgeschehen | Material |
|--------|--------------|-----------------------------|---|----------|
| 5 min | Hinführung | Plenum | Ziele der Stunde (Präsentation und Reflexion) und Ablauf der Präsentationsphase erläutern | |
| 10 min | Hinführung | Einzel- oder Paararbeit | Optionale Übung der Präsentationen | |
| 30 min | Präsentation | Plenum | Alle SuS präsentieren ihr Projekt in der Klasse. Dies kann live mithilfe des Snap!-Projekts erfolgen. Das Projekt kann wie in Lehrkräftematerial KI-A3-2.2 erläutert über URL oder als Datei geteilt werden. Alternativ können die SuS vorher ein Video (Screencast mit Ton) erstellen, das während der Präsentationsphase abgespielt wird. | |

| | | | | |
|--------|------------------------------|------------------------------|--|---------------------------|
| | | | Das Projekt kann wie in Lehrkräftematerial KI-A3-2.2 erläutert über URL oder als Datei geteilt werden. | |
| 15 min | Erarbeitung (Reflexion) | Paararbeit | Reflexion der Gestenerkennung anhand des Arbeitsmaterials KI-A3-1.7 Welche Fälle kann dein Programm nicht erkennen? In Zweiergruppen die im Arbeitsmaterial beschriebenen Fälle testen. Rückblick auf die Einstiegsdiskussion zu Anwendungsmöglichkeiten, Vor- und Nachteile von Gestenerkennung. | Arbeitsmaterial KI-A3.1.7 |
| 10 min | Sicherung (Reflexion) | Plenum | Die Ergebnisse werden gesammelt und beispielsweise in Form eines Tafelaufschriebs festgehalten. Besonders gut eignet sich dafür eine Tabelle, in deren zweiter Spalte die Lösungsansätze der nächsten Aufgabe notiert werden können. | |
| 15 min | Erarbeitung (Reflexion 2) | Plenum oder Gruppenarbeit | Reflexion der Gestenerkennung anhand des Arbeitsmaterials KI-A3-1.7 Vorurteile (Bias): Wie benachteiligt dein Programm Menschen und was kann man dagegen tun? In der Gruppe oder im Plenum wird gesammelt, wie sich die erarbeiteten Probleme auf bestimmte Personengruppen auswirken können. Es werden Lösungsansätze erarbeitet, um dies zu mitigieren: <ul style="list-style-type: none"> - Mehr und diversere Beispiele - Verwendung von Gesten, die sich deutlicher unterscheiden - Einführung weitere Funktionen wie translatieren, rotieren und skalieren | Musterlösung KI-A3 |
| 5 min | Sicherung | Plenum | Die erarbeiteten Lösungsansätze werden in der zweiten Spalte der Tabelle auf der Tafel festgehalten. | |

6.3 Hinweise zur unterrichtlichen Umsetzung

Ein Beispielprojekt das zum Einstieg in den Unterricht verwendet werden kann, findet sich hier: <https://tinyurl.com/GrosseGestenBeispiel>.

Ein leeres Starterprojekt, das bereits alle benötigten Blöcke zur Durchführung des Moduls enthält, befindet sich hier: <https://tinyurl.com/GrosseGesten>.

Ein Lösungsprojekt mit zwei Beispielen ist unter folgendem Link abgelegt: <https://tinyurl.com/GrosseGestenLoesung>.

7 Einbettung in verschiedene Fächer und Themen

Dieses Modul eignet sich neben dem Einsatz im Informatikunterricht auch für weitere Fächer. Insbesondere die weiterführenden Aktivitäten, die dafür sorgen sollen, dass die Gestenerkennung „fairer“ wird, sind im Mathematikunterricht einsetzbar. Dabei werden Blöcke erstellt, mit denen die Pfade skaliert, rotiert und translatiert werden können.

Aufgrund der Abschlussaufgabe eignet sich das Projekt natürlich auch für den Einsatz im Sprachunterricht, wo SuS ihre eigenen Kurzgeschichten illustrieren können. Alternativ kann auch eine Lektüre in mehrere Teile geteilt werden. Diese müssen dann von den SuS zusammengefasst und animiert werden. Im Anschluss kann daraus eine größere zusammenhängende Geschichte animiert werden.

8 Anschlusssthemen

Zur Vertiefung des Konzepts des überwachten Lernens beziehungsweise den Vergleich zu weiteren Formen des maschinellen Lernens eignen sich die KI-Module B3 – Schlag den Roboter und A2 – Die Bananenjagd.

Eine mögliche Reihung, die man auch für eine Projektwoche nutzen kann, ist:



9 Literatur und Links

- Sutherland, Ivan Edward (1963): Sketchpad: A man-machine graphical communication system. Massachusetts Institute of Technology, PhD
- Video zu Sketchpad von Ivan Sutherland mit Kommentaren von Alan Kay (Englisch): <https://www.youtube.com/watch?v=495nCzxM9PI>
- Myers, Brad A. (1998): A Brief History of Human Computer Interaction Technology. ACM interactions. Vol. 5, no. 2. pp. 44-54.

- Video zu GRAIL (Englisch):
<https://www.youtube.com/watch?v=2Cq8S3jzJiQ>
- So funktioniert die Kinect Tiefenkamera: <https://docs.microsoft.com/de-de/azure/kinect-dk/depth-camera>
- Wobbrock, Jacob, Wilson, Andrew, Li, Yang (2007): Gestures without libraries, toolkits or training: A \$1 recognizer for user interface prototypes. UIST: Proceedings of the Annual ACM Symposium on User Interface Software and Technology. 159-168.
Link zum Original-Paper: <http://faculty.washington.edu/wobbrock/pubs/uist-07.01.pdf>
- Einführung in Snap!: <https://open.sap.com/courses/snap1-1-de>
- Video zum Bau des Gestenerkenners in Snap! (Englisch):
<https://www.youtube.com/watch?v=vhnWey4IFlo>

10 Arbeitsmaterialien

| Nr. | Titel | Beschreibung |
|----------------------------|----------------------------------|---|
| 😊 KI-A3.1.1 | Gestenerkennung im Alltag | Arbeitsblatt mit Fragestellungen zur Verwendung von Gestenerkennung sowie deren Vor- und Nachteile. |
| 😊 KI-A3.1.2 | Ein Zeichenprogramm erstellen | Arbeitsblatt mit Programmierhinweisen zur Erstellung eines Zeichenprogramms für Einstrichgesten. |
| 😊 KI-A3.1.3 | Zeichnungen animieren | Arbeitsblatt mit Programmierhinweisen zur Animation von Malspuren. |
| 😊 KI-A3.1.4 | Pfade aufzeichnen | Arbeitsblatt mit Programmierhinweisen zum Speichern von Pfaden, d.h. Listen aus x- und y-Koordinaten. |
| 😊 KI-A3.1.5 | Pfade miteinander vergleichen | Arbeitsblatt mit Erklärungen zum „resample“ und „Unterschied zwischen“ Block. Programmierhinweise zur Erstellung des „erkenne“ Blocks, d.h. der Implementierung des Nächsten Nachbar Algorithmus. |
| 😊 KI-A3.1.6 | Eine Geschichte erzählen | Arbeitsblatt mit Erklärungen zum Nachrichten versenden in Snap!. Arbeitsauftrag für das Storytelling-Projekt. |
| 😊 KI-A3.1.7 | Gestenerkennung Reflexion | Arbeitsblatt mit Anweisungen zum Testen des Programms auf Biases. |
| 😊 KI-A3 Zusatzmaterial | Blöcke in Snap! | Blockglossar für Blöcke, die nicht in Scratch vorhanden sind. Benutzerdefinierte Blöcke in Snap! erstellen. Erklärung benutzerdefinierter Blöcke, die im Modul verwendet werden. |
| 😊 KI-A3 Blockbibliothek | Blockbibliothek für dieses Modul | Blockbibliothek für Snap!, die alle benutzerdefinierten Blöcke enthält, die für das |

| | | |
|---------------------------|------------------------------------|---|
| | | Modul benötigt werden. Kann in das Snap!-Fenster gezogen oder über <i>Dateimenü</i> -> <i>Importieren</i> in Snap! geladen werden.“ |
| ☺ KI-A3 Zusatzmaterial | Speichern und Teilen von Projekten | Speichern von Projekten in der Cloud und lokal als Datei. Teilen von Projekten, die in der Cloud gespeichert sind. |
| 😊 KI-A3 Musterlösung | Musterlösung | Lösungen zu allen Aufgabenstellungen inklusive Links zu den Projekten. |

Legende

- ☺ Material für Schülerinnen und Schüler
- 😊 Material für Lehrkräfte sowie Unternehmensvertreterinnen und Unternehmensvertreter
- ☺ Zusatzmaterial

11 Glossar

| Begriff | Erläuterung |
|--------------------------------------|--|
| Animation | Erzeugen eines bewegten Bildes durch rasche Abfolge von Standortveränderungen, Grafikeffekten oder Teilbildern. |
| Bias | Verzerrung statistischer Ergebnisse aufgrund falscher Untersuchungsmethoden. Im Bereich Künstliche Intelligenz häufig aufgrund nicht repräsentativer Trainingsdaten. |
| Einstrichgeste (Unistroke Gesture) | Bewegung, die ein Symbol in einem Zug ohne Absetzen aufzeichnet. |
| HCI (Human Computer Interaction) | Forschungsfeld, das sich mit dem Design und der Verwendung von Computer-Technologie an der Schnittstelle zwischen Menschen und Computern beschäftigt. Entwickelt und erforscht Methoden, wie Menschen mit Computern interagieren können. |
| Nächster Nachbar (Nearest Neighbour) | Algorithmus, der aus einer Liste von Kandidaten den auswählt, der bestimmten Kriterien am nächsten kommt. In diesem Fall die Geste aus einer Liste von Beispielgesten, deren Pfad dem Pfad der aktuellen Zeichnung am ähnlichsten ist. |
| Storytelling | Mitteilen von Erkenntnissen, Fakten oder Meinungen, indem sie mit eigenen Erfahrungen und Erlebnissen verknüpft und in für das Zielpublikum ansprechende Geschichten gekleidet werden. |

12 Fragen, Feedback, Anregungen

Sie haben das Modul ausprobiert und nun Fragen, Anregungen oder Feedback für uns? Darüber freuen wir uns, denn mit Ihren Erfahrungen können wir Schritt für Schritt einen FAQ (Frequently Asked Questions) für die neuen KI-Module aufbauen oder die Module weiter entwickeln.

Bitte füllen Sie folgende Umfrage über SurveyMonkey aus: <https://bit.ly/3DAemzw> über den folgenden QR-Code kommen Sie ebenfalls zur SurveyMonkey-Umfrage:



Sie können sich auch gerne unter bildung@wissensfabrik.de melden.

13 Bildverzeichnis

- S. 6 Lichtgriffel: CC BY SA 3.0 Kerry Rodden, von Wikimedia Commons: <https://commons.wikimedia.org/wiki/File:SketchpadDissertation-Fig4-1.tif>
- S. 6 Sketchpad: CC BY SA 3.0 Kerry Rodden, von Wikimedia Commons: <https://commons.wikimedia.org/wiki/File:SketchpadDissertation-Fig1-2.tif>

Gestenerkennung im Alltag

Wir alle benutzen Gestenerkennung im Alltag in verschiedenen Situationen. Dabei verwenden wir häufig nur unsere Hände, z.B. um zu kommunizieren, wo Unterhaltungen nicht möglich sind. Oft werden Gesten auch digital verarbeitet, z.B. zur Steuerung von Computern.

Überlege, wo dir Gestenerkennung im Alltag begegnet:

Gesten werden häufig zur Steuerung von Computern verwendet. Mit der sogenannten „Pinch-Zoom“-Geste, bei der Daumen und Zeigefinger einer Hand zusammen- oder auseinanderbewegt werden, können Dinge auf einem Bildschirm verkleinert oder vergrößert werden. Diskutiere anhand dieses Beispiels, was die Vorteile und Nachteile von Gesten zur Steuerung bestimmter Anwendungen sein könnten. Fallen dir weitere Vor- oder Nachteile von Gestenerkennung ein?

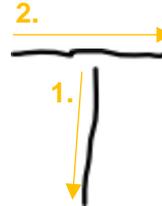
¹ Von englisch „pinch“ = kneifen

Ein Zeichenprogramm erstellen

Es gibt mehrere Formen von Gesten, z.B. bei Mustern zum Entsperren eines Mobiltelefons. Einige Gesten bestehen aus nur einem Strich, manche aus mehreren Strichen. Unter sogenannten **Einstrichgesten** versteht man Gesten, die an einem Stück, ohne abzusetzen, erzeugt werden. **Mehrstrichgesten** bestehen aus mehreren Strichen:



Einstrichgeste, die am Stück durchgeführt wurde.



Mehrstrichgeste, die in zwei Bewegungen erstellt wurde.

Zeichenprogramm selbst gemacht mit Snap!

Um später eine Gestenerkennung programmieren können, benötigst du zuerst ein Programm, mit dem du Einstrichgesten aufzeichnen kannst.

Dieses Zeichenprogramm soll:

- starten, sobald die Maus auf der Bühne¹ gedrückt wird.
- mögliche vorherige Zeichnungen von der Bühne wischen.
- die Bewegung der Maus auf der Bühne zeichnen, solange die Maus gedrückt ist.
- die Bewegung der Maus nicht mehr zeichnen, sobald die Maus nicht mehr gedrückt ist.

Überlege zunächst selbst oder mit anderen, wie du ein Zeichenprogramm programmieren kannst, das die oben genannten Anforderungen erfüllt.

Wenn du einen Denkanstoß benötigst, gibt es Tipps dafür auf der nächsten Seite.

Hinweis: Wenn du dein Zeichenprogramm in dem Projekt programmierst, das unter dieser URL zu finden ist: <https://tinyurl.com/GrosseGesten>, hast du fürs nächste Mal bereits alle benötigten Blöcke zur Verfügung.

¹ Als Bühne bezeichnet man das Fenster im oberen rechten Bereich von Snap!, in dem die Programme abgespielt werden.

Tipps zum Programmieren des Zeichenprogramms

Diese Blöcke könnten für das Zeichenprogramm hilfreich sein:



So baust du die Blöcke zusammen:

- starten, sobald die Bühne angeklickt wird:

Starte dein Zeichenprogramm **von der Bühne aus**, indem du eine Nachricht an alle Objekte sendest, sobald die Maus auf der Bühne gedrückt wird. Das Objekt, das die Geste zeichnet, muss mit dem „Wenn ich empfangen“-Startblock auf die Nachricht reagieren.



- mögliche vorherige Zeichnungen von der Bühne wischen:

Nachdem das Objekt die Nachricht empfangen hat, soll es die Bühne wischen.



- die Bewegung der Maus auf der Bühne zeichnen, solange die Maus gedrückt ist:

Wiederhole, solange die Maus gedrückt ist, kann auch als „wiederhole, bis nicht Maus gedrückt“ formuliert werden. „Gehe zum Mauszeiger“ und „Stift runter“ sind die Blöcke, die wiederholt werden müssen.



- Das Objekt nicht mehr „greifbar“ machen.

Da sich dein Objekt wegen des „gehe zu Mauszeiger“ Blocks unter deiner Maus befindet, hebst du es von der Bühne an. Es kann daher nicht malen. Um das zu vermeiden, musst du das Objekt nicht mehr „greifbar“ machen, indem du einmal auf die passende Box in den Objekteigenschaften klickst.

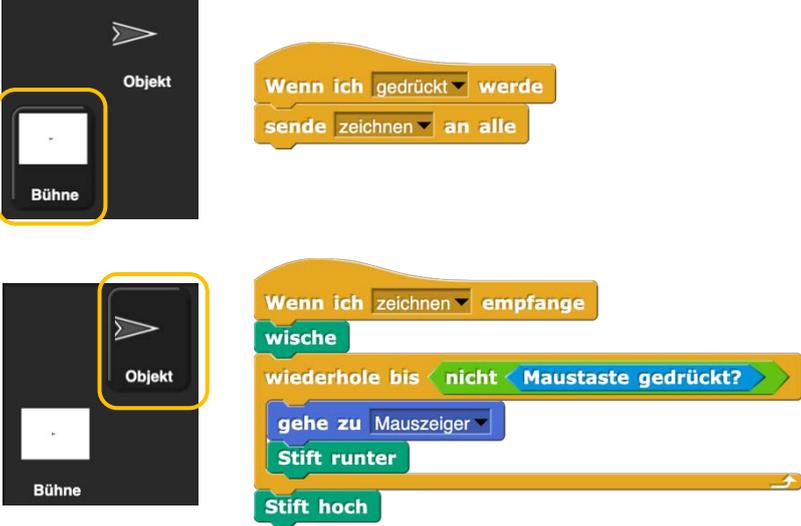


- die Bewegung der Maus nicht mehr zeichnen, wenn die Maus nicht mehr gedrückt ist:

Wenn die „wiederhole bis“ Schleife abgeschlossen ist (also die Maus nicht mehr gedrückt), muss der „Stift hoch“ Block verwendet werden, damit weitere Bewegungen auf der Bühne nicht mehr gezeichnet werden.



- Das gesamte Zeichenprogramm:



Testen des Zeichenprogramms

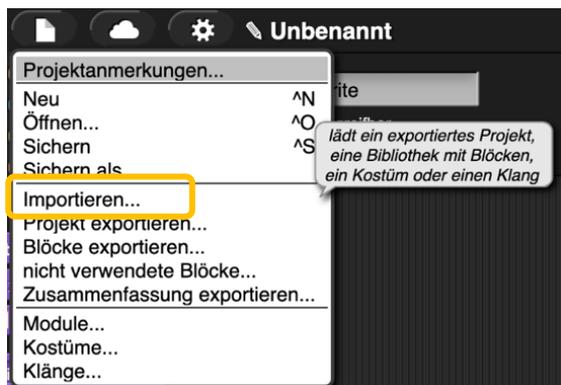
Überlege dir einige Einstrichgesten und versuche, sie mit deinem Zeichenprogramm zu erstellen.

Versuche nun Mehrstrichgesten. Warum funktionieren diese nicht?

Striche in Bewegung bringen

Da du das Zeichenprogramm selbst programmierst, kannst du es tun lassen, was du willst. Wie wäre es, wenn du deinen Zeichnungen nun Bewegungen beibringst?

In diesem Teil sollen die Zeichnungen mithilfe des „animiere“-Blocks zum Leben erweckt werden. Falls du dein Projekt beim letzten Mal von der URL <https://tinyurl.com/GrosseGesten> gestartet hast, kannst du direkt daran weiterprogrammieren. Ansonsten musst du das Zusatzmaterial **KI-A3 Blockbibliothek Große Gesten** in dein Projekt importieren. Du kannst Dateien entweder einfach in Snap! ziehen oder über Dateimenü -> Importieren... von deinem Computer laden.



Danach findest du den „animiere“ Block, wie alle benutzerdefinierten Blöcke, am unteren Ende der jeweiligen Kategorie.

Teste den „animiere“ Block mit einfachen Beispielen.

Erzeuge dafür mit deinem Zeichenprogramm eine Zeichnung auf der Bühne. Füge dann jeweils eines der folgenden Skripte in das C-förmige Eingabefeld des „animiere“ Blockes ein und teste, was passiert:



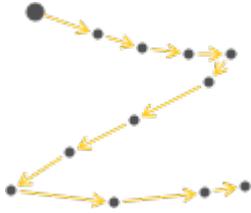
Überlege dir zwei Zeichen, die mit Einstrichgesten erzeugt werden können, und dazu passende Animationsideen.

| | | | |
|-----------------------------|--|--|--|
| Zeichen | Beispiel:  | | |
| Animations- idee | Herzschlag erzeugen. Zeichnung mehrmals verkleinern und vergrößern | | |

Setze deine Ideen in deinem Snap! Projekt um.

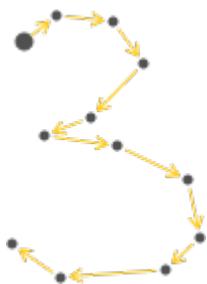
Punkt vor Strich

In diesem Abschnitt soll es darum gehen, eine Zeichnung als Bewegung umzusetzen, also nicht als das Bild bzw. Striche selbst, sondern als eine Abfolge von Punkten, die ein Malstift durchläuft, um etwas zu zeichnen. Eine solche Abfolge von Punkten nennt man "Pfad¹".

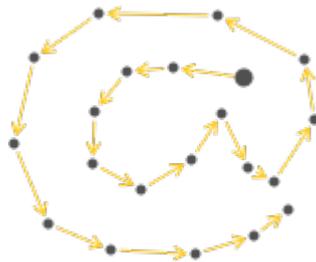


Beispiel für einen "Zickzack"-Pfad, der ein Z-förmiges Zeichen malt.

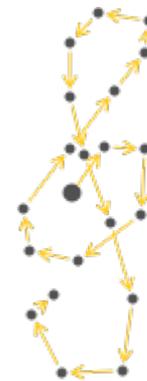
Weitere Beispiele für Pfade, die verschiedene Zeichen darstellen:



die Zahl 3



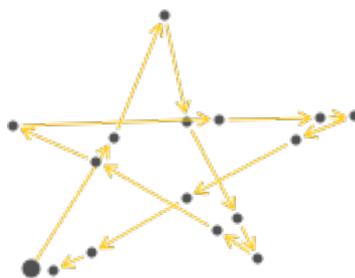
das E-Mail-Zeichen @



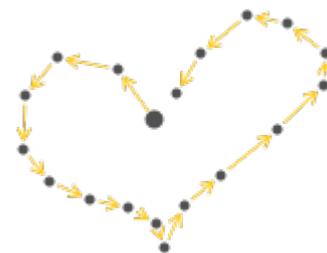
Ein Sopran-Notenschlüssel



Die Zahl 8



ein Stern



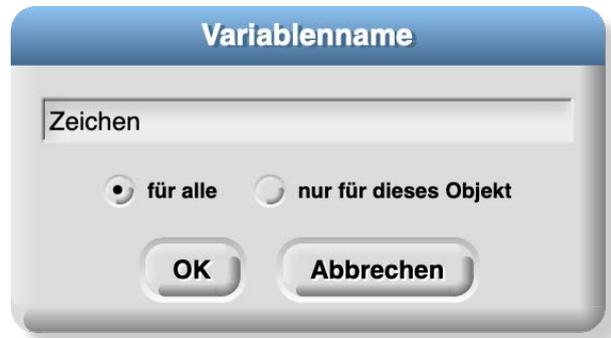
ein Herz

Allen diese Zeichen ist gemeinsam, dass du sie mit deinem Programm in einem einzigen Zug zeichnen kannst, also ohne den Stift abzusetzen.

¹ Wusstest du, dass Vektorgrafiken auch als Pfade von Punkten (Vektoren) gespeichert werden? Das Tolle daran ist, dass man die Grafik beliebig vergrößern kann, ohne dass die Auflösung schlechter wird.

Einen Pfad aufzeichnen

Um einen Pfad zu speichern, benötigst du eine Variable. Am besten gibst du ihr einen aussagekräftigen Namen, also z.B. "Zeichen":



In dieser neuen Variablen soll nun jedes Mal, wenn dein Programm ein Bild malt, die Abfolge der durchlaufenen Punkte aufgezeichnet werden. Dafür benötigst du die folgenden Blöcke bzw. Anweisungen:



Die erste Anweisung initialisiert die Variable als Liste. "Initialisieren" heißt hier, dass die Variable zu einer neuen, leeren Liste wird. Das soll immer dann passieren, wenn du ein neues Zeichen malst, also am Anfang deines Zeichenprogramms.

Eine Liste speichert Daten der Reihe nach. Elemente einer Liste können Daten aller Art sein, also Zahlen, Text, Figuren, Kostüme, Klänge, Blöcke und natürlich auch andere Listen.

Wie also kannst du eine Position, einen geometrischen Punkt, in einer Liste speichern? Die Position deines Objekts und damit des Malstiftes hat zwei Werte, eine x-Position und eine y-Position. Du kannst also einen Punkt als Liste mit zwei Werten beschreiben. Genau das macht die zweite Anweisung.

Füge die beiden Anweisungen in dein Malprogramm ein, damit die Liste

- am Anfang einer neuen Geste auf eine leere Liste gesetzt wird.
- die aktuelle x- und y-Position nach jedem Positionswechsel zur Liste hinzugefügt wird.

Einen Pfad abspielen

Was man aufgezeichnet hat, kann man natürlich auch wieder abspielen. Anstatt zum "Mauszeiger" zu gehen, wie in diesem Block,

gehe zu Mauszeiger

kann eine Figur auch an einem bestimmten Punkt gehen, der eine x-Position und eine y-Position hat:

gehe zu x: 7 y: 12

gehe zu Liste 7 12

Diese Blöcke machen beide dasselbe, sie bewegen die Figur zur Koordinate [7, 12].

Um eine Liste der Reihe nach von vorn bis hinten "abzuspielen", kannst du Schleifenblöcke verwenden. Einer davon ist dafür ganz besonders praktisch, du findest ihn bei den anderen Listenblöcken:

für jedes Element von

Dieser "für jedes Element von _" Block nimmt der Reihe nach jedes Element aus einer Liste und führt damit diejenigen Blöcke aus, die eingerückt in seinem C-förmigen Bereich stecken. Der Block hat eine eingebaute Variable namens "Element". Von dieser Variablen kannst du ganz einfach Kopien "abziehen" und in anderen Blöcken innerhalb der Schleife verwenden.

Damit kannst du jetzt ein kleines Wiedergabeprogramm bauen. Diese drei Blöcke helfen dir dabei:

wische

Stift runter

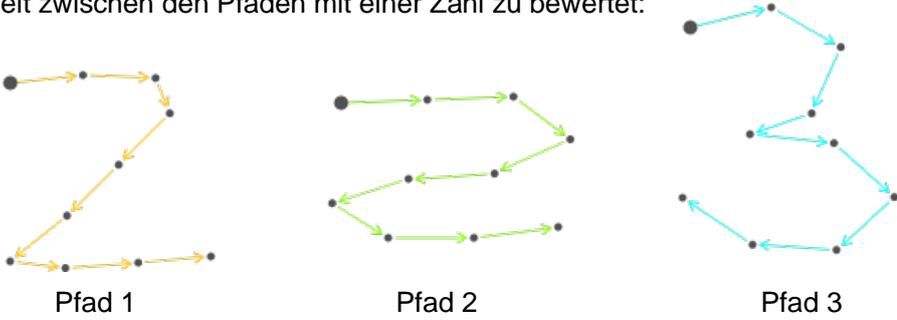
Stift hoch

Teste dein Programm! Male ein Zeichen, danach klicke auf das Abspielprogramm. Wenn du alles richtig gemacht hast, dann gibt das Programm dein Zeichen exakt so wieder, wie du es zuvor gemalt hast.

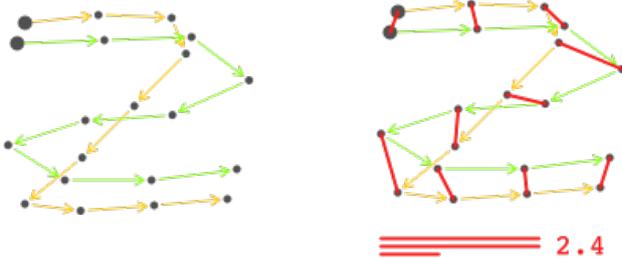
G3N4U H1NG35CH4U7¹

Hast du die Überschrift lesen können? Falls ja, warum? Falls nein, lies die Fußnote und überlege dann.

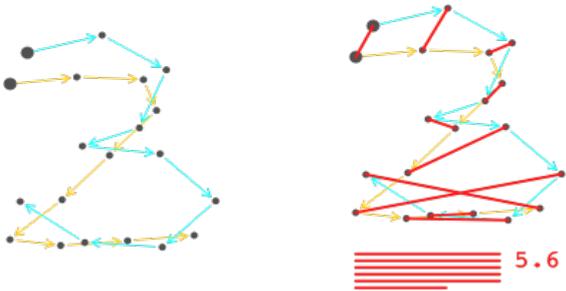
In diesem Abschnitt lernst du, wie Computer den Unterschied bzw. die Ähnlichkeit zwischen verschiedenen Zeichen bzw. ihrer Pfade bestimmen. Die Idee ist, Pfade miteinander zu vergleichen, indem man jeweils zwei übereinanderlegt und den Unterschied bzw. die Gleichheit zwischen den Pfaden mit einer Zahl zu bewertet:



Man kann - wenn beide Pfade die gleiche Anzahl Punkte haben - für jeden Punkt des einen Pfads die Entfernung zum entsprechenden Punkt des anderen Pfads ermitteln:



Die Summe der Entfernungen jedes Punktepaars kann dann als "Unterschied" zwischen beiden Pfaden dienen. Auf diese Weise kann man vergleichen, ob der Unterschied zu einem dritten Pfad kleiner oder größer ist:



In diesem Beispiel sind die Entfernungen zwischen den entsprechenden Punkten des ersten und dritten Pfads insgesamt deutlich länger als die die zum zweiten Pfad. Der zweite Pfad ist dem ersten deshalb ähnlicher als der dritte.

¹ Genau hingeschaut, Diese Art der Darstellung nennt man auch Leet-Speak. Leetspeak ['li:tspi:k] (auch Leetspeek, 1337; von engl. elite, „Elite“, und speak, „sprechen“) bezeichnet das Ersetzen von Buchstaben durch ähnlich aussehende Ziffern oder Sonderzeichen. Es ist vor allem in der Nerd-Szene gebräuchlich und soll maschinelle Worterkennung erschweren.

Die Anzahl der Punkte verändern

Um zwei Zeichen Punkt für Punkt miteinander zu vergleichen, müssen beide Pfade gleich viele Punkte haben. Mit dem "resample" Block kannst du bestimmen, auf wie viele Punkte sich ein gezeichneter Pfad verteilen soll:

resample auf **64 Punkte**

Der "resample" Block hat zwei Eingaben: Einen Pfad (Liste) und die Anzahl der gewünschten Punkte. Er gibt einen neuen Pfad zurück, der genau die Anzahl gewünschter Punkte enthält, und die ursprüngliche Zeichnung so gut wie möglich abbildet.

Du kannst die Punkte und ihre Verteilung in einem Pfad mit dem "zeichne" Block sichtbar machen:

zeichne

Wenn du etwas mit deinem Zeichenprogramm zeichnest, z.B. eine Doppelschleife:



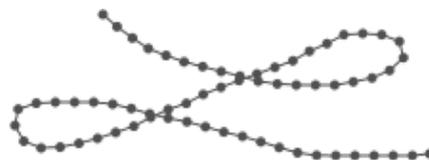
dann kannst du die von dir aufgezeichneten einzelnen Punkte sichtbar machen mit:

zeichne Zeichen

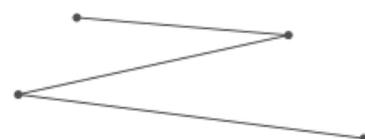


Mit dem "resample" Block kannst du nun die Anzahl der Punkte genau bestimmen und gleichmäßig über den Pfad verteilen:

zeichne resample Zeichen auf 64 Punkte



Die Anzahl der Punkte kannst du frei wählen. Nimmst du eine sehr hohe Zahl, dann dauert es länger, bis du alle Punkte miteinander verglichen hast. Wählst du aber eine zu niedrige Zahl, dann verlieren deine Zeichnungen irgendwann die Merkmale, die sie unterscheidbar machen.



Teste für mehrere Zeichen, wie viele Punkte du benötigst, damit du sie noch erkennen kannst. Versuche Zeichen zu finden, bei denen diese Anzahl an Punkten sich unterscheidet:

| Zeichen | Zeichen mit 10 Punkten | Anzahl an Punkten, bei der Zeichen noch erkennbar ist |
|---------|------------------------|---|
| | | |
| | | |

Den Unterschied zwischen zwei Pfaden messen:

Jetzt brauchst du noch einen zweiten Pfad, mit dem du deine Zeichnung vergleichen kannst. Dazu machst du eine zweite Variable und nennst sie „Beispiel“.

Dieser Variable kannst du dann eine Zeichnung zuweisen. Vergiss nicht, dabei mithilfe von „resample“ die Anzahl der Punkte einzustellen:



Das kleine Programm - bestehend aus einem einzigen Befehl - kannst du einfach anklicken! Damit speicherst du eine auf 64 Punkte normalisierte Version deiner aktuellen Zeichnung als Beispiel.

Übrigens: Jede Variable, die du neu erstellst, wird auf der Bühne als kleines Fenster angezeigt. Dadurch kannst du mitverfolgen, was der Variable jeweils zugewiesen ist. Allerdings können diese Fenster auch Teile der Bühne verdecken. Deshalb gibt es neben den Variablen-Blöcken in der Palette jeweils einen Knopf mit einem Häkchen drauf. Damit kannst du die Anzeige für jede deiner Variablen aus- und einschalten. Probiere es gleich mal aus!



Nachdem du ein Beispiel gespeichert hast, kannst du jetzt verschiedene andere Formen zeichnen, und mit dem "Unterschied" Block messen, wie sehr sie sich von dem Beispiel unterscheiden. Achte darauf, dass beide Pfade, Beispiel und neue Zeichnung, die gleiche Anzahl von Punkten haben. Du kannst dir wieder ein Mini-Programm basteln, das diesen Vergleich für dich vornimmt.

Baue dieses Skript nach und teste es mit sehr unterschiedlichen und sehr ähnlichen Gesten:

Ähnlichere Gesten:

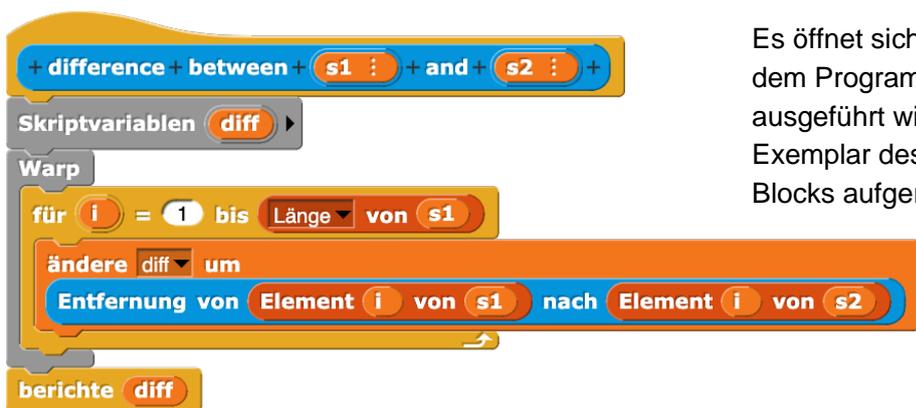


Sehr unterschiedliche Gesten:



Der Unterschied zwischen den Pfaden ist bei der zweiten Zeichnung fast zehnmal größer als bei der ersten.

Was bedeutet das und wie funktioniert das? Du kannst dir ansehen, wie der "Unterschied" Block aufgebaut ist und was er genau tut. Klicke dafür mit der rechten Maustaste auf den Block und wähle „Bearbeiten...“. Auf einem Touchscreen bleibe einfach lange mit dem Finger auf dem Block, um das Kontextmenü zu öffnen:



Es öffnet sich ein Blockeditor mit dem Programm, das immer dann ausgeführt wird, wenn ein Exemplar des "Unterschied" Blocks aufgerufen wird.

Sieh dir dieses Programm in Ruhe an. Versuche es nachzuvollziehen und beschreibe in deinen eigenen Worten, was es macht.

Beispiele für Gesten speichern

Damit dein Programm herausfinden kann, welchem Beispiel eine Zeichnung am nächsten kommt, braucht es nicht nur ein einziges Beispiel, sondern mehrere. Erstelle also nochmal eine neue Variable und nenne sie "Beispiele" (Plural). Danach kannst du die Variable mit dem Namen "Beispiel" (Singular) löschen, wenn du willst, du brauchst sie nicht mehr.

Die "Beispiele" Variable soll beliebig viele Pfade speichern können, deshalb initialisiert du sie als Liste. Das geht genauso, wie du es schon für die andere Variable mit dem Namen "Zeichen" getan hast:

setze Beispiele auf Liste ▶

Baue dieses Mini-Programm nach und führe es aus, indem du es einmal anklickst. Danach sieht deine Variable in ihrem Fenster auf der Bühne wie eine leere Liste aus:

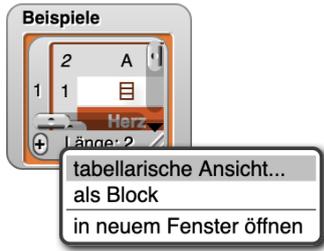


Jetzt kannst du darin Beispielgesten speichern und mit Namen versehen. Male ein Zeichen und speichere es mit dem folgenden Skript:

füge Liste resample Zeichen auf 64 Punkte Stern ◀▶ zu Beispiele hinzu

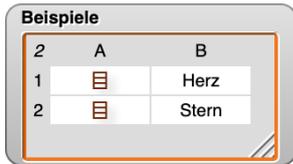
Dieser Teil des Programms macht eine neue Liste aus zwei Elementen, dem auf 64 Punkte normalisierten Pfad deiner aktuellen Zeichnung und einem Wort als Bezeichnung für den Pfad. Diese Liste wird den Beispielen hinzugefügt. Einer Liste weitere Listen hinzuzufügen kannst du schon, es ist das Gleiche, wie zu einem Pfad eine neue Liste aus x- und y-Koordinate hinzuzufügen.

Wenn du ein oder zwei Beispiele gezeichnet und gespeichert hast, wird deine Beispiel-Liste wahrscheinlich so angezeigt:



Du kannst die Darstellung verändern, indem du entweder das Projekt speicherst und neu öffnest oder das Wort "Länge" mit der rechten Maustaste anklickst bzw. auf einem Touchscreen mit dem Finger länger draufhältst und dann in Kontextmenü "tabellarische Ansicht" auswählst

Danach werden die Beispiele als Tabelle dargestellt:



Die Anzeige spielt aber für den Inhalt und die Funktion deiner Beispiel-Liste keine Rolle.

Speichere für die Gesten, für die du dir schon eine Animation überlegt hast, in der „Beispiele“-Liste einen Pfad und Namen ab.

Die ähnlichste Geste ermitteln

In diesem Abschnitt baust du deinen eigenen Block, der beliebige Gesten erkennen kann. Alle dafür erforderlichen Bausteine hast du schon. Der Block soll ein Reporterblock sein, also eine Funktion, mit zwei Eingaben:



Die erste Eingabe ist der Pfad, der zuletzt gezeichnet worden ist. Um ihn mit anderen Pfaden zu vergleichen, musst du ihn zuerst auf die gleiche Anzahl von Punkten bringen. Dazu benutzt du wieder "resample". Die zweite Eingabe ist die Liste von Beispielen. Das heißt, du wirst den Block später so verwenden:



Dieser "erkenne" Block soll ein Wort berichten, den Namen eines Beispiels, das der Zeichnung am ähnlichsten ist. Dieses Wort möchtest du an alle Objekte senden, um später ein Ereignis auszulösen:



Erstelle jetzt den Block, indem du in einer beliebigen Kategorie auf das Plus-Zeichen am oberen Ende klickst. Wähle die passende Form (oval, d.h. Funktion), Farbe (hellblau für Fühlen), Geltungsbereich ("für alle") und Namen.

Füge danach die beiden Eingaben hinzu, die du z.B. "Pfad" und "Muster" nennen, und als Typ "Liste" markieren kannst. Du kannst neue Eingabefelder für einen Block erstellen, indem du auf ein Pluszeichen an der Stelle klickst, an der du die Eingabe haben möchtest (in diesem Fall also zwischen „erkenne“ und „in“ und nach „in“:



Und jetzt nimm dir ein paar Minuten Zeit und überlege dir in deinen eigenen Worten, was passieren soll, wenn der Block aufgerufen wird und wie du das Beispiel ermitteln kannst, das einer Zeichnung am nächsten kommt!

Um den Block tatsächlich zu bauen, benötigst du die folgenden Bausteine:



Du kannst den „Skriptvariablen“ Block mit den schwarzen Pfeilen um neue Variablen erweitern.

Du kannst Skriptvariablen und Blockvariablen wie im „für jedes Element von“ Block umbenennen, indem du einmal auf den orangenen Variablenbereich klickst.

Zuerst bestimmst du diejenigen Variablen, die deine Funktion braucht:

- Unterschied von zwei Pfaden
- bisher kleinster Unterschied
- Beispiel mit bisher kleinstem Unterschied (Treffer)

Die Variable für den kleinsten Unterschied (“min”) initialisierst du mit dem Wert “unendlich”. Warum? Damit du auf jeden Fall mindestens einen Treffer findest, dessen Unterschied kleiner als der Anfangswert ist.

Zum Schluss wird das zweite Element des “Treffers” zurückgegeben. Warum das zweite Element, und nicht der ganze Treffer? Weil jedes Beispiel aus einem Pfad und einem Wort (Namen) besteht, und du nur am Wort interessiert bist.

Viel Spaß beim Puzzeln!

Jetzt gibt es nur noch eines zu tun: Verbinde das Skript, das die Geste erkennt, mit dem Zeichenprogramm, das den Malstift steuert.

Eine Geschichte erzählen

In diesem Teil sollen die erkannten Zeichnungen mit den Animationen verbunden werden, um eine Geschichte zu erzählen.

Auf Nachrichten reagieren

Dein Zeichenskript sendet eine Nachricht an alle Objekte, sobald eine Zeichnung erkannt wurde. Auf diese Nachricht können andere Skripte reagieren.



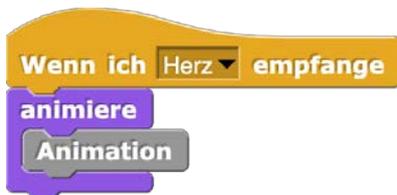
Dieser Startblock löst das Skript aus, mit dem er verbunden ist, wenn er die Nachricht empfängt, die im Eingabefeld angegeben ist.

Du kannst Nachrichten erstellen, auf die der Startblock reagieren soll, indem du auf den kleinen schwarzen Pfeil im Eingabefeld klickst und „Neu...“ auswählst. Gib eine Nachricht im Fenster ein und klicke „OK“.

Wähle als Nachricht ein Beispiel aus deiner „Beispiele“-Liste aus, z.B. „Herz“.



Verbinde den Startblock mit der passenden Animation:



Abschlussprojekt

Erstelle ein Abschlussprojekt, mit dem du eine kurze Geschichte illustrieren kannst. Zeichne Gesten, während du die Geschichte erzählst, und animiere diese, um deine Worte zu unterstreichen.

Übrigens kannst du mit den Nachrichten auch Skripte in anderen Objekten oder den Hintergrund steuern, um dein Projekt noch auszuschnücken.

Gestenerkennung Reflexion

Du hast ein Programm geschrieben, das funktioniert, prima! Jetzt ist eine gute Gelegenheit, darüber nachzudenken, wie es weitergeht und was dein Programm nicht kann und ggf. ungewollt tut.

Grenzen: Welche Fälle kann dein Programm nicht gut erkennen?

Viele Zeichen kann man auf die eine und andere Weise malen, so herum und andersherum. Zum Beispiel kann man den Buchstaben „S“ in einem Zug von rechts oben nach links unten zeichnen, oder umgekehrt vom links unten nach rechts oben. Wovon hängt es bei deinem Modell ab, ob das Programm eine Geste erkennt, die „andersherum“ gemalt wird?

Die Bedeutung eines Zeichens ist meistens unabhängig davon, in welcher Größe es gemalt wird. Sprüht man es riesig auf eine Wand bedeutet es das Gleiche, wie wenn man es winzig in die Ecke eines Zettels kritzelt. Wie geht dein Programm mit solchen individuellen Ausdrucksformen um?

- Überlege und teste verschiedene Beispiele selbstständig in deinem Programm.
- Setze dich jemandem gegenüber und schreibe ein Zeichen auf das Papier. Kannst du das Zeichen deines Gegenübers erkennen, auch wenn es „auf dem Kopf“ steht? Probiere das Gleiche mit eurem Gestenerkennungsprogramm aus! Funktioniert es immer noch? Was ist anders?
- Setzt euch beide an denselben Tisch vor dasselbe Blatt Papier, aber nicht gegenüber, sondern nebeneinander. Schreibt beide das gleiche Zeichen auf das Blatt Papier vor euch, aber jeweils in eine andere Ecke. Überlegt euch und probiert aus, ob euer Programm das Zeichen auch dann erkennt, wenn es „in der Ecke“ gemalt wird. Wovon hängt die Wahrscheinlichkeit ab, dass solche Zeichen erkannt werden können?

Vorurteile (Bias): Wie benachteiligt dein Programm Menschen und was kann man dagegen tun?

Diskutiere in der Gruppe, wie dein Programm Menschen benachteiligen könnte. Hast du zum Beispiel dein Programm auch mit anderen Menschen getestet? Auch mit Älteren oder Personen, die mit der anderen Hand schreiben? Entwickle Lösungsansätze, um das Programm fairer und inklusiver zu gestalten.

Musterlösungen

Gestenerkennung im Alltag:

Überlege, wo dir Gestenerkennung im Alltag begegnet:

- Digital: Handy entsperren, Mehrfinger-Gesten zur Computersteuerung (z.B. Pinch-Zoom), Steuerung von Spielen bei Kinect/Wii, Handschrifterkennung bei Digitalisierungsprogrammen z.B. auf Tablets
- Analog: z.B. Verständigung beim Tauchen, Gebärdensprache -> Zwischenmenschliche Kommunikation

Diskutiere, was die Vorteile, was die Nachteile von Gesten zur Steuerung bestimmter Anwendungen sein können.

- Nachteile:
 - Müssen korrekt erkannt werden, um sie weiterverarbeiten zu können
 - Manchmal schwierig für Menschen mit motorischen Einschränkungen
- Vorteile:
 - Geht oft schneller
 - International anwendbar, da auf allen Systemen gleich.
 - Geräteübergreifend anwendbar (z.B. funktioniert Pinch-Zoom auf Handys und Tablets mit Touchscreen sowie auf Laptops mit Touchpad)

Ein Zeichenprogramm erstellen:



The image shows two Snap! scripts for a drawing program. The first script is triggered by a 'gedrückt' (pressed) event and sends a 'zeichnen' (draw) message to all objects. The second script is triggered by a 'zeichnen' (draw) message and performs a sequence of actions: 'wische' (erase), a loop 'wiederhole bis nicht Maustaste gedrückt?' (repeat until mouse button not pressed) containing 'gehe zu Mauszeiger' (go to mouse pointer), 'Stift runter' (pen down), and 'Stift hoch' (pen up).

Musterlösung in Snap!: <https://tinyurl.com/Zeichenprogramm>

Zeichen animieren:

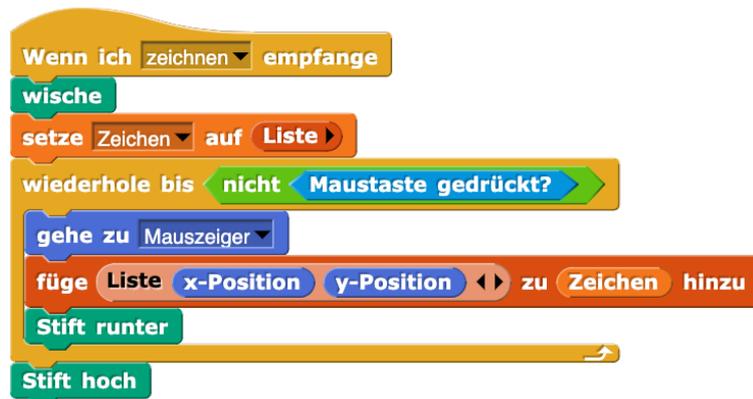
Beispiel für Herzschlag:



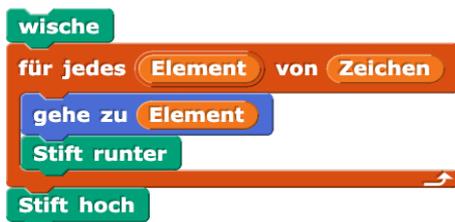
Beispiele in Snap!: <https://tinyurl.com/GestenAnimieren>

Pfade aufzeichnen und wieder abspielen:

Das angepasste Zeichenprogramm, das Pfade speichern kann:



Ein Wiedergabeprogramm erstellen:



Musterlösung Pfade aufzeichnen und abspielen in Snap!:

<https://tinyurl.com/GrosseGestenPfade>

Pfade miteinander vergleichen:

Zeichen resampeln: <https://tinyurl.com/GrosseGestenResample>

| Zeichen | Zeichen mit 10 Punkten | Anzahl an Punkten, bei der Zeichen noch erkennbar ist |
|---------|---|---|
| Blume |  | 24 |
| Herz |  | 10-11 |

Unterschiede zwischen Pfaden messen:

Der "Unterschied" Block ermittelt die Summe der Entfernungen aller korrespondierenden Punkte zweier Pfade. Dafür werden alle Punkte eines Pfades durchlaufen, und jeweils die Entfernung zum entsprechenden Punkt an der gleichen Stelle des anderen Pfades ermittelt. Alle Entfernungen werden in einer lokalen Variable zusammengezählt und die Summe als Ergebnis zurückgegeben. Das ganze geschieht in "Warp-" Geschwindigkeit, also so schnell wie möglich in einem einzigen Bildschirmzyklus bzw. Frame.

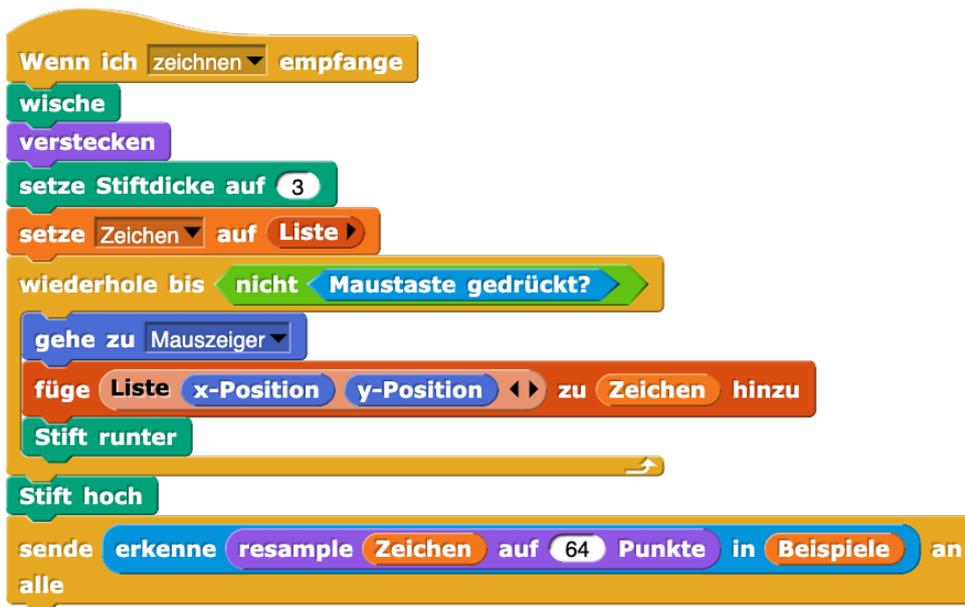
Nearest Neighbour Algorithmus in eigenen Worten beschreiben:

Bestimme für jedes Beispiel den Unterschied des Pfades zum Pfad der aktuellen Zeichnung -> Nimm das Beispiel mit dem kleinsten Unterschied -> Berichte seinen Namen.

Erkenne Block:



Fertiges Zeichenskript:



Gesamtlösung mit 2 Beispielen:

<https://tinyurl.com/GrosseGestenLoesung>

Gestenerkennung Reflexion

Grenzen: Welche Fälle kann dein Programm nicht gut erkennen?:

Gesten, die sich in einem der folgenden Punkte stark von den in der Beispielliste gespeicherten Gesten unterscheiden:

- Größe (Skalierung)
- Drehung (Rotation)
- Positionierung (Translation)

Vorurteile (Bias): Wie benachteiligt dein Programm Menschen und was kann man dagegen tun?:

Das Programm benachteiligt Menschen, die motorische Schwierigkeiten haben, eine andere Händigkeit aufweisen als die Person, die das Programm trainiert hat.

Um das zu verhindern, können mehr und diversere Beispiele eingespeichert werden, es könnten Zusatzinformationen zur Ausführung einer Geste im Programm integriert werden, bspw. Das Anzeigen eines Felds, das beim Zeichnen der Geste möglichst ausgefüllt werden soll, um starke Abweichungen in der Größe und Positionierung zu vermeiden. Oder ein Pfeil, der anzeigt, wo bei eindeutigen Gesten oben oder unten sein soll.

Dies könnte auch automatisiert geschehen, wie im original \$1-Gesture Recognizer. Dieser enthält die weiteren Funktionen „translate“, „rotate“ und „scale“ (also translatiere, rotiere, skaliere), die Abweichungen in Größe, Drehung und Position ausgleichen sollte.

Blöcke in Snap!

1.1 Block-Glossar

Hier werden Blöcke, die in der Programmiersprache Scratch nicht vorhanden sind, kurz erläutert. Selbsterstellte Blöcke für das Modul werden im nächsten Abschnitt detaillierter besprochen.

| Block | Erläuterung |
|--------------------------|---|
| <p>gehe zu _</p> | <p>Bewegt das Objekt an den Ort, der im Eingabefeld ausgewählt ist (zufällige Position auf der Bühne, Mitte der Bühne, Mausposition) oder direkt zu einem Punkt, der als Liste aus gewünschter x- und y-Koordinate eingegeben werden kann.</p> |
| <p>Malspuren</p> | <p>Berichtet die aktuell auf der Bühne gezeichneten Malspuren als Bild. Kann z.B. als Eingabe für den „ziehe Kostüm _“ Block dienen, um die aktuellen Malspuren als Kostüm zu verwenden.</p> |
| <p>_ von _</p> | <p>Nimmt als erste Eingabe eine mathematische Operation, die aus dem Dropdown-Menü ausgewählt werden kann, als zweite eine Zahl, auf die diese Operation angewandt wird, und berichtet das Ergebnis.</p> |
| <p>Warp _</p> | <p>Die Blöcke im c-förmigen Eingabefeld werden mit maximaler Geschwindigkeit ausgeführt, ohne Auswirkungen und Ausführen anderer Skripte zu berücksichtigen.</p> |
| <p>für i von _ bis _</p> | <p>Schleife mit einer internen Variablen i sowie einem Start- und Endwert. Beim Aufruf der Schleife wird i auf den Startwert gesetzt und bei jeder Iteration der Blöcke im c-förmigen Eingabefeld automatisch um 1 erhöht, bis der Endwert erreicht ist. Danach stoppt die Schleife. i kann auch innerhalb der Schleife verwendet werden, indem die orange Variable angeklickt und weggezogen wird.</p> |
| <p>führe _ aus</p> | <p>Nimmt einen Befehlsblock (oder Skript) als Eingabe und führt diesen aus. Insbesondere nützlich, um c-förmige Blöcke wie „animiere“ selbst zu schreiben. Die Blöcke, die sich im c-förmigen Eingabefeld des Blockes befinden, werden in der Blockdefinition in einer Variablen gespeichert. Diese kann mithilfe des „führe aus“ Blocks die gespeicherten Blöcke an der passenden Stelle ausführen. Kann durch Klicken auf den Pfeil um Eingaben erweitert werden, die bestimmen, mit welchen Eingabewerten der Block/das Skript ausgeführt werden soll.</p> |
| <p>berichte _</p> | <p>Gibt den Wert im Eingabefeld als Sprechblase aus. Wird benötigt, um benutzerdefinierte Reporter/Funktionen (Blöcke, die Werte zurückgeben) zu bauen.</p> |
| <p>Skriptvariablen a</p> | <p>Erzeugt sogenannte Skriptvariablen, die nur innerhalb des Skriptes, in dem sie definiert wurden, gültig sind. Durch Klicken</p> |

| | |
|--|---|
|  | <p>auf den Pfeil nach rechts kann der Block um zusätzliche Skriptvariablen erweitert werden. Durch Klicken auf die Variable kann diese umbenannt werden.</p> |
| <p>_ von _</p>   | <p>Gibt die im ersten Eingabefeld ausgewählte Eigenschaft der Liste im zweiten Eingabefeld zurück. Hier beispielsweise die Länge, also Anzahl der Elemente, der eingegebenen Liste.</p> |
| <p>Liste _</p>   | <p>Berichtet eine Liste mit den in den Eingabefeldern eingegebenen Elementen. Die Zahl der Elemente kann durch die Pfeile erweitert oder verringert werden. Bevor zu einer Variablen Listen-Elemente hinzugefügt werden können, muss diese als Liste initialisiert werden. Dafür kann die Variable auf eine leere Liste gesetzt werden.</p> |
| <p>für jedes _ von _</p>  | <p>Schleife, die über eine Liste (erstes Eingabefeld) iteriert. Für jedes Element der Liste werden die Blöcke im c-förmigen zweiten Eingabefeld ausgeführt. Das jeweils aktuelle Element ist in der internen Variablen gespeichert und kann durch Klicken und Ziehen auch in weiteren Blöcken innerhalb des Skripts verwendet werden.</p> |
| <p>Element _ von _</p>  | <p>Gibt das Element aus der Liste im zweiten Eingabefeld zurück, dessen Index der im ersten Eingabefeld genannten Zahl entspricht. Im Dropdown-Menü des ersten Eingabefeldes können direkt weitere Möglichkeiten wie „zufälliges“ oder „letztes“ ausgewählt werden.</p> |
| <p>füge _ zu _ hinzu</p>  | <p>Fügt ein Element mit dem Inhalt des ersten Eingabefelds zur Liste im zweiten Eingabefeld hinzu. Beispielsweise können so Elemente zu in Variablen gespeicherten Listen hinzugefügt werden.</p> |
| <p>kombiniere die Elemente von _ mit _</p>  | <p>Kombiniert die Werte der Liste im ersten Eingabefeld mithilfe der Funktion im zweiten Eingabefeld zu einem einzigen Wert. Die Funktion im zweiten Eingabefeld muss mindestens zwei Parameter haben.</p> <p>Beispiel: Summe aller Werte einer Liste</p>  <p>Beispiel: Wort aus Buchstabenliste verbinden</p>  |

1.2 Benutzerdefinierte Blöcke

1.2.1 Benutzerdefinierte Blöcke erstellen

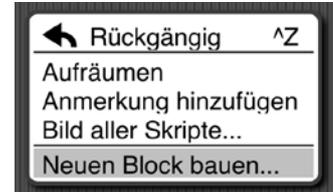
Benutzerdefinierte Blöcke können in Snap! auf verschiedene Weise erzeugt werden. Um einen Block zu erstellen und den Blockeditor zu öffnen, gibt es in der Palette am Anfang jeder Kategorie einen Knopf mit einem Pluszeichen, am Ende jeder Kategorie einen Knopf „Neuer Block“. Auch über Rechtsklick ->

„Neuen Block bauen...“ im

Skriptbereich kann der Blockeditor geöffnet werden.



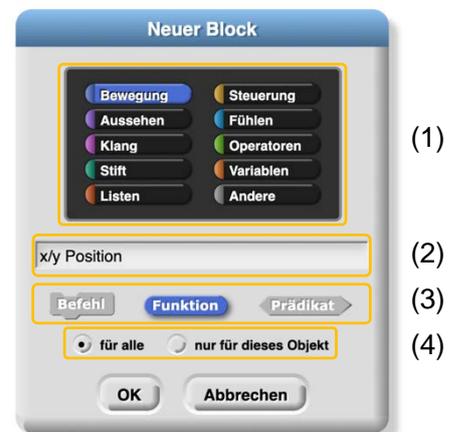
in der Palette



Im Skriptbereich

Im jetzt geöffneten Fenster (Neuer Block) kann eine Kategorie für den neuen Block gesetzt (1), ein Name gegeben (2), der Blocktyp (Befehl, Funktion oder Prädikat) und damit die Blockform gewählt (3) und die Verfügbarkeit des Blocks im Projekt (4) festgelegt werden. Blöcke „nur für dieses Objekt“ sind nur innerhalb des Objekts verfügbar, in dem sie definiert wurden. Sie sind in der Palette am Positionsmarker vor dem Blocknamen erkennbar.

Sind diese Informationen definiert, kann über „OK“ der Blockeditor geöffnet werden.



Im Blockeditor befindet sich ein Startblock mit dem Namen des neuen benutzerdefinierten Blocks, der sogenannte Blockprototyp. Darunter wird die Blockdefinition programmiert, also das Skript, das ausgeführt wird, wenn der Block aufgerufen wird.

Klicke auf „OK“, um den Block zu speichern und den Blockeditor zu schließen oder „Anwenden“, um den Block zu speichern und den Editor weiterhin geöffnet zu lassen.

Benutzerdefinierte Blöcke befinden sich am Ende der jeweiligen Kategorie, der sie zugewiesen sind.

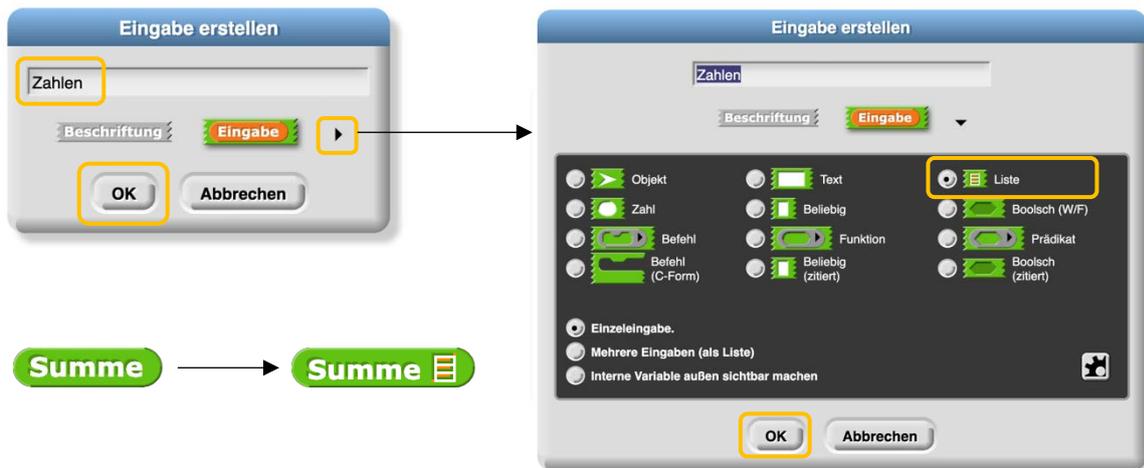
Blockprototyp
Blockdefinition





Wie Basisblöcke können auch benutzerdefinierte Blöcke Eingaben (Parameter) haben. Um ein Eingabefeld zu einem Block hinzuzufügen muss man auf eines der Pluszeichen im Blockprototypen klicken.

Dadurch öffnet sich ein Dialogfenster, in dem ausgewählt werden kann, ob die Blockbeschriftung erweitert oder ein Eingabefeld hinzugefügt werden soll und welchen Namen diese haben soll. Mit Klick auf den kleinen schwarzen Pfeil gelangt man zum erweiterten Eingabe-Dialog, wo beispielsweise spezifiziert werden kann, welcher Datentyp als Eingabe erwartet wird (in diesem Fall Liste), oder ob ein Standardwert eingegeben werden soll. Wird kein Eingabetyp angegeben, wird dieser auf „Beliebig“ gesetzt – das Eingabefeld erscheint als weißes Rechteck. Mit Klick auf „OK“ wird die Eingabe hinzugefügt.



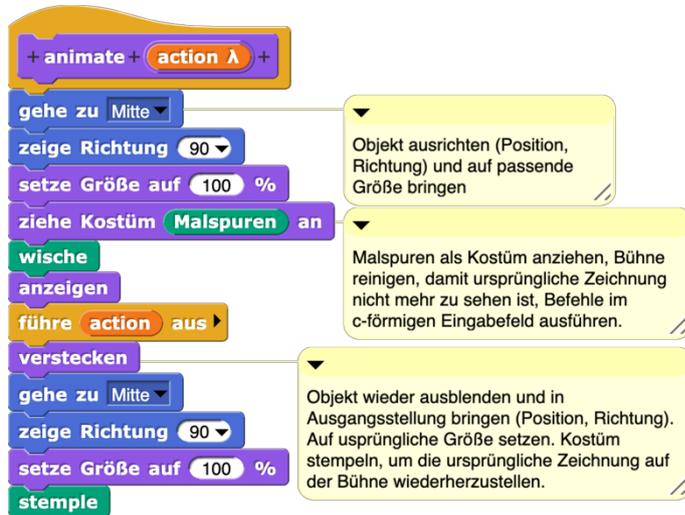
Wird der Block gespeichert, ist der Parameter im Block nun als Eingabefeld sichtbar. Im Blockeditor ist er als orange Variable angezeigt, in der die Werte aus dem Eingabefeld gespeichert werden. Diese orange Variable kann mit Klicken und Ziehen aus dem Blockprototypen „abgezogen“ und innerhalb der Blockdefinition verwendet werden.



Blockdefinitionen können über Rechtsklick -> „Bearbeiten...“ wieder geöffnet werden.

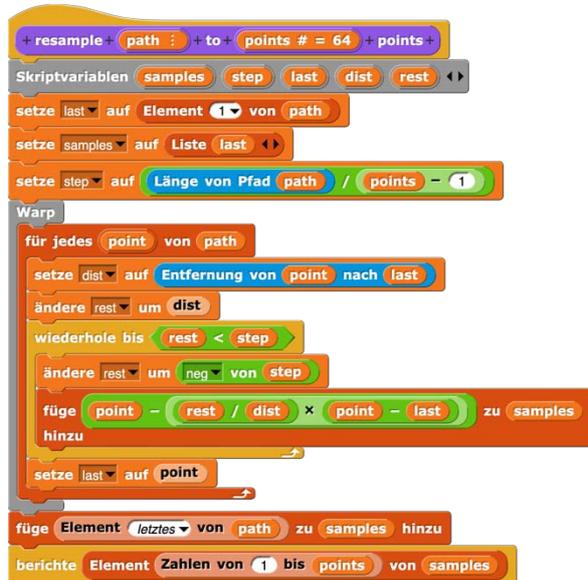
1.2.2 Benutzerdefinierte Blöcke in diesem Modul

- animiere _ 



Der „animiere“ Block ist dafür verantwortlich, dass das Objekt die aktuelle Zeichnung (Malspuren) als Kostüm anzieht und danach die Bühne löscht („wische“). Im Anschluss werden von diesem Objekt die im c-förmigen Eingabefeld angegebenen Befehle ausgeführt. Zum Schluss wird alles wieder in den Ausgangszustand zurückversetzt, d.h. das Objekt wird versteckt und zur Ausgangsposition bewegt. Zudem wird das Kostüm (die ursprünglichen Malspuren) wieder auf die Bühne gestempelt.

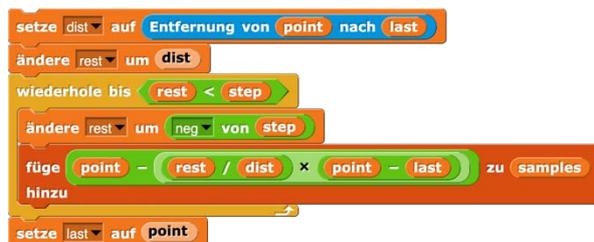
- resample _ auf _ Punkte 



Der „resample“ Block verteilt den Pfad im ersten Eingabefeld („path“) auf die Anzahl an Punkten im zweiten Eingabefeld („points“) und gibt eine neue Liste mit Koordinaten dafür aus („samples“).

Dafür werden zuerst alle benötigten Variablen initialisiert. Der erste Punkt des Pfades wird der Ergebnisliste („samples“) hinzugefügt. Der „step“, die Länge des Segments, das zwischen zwei Punkten liegen soll, wird gesetzt. Dieser errechnet sich aus der Gesamtlänge des Pfades dividiert durch die um eins verringerten Punkte.

Danach wird für jeden Punkt im ursprünglichen Pfad folgendes Skript durchgeführt:

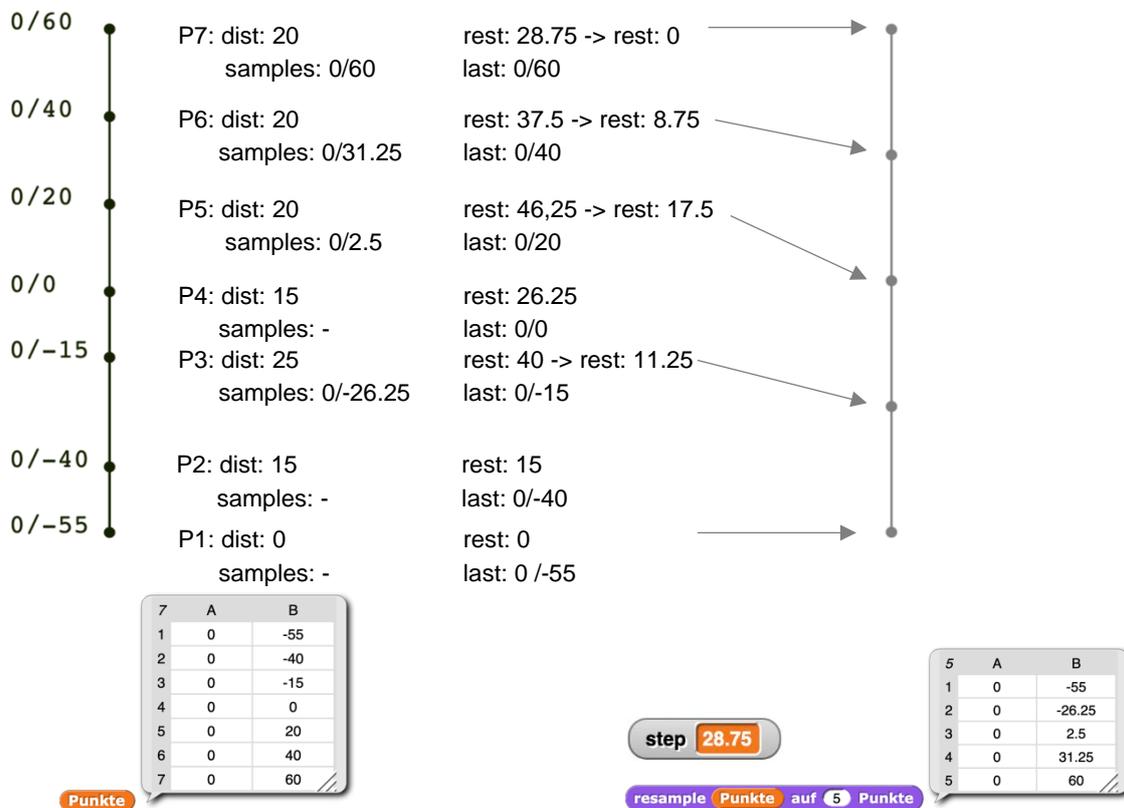


Die Variable „dist“ wird auf die Entfernung zwischen aktuellem Punkt („point“) und dem letzten Punkt, für den das Skript durchgeführt wurde, („last“) gesetzt. Beim ersten Mal ist dies der erste Punkt des Pfades. Danach wird die Variable „rest“ um diese Entfernung verändert (d.h. „dist“ wird zum aktuellen Wert von „rest“ addiert). Nun folgt eine „wiederhole bis“-Schleife, die ausgeführt wird, bis der „rest“ kleiner als der „step“ ist. Das bedeutet, solange „rest“ kleiner „step“ ist, also die Distanz, die

bisher verrechnet wurde („rest“) geringer ist als die Länge eines Segments („step“), wird dieser Teil einfach übersprungen. Danach wird der aktuelle Punkt auf „last“ gesetzt, also der letzte Punkt, der bearbeitet wurde, und die Schleife beginnt von vorn mit dem nächsten Punkt des Pfades.

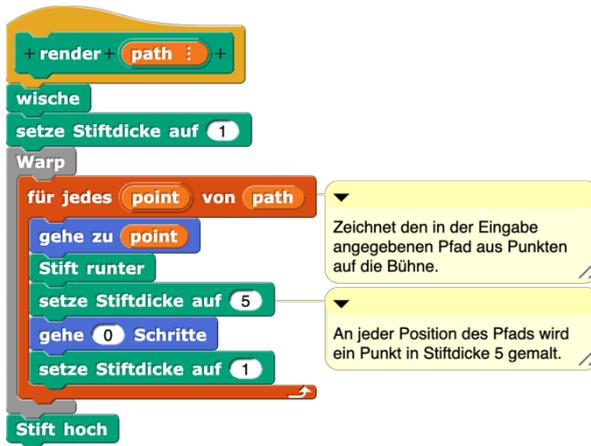
Gelangt man nun an einen Punkt, an dem „rest“ größer als die gewünschte Länge des Segments („step“) ist, wird die „wiederhole bis“ Schleife durchlaufen. Dabei wird zuerst berechnet, wie weit „rest“ bereits über den gewünschten „step“ hinausgeht (indem „step“ von „rest“ abgezogen wird). Danach werden die Koordinaten berechnet, an denen der nächste Punkt von „samples“ gesetzt wird. Die x- und y-Koordinate des neuen Punktes errechnen sich dadurch, dass von der jeweiligen Koordinate des aktuellen Punktes ein Wert abgezogen wird, der berücksichtigt, um welchen Anteil („rest“ / „dist“) die Strecke zwischen dem aktuellen Punkt (point) und dem letzten Punkt (last) bereits überschritten ist. Aufgrund der Hyperblocks-Eigenschaft können auch hier x- und y-Koordinate gleichzeitig verrechnet werden.

Im folgenden Beispiel werden die Schritte angezeigt, die durchlaufen werden, um einen Pfad aus sieben ungleichmäßig verteilten Punkten einen Pfad aus fünf gleichmäßig verteilten Punkten zu berechnen.



Hier ist beispielhaft aufgezeigt, welche Schritte für P5 (0/20) innerhalb der „wiederhole bis“ Schleife durchlaufen werden.





Der „zeichne“ Block zeichnet einen Pfad aus den in der Eingabe angegebenen Punkten (Liste mit x- und y- Koordinaten) auf die Bühne, indem er für jeden Punkt des Pfades an die dementsprechende Stelle geht. Dies erzeugt eine Linie vom vorherigen zum aktuellen Punkt, da der Stift währenddessen unten ist. Zusätzlich wird an der jeweiligen Stelle noch ein Punkt in Stiftdicke 5 gemalt.

- Konstante

Der „Konstante“ Block sucht den Index der im Dropdown-Menü der Eingabe



ausgewählten Konstante in der Liste $\infty \pi$. Der ermittelte Index wird als Eingabe für den „Element _ von“ Block verwendet, um das passende Element aus der Liste 1/0 (infinity) und 3.141... auszugeben.

- Entfernung von nach

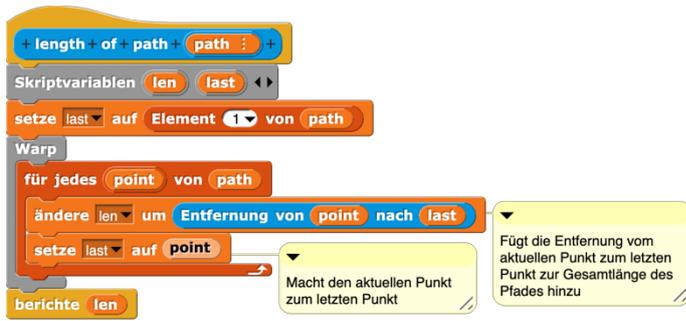


Der „Entfernung von _ nach _“ Block ermittelt die Entfernung zwischen zwei Punkten p1 und p2 (bestehend aus einer Liste aus x- und y-Koordinate) mithilfe des Satz des Pythagoras.



Der Block macht sich dabei die Vorteile von Hyperblocks zunutze, bei denen (mathematische) Operationen auf gesamte Listen angewandt werden können. So führt beispielsweise die Operation $p2 - p1$ zu einer Liste mit zwei Elementen. Das erste Element ist die Differenz der x-, das zweite Element die der y-Koordinate von p2 und p1.

- Länge von Pfad



Der „Länge von Pfad“ Block berichtet die Länge des im Eingabefeld angegebenen Pfades. Dafür wird für jeden Punkt des Pfades die Entfernung zum vorherigen Punkt gemessen (beim ersten Punkt ist diese 0). Diese Entfernung wird jeweils zum gespeicherten Wert in der Variablen „len“ hinzugefügt, der zum Schluss ausgegeben wird.

- Unterschied zwischen _ und _

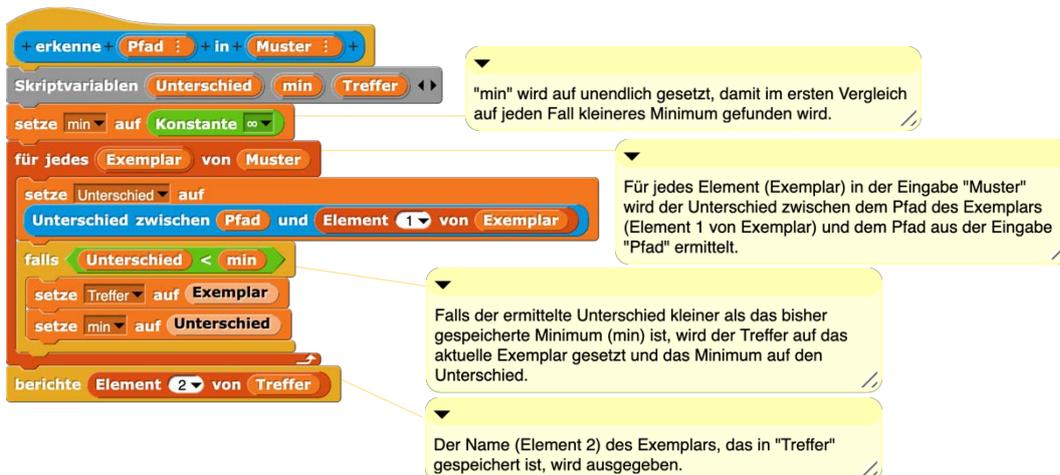
Unterschied zwischen und



Der „Unterschied zwischen“ Block vergleicht die Punkte zweier Pfade (s1 und s2) miteinander und berichtet die Summe der Entfernung jedes Punktes des ersten Pfades zum jeweiligen Punkt im zweiten Pfad.

- erkenne _ in _

erkenne in

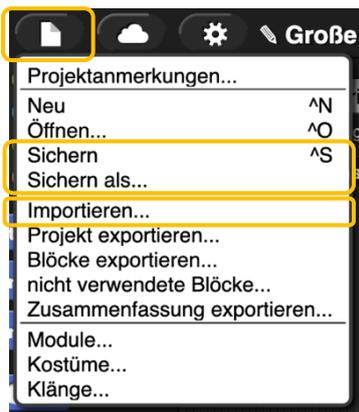


Der „erkenne _ in _“ Block vergleicht einen Pfad (Eingabe Pfad) mit allen Pfaden, die in der Eingabe Muster angegeben wurden. Jeder Eintrag von Muster besteht aus einem Pfad (Element 1 des Eintrags) und einem Namen (Element 2 des Eintrags). Zu Beginn wird die Variable „min“ mit der Konstante ∞ initialisiert, um sicherzustellen, dass der Unterschied zwischen Pfad und Pfad des ersten Musters auf jeden Fall kleiner ist. Danach wird der Pfad mit allen Pfaden der Elemente in der Eingabe Muster verglichen. Ist der Unterschied der Pfade kleiner als der Wert, der gerade in „min“ gespeichert ist, wird „min“ auf den aktuellen Unterschied gesetzt und das aktuelle Exemplar der Muster-Liste wird als „Treffer“ gespeichert. Wurde dieser Vergleich für alle Elemente von Muster durchgeführt, wird der Name (Element 2) des Exemplars ausgegeben, das aktuell als Treffer gespeichert ist.

Projekte speichern und teilen

Lokal auf dem Computer

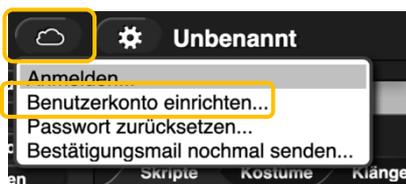
Projekte können lokal auf dem Computer als Dateien im XML-Format gespeichert werden. Klickt man im Dateimenü auf „Sichern als...“, öffnet sich ein Dialogfenster, in dem der Speicherort (Computer) und Name (Große Gesten) des Projekts gewählt werden können. „Sichern“ speichert das Projekt im Downloads Ordner des Browsers. Über Dateimenü -> „Importieren...“ oder „Öffnen...“ können lokal gespeicherte Projekte wieder in Snap! geladen werden.



In der Snap! Cloud

Um Projekte in der Snap! Cloud speichern zu können, muss ein Konto angelegt werden. Dies ist auf der Snap! Website oben rechts über „Sign up“ möglich. Alternativ kann ein Konto auch direkt aus dem Snap! Editor erstellt werden.

Klickt man auf das Cloud-Menü und wählt „Benutzerkonto einrichten...“, öffnet sich der Sign-up Dialog, in den die passenden Daten eingetragen werden können.



Das Geburtsdatum wird erfragt, da unter 16-Jährige sich mit der E-Mail-Adresse einer* Erziehungsberechtigten anmelden müssen.

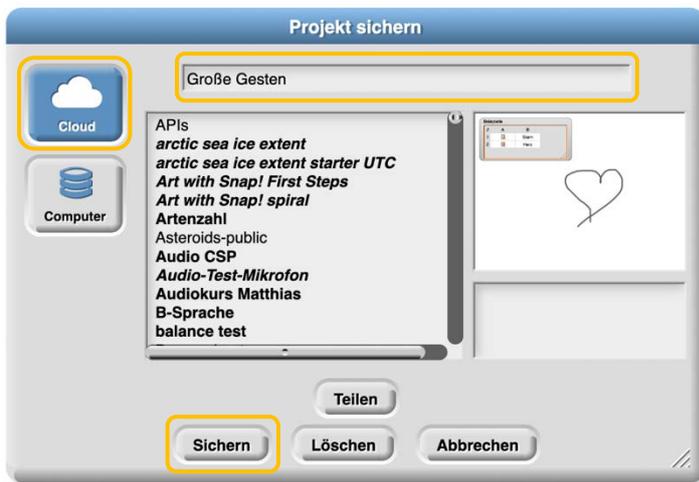
Die Nutzungsbedingungen und die Privatsphäre-Vereinbarung können hier nachgelesen werden (Englisch):

<https://snap.berkeley.edu/tos>
<https://snap.berkeley.edu/privacy>

Das Snap! Cloud Konto wird durch Klicken auf „OK“ angelegt und muss nun noch aktiviert werden. Das funktioniert entweder über den Aktivierungslink, der an die angegebene E-Mail-Adresse versandt wurde, oder indem ein Projekt in diesem Konto gespeichert wird.

Weitere Funktionalitäten zur Verwaltung des Kontos, z.B. Zurücksetzen des Passworts, sind ebenfalls von der Snap! Webseite oder über das Cloud Menü aus dem Editor möglich.

Momentan werden Gruppen-Anmeldungen nicht unterstützt, d.h. Konten müssen für alle SuS einzeln erstellt werden.

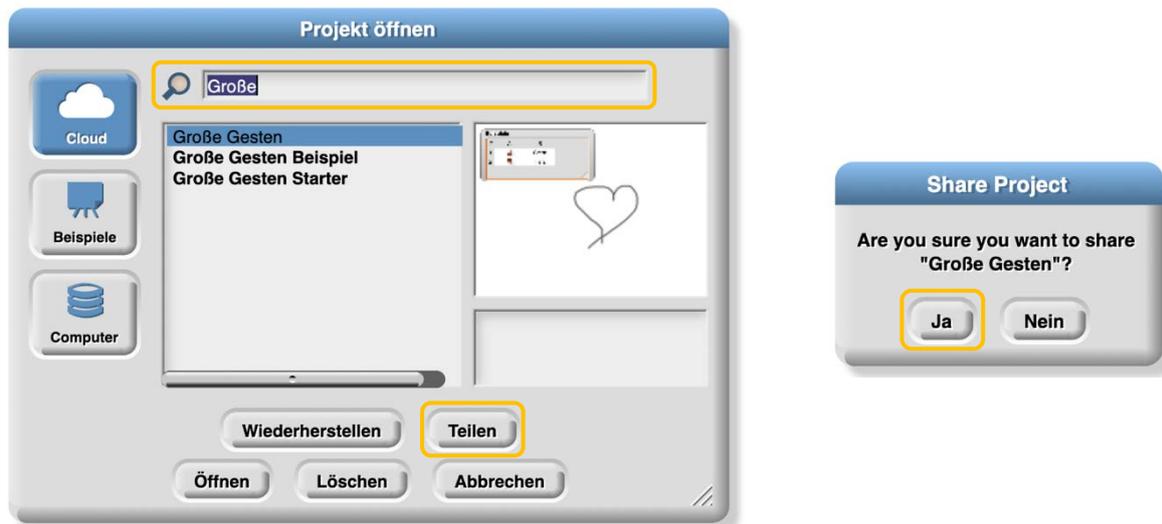


Das Sichern der Projekte verläuft wie beim lokalen Speichern. Über Dateimenü -> „Sichern als...“ öffnet sich ein Dialogfenster, in dem der Speicherort (Cloud) gewählt und ein Name (Große Gesten) vergeben werden kann. Das Fenster zeigt außerdem alle in diesem Account gespeicherten Projekte an.

Über Dateimenü „Öffnen...“ kann ein Dialogfenster geöffnet werden, das alle in der Cloud gespeicherten Projekte des aktuellen Kontos anzeigt. Wird ein Projekt ausgewählt und auf „Öffnen“ geklickt, wird dieses im Editor geladen.

Projekte teilen

Für Projekte, die in der Cloud gespeichert sind, kann ein teilbarer Link erzeugt werden. Dies kann ebenfalls auf der jeweiligen Projektseite auf der Snap! Website oder aus dem Snap! Editor erledigt werden. Über Dateimenü -> „Öffnen...“ wird ein Fenster geöffnet, in dem nach dem passenden Projekt gesucht werden kann. Um das Projekt zu teilen, muss es ausgewählt sein, und der Knopf „Teilen“ gedrückt werden. Wird im folgenden Dialog durch Klicken auf „Ja“ bestätigt, dass das Projekt geteilt werden soll, verändert sich die Adresse in der URL-Leiste zu einem eindeutigen Link, der den Projekt- und Kontonamen enthält. Dieser Link und damit das Projekt kann nun mit anderen geteilt werden. Geteilte Projekte kann man daran erkennen, dass sie in der Liste der Projekte fett geschrieben sind oder auf ihrer Projektseite ein Link-Symbol zu sehen ist.



Zudem gibt es die Möglichkeit, geteilte Projekte zu veröffentlichen. Das bedeutet, dass das Projekt nicht nur einen eindeutigen Link hat, sondern außerdem auf der Snap! Startseite unter „Neueste Projekte“ angezeigt wird und über die Suche auf der Snap! Webseite von anderen gefunden werden kann. Das funktioniert genau wie das Teilen von Projekten über Dateimenü -> „Öffnen“.

Natürlich kann das Teilen und Veröffentlichen von Projekten auch wieder rückgängig gemacht werden, indem die dementsprechenden Knöpfe aus dem „Öffnen...“ Dialogfenster ausgewählt werden.