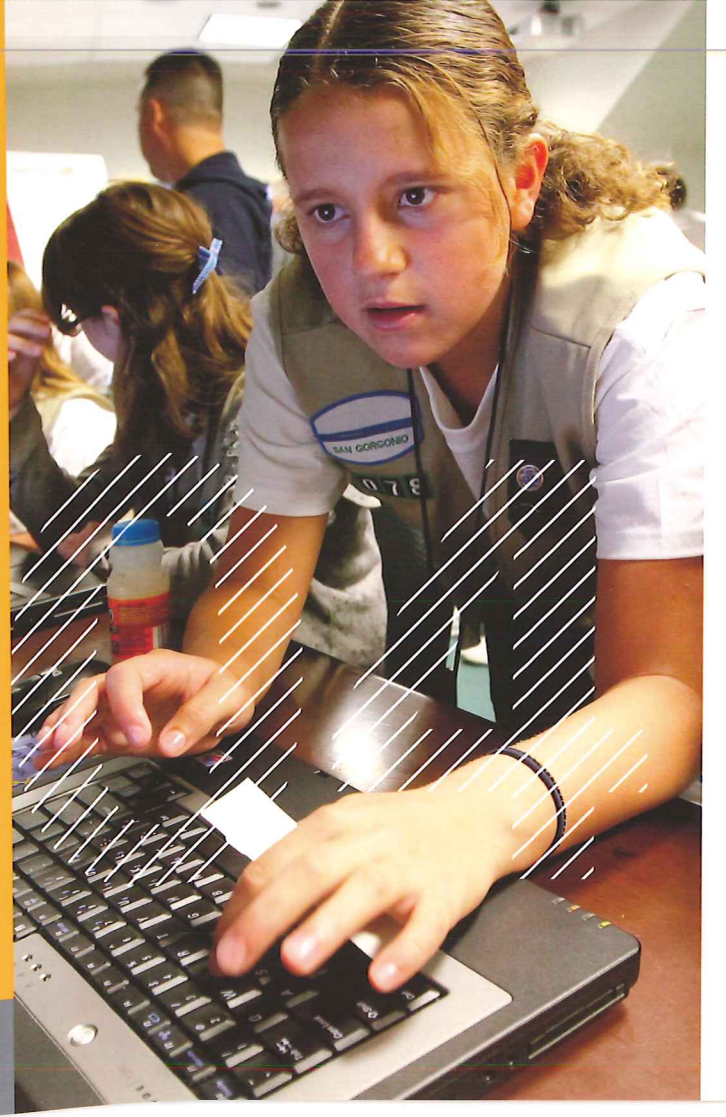
SCHRIFTENREIHE • JANUAR 2015

HASLERSTIFTUNG

COMPUTATIONAL THINKING IN DER LEHRERBILDUNG

ALEXANDER REPENNING



INHALT

COMPUTATIONAL THINKING IN DER LEHRERBILDUNG

Ziel dieser Broschüre ist es, den Begriff *Computational Thinking* in der Lehrerbildung in einer praxisorientierten Weise einzuführen. Obwohl viele Kinder eine hohe Affinität zur Computertechnologie haben und Computerspiele konsumieren oder mit ihren Freunden über soziale Webseiten kommunizieren, sind sie im Allgemeinen verblüffend wenig am Programmieren interessiert. Viele Schüler und Schülerinnen beschreiben Informatik mit den vernichtenden Worten: «Programmieren ist schwierig und langweilig.»

Der Autor dieser Schrift hat vor etlichen Jahren an der Universität Boulder, Colorado, die Methode des *Scalable Game Design* entwickelt, um Kinder und Jugendliche an die Informatik heranzuführen. *Scalable Game Design* beinhaltet eine Kombination aus pädagogischen Ansätzen, Werkzeugen und Aktivitäten, mit denen Lehrpersonen informatische Bildung auf motivierende Art und Weise mit ihren Schülerinnen und Schülern erarbeiten, um sie gezielt mit der Denkweise des *Computational Thinking* vertraut zu machen. Programmieren soll einfach und spannend werden!

WIESO COMPUTATIONAL THINKING?

Was genau ist *Computational Thinking* und wer braucht es? Wenn es um informatische Bildung geht, scheiden sich die Geister, denn vielen ist nicht klar, was dieser Begriff genau beinhaltet. Die Einen denken, informatische Bildung bedeute Tastaturschreiben oder das Anwenden von Textverarbeitungs- oder Gestaltungsprogrammen. Andere weisen der informatischen Bildung einen *Hardcore*-Charakter zu, also eine sehr programmierfokussierte Ausbildung, welche die nächste Generation von «Super-Programmierern» und «Mega-Firmengründern» wie Bill Gates (Microsoft) oder Mark Zuckerberg (Facebook) hervorbringe.

Nun sieht der Lehrplan 21 vor, dass informatische Bildung schon in der Primarstufe eingeführt wird, und die Gemüter erhitzen sich: Was soll unterrichtet werden und wie können die neuen Inhalte mit bereits vollen Stundenplänen in die Lehrpraxis umgesetzt werden?

Was, wann und ob

Statistiken zur Arbeitssituation deuten darauf hin, dass die Schweiz tatsächlich Informatikerinnen und Informatiker braucht. Bedeutet dies aber zwangsläufig, dass Schulen *Hardcore*-Informatik im obigen Sinne unterrichten müssen? Die Forschung zeigt, dass auch Länder wie die USA, die schon mehr Erfahrung mit informatischer Bildung gemacht haben, nur mühsam traditionelles Programmieren erfolgreich in den Unterricht integrieren konnten. Wie so oft liegt die Lösung in einem Kompromiss.

Computational Thinking ist ein Ansatz, der bewusst auf Konzepte und Problemlösungsstrategien allgemeiner Relevanz fokussiert. Es geht unter anderem darum, den Zusammenhang zwischen sequenziellen und parallelen Prozessen zu verstehen. Das tönt zwar kompliziert, ist es aber nicht. Ein konkretes Beispiel ist die Grossmutter, die einen Kuchen backen will. Sie weiss, dass sie den Zuckerguss schon vorbereiten kann, während der Kuchen im Ofen ist.

Was, wann und ob, das ist Computational Thinking. Die Inhalte von Computational Thinking sind nicht nur für die Informatik relevant, sondern auch für andere Wissenschaften wie Mathematik und Naturwissenschaften oder auch Sprachen, Kunst und Design. Wer zum Beispiel in Naturwissenschaften durch Programmieren die Simulation eines Ökosystems eigenständig erstellt, erfährt Wissenschaft wie ein Wissenschaftler und lernt sie nicht einfach auswendig. Die Schweiz braucht Denker – *Computational Thinkers* – und keine «Memorysticks».

Erfahrung aus den USA

Das tönt alles sehr akademisch. Kann das wirklich klappen? In den USA wurde vor vielen Jahren begonnen, systematisch zu untersuchen, wie man Computational Thinking in Schulen fördern könnte. Mit den Inhalten von *Scalable Game Design* beginnen Schülerinnen und Schüler inzwischen schon in der dritten Klasse, eigenständig Videospiele zu bauen, um Computational Thinking zu entwickeln. Sie spielen nicht, sondern entwerfen und lernen damit Computational Thinking – durch Programmieren. Das Verblüffende ist nicht nur die unglaubliche Motivation, sondern auch der Schwierigkeitsgrad, an den sich die Schülerinnen und Schüler freiwillig herantrauen. So wollen beispielsweise 10-Jährige Diffusionsgleichungen begreifen, die sie benötigen, um künstliche Intelligenz in ihrem Spiel einzubauen. Medizin muss nicht bitter sein. Mit der richtigen Motivation kann Lernen Spass machen.

In der grössten Studie dieser Art in den USA hat sich nicht nur gezeigt, dass Knaben und Mädchen gleichermassen an diesen Konzepten interessiert sind und diese umsetzen können, sondern auch, dass selbst Lehrpersonen ohne Vorkenntnisse im Bereich der Informatik Computational Thinking vermitteln können. Ziel ist es nun, das Programm zur Förderung von Computational Thinking in der Schweiz umzusetzen.

WIE FUNKTIONIERT COMPUTATIONAL THINKING?

Jeannette Wing definiert *Computational Thinking* wie folgt:

> **Computational Thinking ist der Gedankenprozess, der sowohl die Formulierung eines Problems als auch die Repräsentation der Problemlösung so darstellt, dass sie von Menschen oder durch Maschinen ausgeführt werden könnte.**

Diese Definition impliziert in den meisten Fällen einen dreistufigen Prozess:

1. **Problemformulierung:** In der einfachsten Form eine präzise Fragestellung basierend auf einer Problemanalyse.
2. **Repräsentation einer Lösung:** Eine Darstellung basierend auf einer Kombination von Text und Diagrammen, zum Beispiel eine Möbelmontageanleitung, ein Rezept oder ein Computerprogramm.
3. **Ausführung und Bewertung der Lösungsrepräsentation:** Zum Beispiel die Umsetzung eines Kuchenrezeptes und die darauffolgende Überprüfung der Qualität des gebackenen Kuchens.

Im Beispiel mit der Grossmutter besteht die Problemformulierung aus dem Erkennen der spezifischen Fragestellung beim Kuchenbacken. Zum Beispiel beinhaltet dies die Erkenntnis, dass der Zuckerguss erst ganz am Schluss auf den gebackenen Kuchen gegossen werden kann. Die Repräsentation einer Lösung kann das Kuchenrezept sein, also die detaillierte Beschreibung von zeitlich aufeinanderfolgenden oder parallelen Arbeitsschritten. Die Lösungsrepräsentation kann durch die Grossmutter selbst oder zum Beispiel von den Enkeln, für welche die Grossmutter das Rezept geschrieben hat, ausgeführt werden. Grossmutter bzw. Enkel können anschliessend überprüfen, wie gut das Ergebnis ausgefallen ist, indem sie den Kuchen essen.

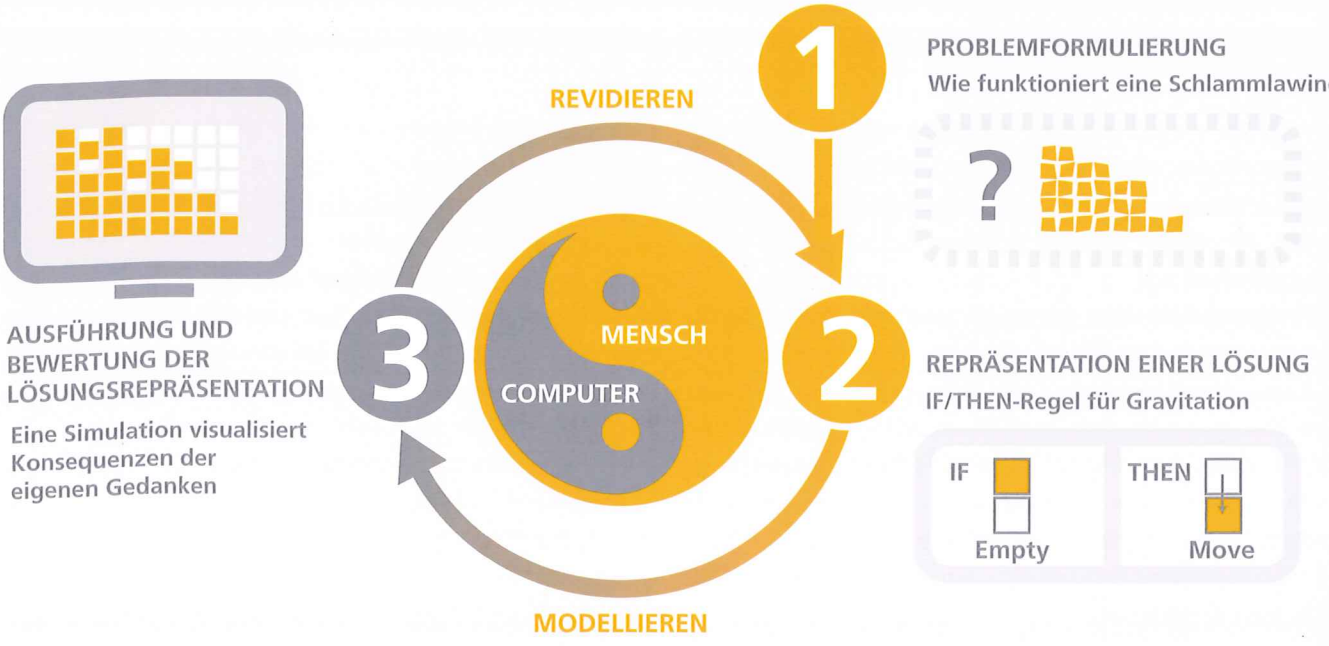
Wieso braucht man *Computational Thinking*? Es gibt viele Ziele, aber dieses Dokument fokussiert auf folgende zwei Aspekte:

- **Delegation und Automation:** Die Repräsentation einer Lösung kann dazu benutzt werden, einen Arbeitsprozess entweder an andere Personen oder an einen Computer zu delegieren. Das Kuchenrezept, wenn es detailliert, präzise und verständlich genug ist, kann von beliebigen

Personen in einen Kuchen umgesetzt werden. Auch ein Computerprogramm ist eine Lösungsrepräsentation. Ein sehr einfaches Beispiel einer Automation ist die von einer Benutzerin «programmierte» Aufzeichnung einer Fernsehsendung.

- **Wissenserwerb:** Der einmalige und geradlinige dreistufige Prozess funktioniert nicht wirklich, wenn die Lösung eines Problems nicht offensichtlich ist. Dies ist typischerweise der Fall, wenn es um Wissenserwerb geht. Im Kontext von Bildung ist gerade diese Anwendung von *Computational Thinking* aber sehr interessant. Der allgemeine *Computational-Thinking*-Prozess ist iterativ (siehe Abbildung Seite 8) und wird unten anhand eines Beispiels beschrieben.

Computational Thinking bietet beim aktiven Wissenserwerb eine ideale Synthese der Fähigkeiten von Menschen und Computern. Der Prozess beginnt ① mit einer Fragestellung. Zum Beispiel: «Wie funktioniert eine Schlammlawine?» Zu diesem Zeitpunkt X kennen wir die Lösung noch nicht und können sie folglich nicht als fertige Repräsentation, also als fertiges Programm darstellen. Aber wir können eine erste Annäherung an eine Lösung erstellen. Eine Schlammlawine kann man sich z. B. als eine Anhäufung von Schlammblöcken vorstellen. Diese Blöcke könnten aufeinander und nebeneinander gestellt werden. Mit einem *Computational Thinking Tool* wie *AgentSheets* können diese Blöcke visualisiert und als Objekte – auch Agenten genannt – organisiert in einer Tabelle repräsentiert werden. In Schritt ② drückt der Anwender bzw. die Anwenderin Zusammenhänge zwischen Agenten rechnerisch aus. Diese Zusammenhänge werden im Falle von *AgentSheets* oder ähnlichen Werkzeugen als Programmierregel ausgedrückt. Die Regel stellt eine einfache Approximation von Gravitation dar. Wenn (IF) sich unter einem Schlammblock nichts befindet (*empty below*), dann (THEN) fällt der Block nach unten (*move down*). Der Mensch hat somit eine Idee rechnerisch erfasst. Nun ist es die Aufgabe des Computers ③ die Konsequenzen dieser Idee zu visualisieren. Es ist nicht die Aufgabe des Computers, die Idee zu finden, darin ist der Mensch wesentlich besser. Der Computer kann aber dank seiner rechnerischen Fähigkeiten solche Konzepte in einer Art und Weise visualisieren, wie es für den Menschen schwierig oder sogar unmöglich ist.



Wie verbindet Computational Thinking den menschlichen Denkprozess mit dem Computer? Es geht nicht darum, zu denken wie ein Computer, sondern gemeinsam mit dem Computer zu denken. Das heisst, Computational Thinking ist eine Denkweise, die den Computer als Instrument benutzt, um den Denkprozess zu unterstützen. Als Instrument ist der Computer vergleichbar mit einem Mikroskop – er hilft, schwer verständliche Zusammenhänge sichtbar zu machen, und unterstützt damit die Entwicklung von Computational Thinking.

Synthese der Fähigkeiten von Menschen und Computern

COMPUTATIONAL THINKING ≠ PROGRAMMIEREN

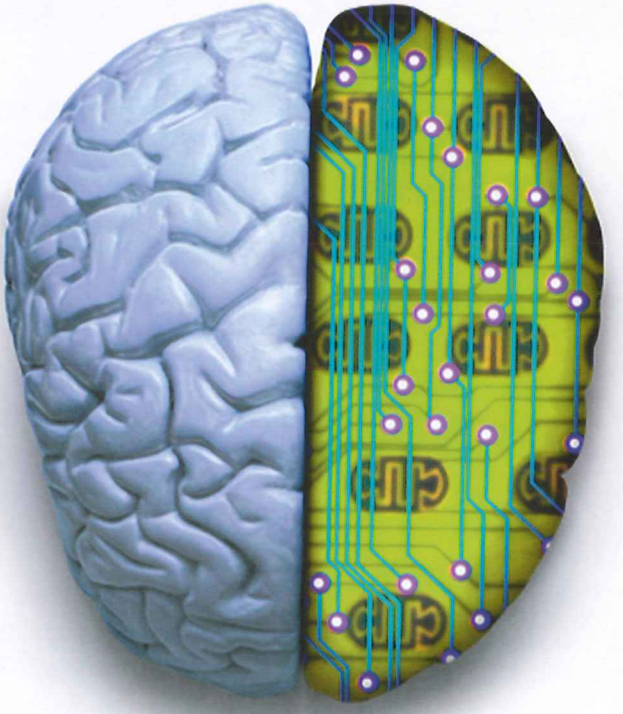
Computational Thinking ist nicht das Gleiche wie Programmieren. Es bezieht sich vielmehr auf das konzeptuelle Verständnis von Programmierkonzepten, die auf verschiedene Programmiersprachen anwendbar sind. Jede Programmiersprache ist hingegen durch eine eigene Syntax gekennzeichnet, die Benutzerinnen und Benutzer beherrschen müssen, um funktionierende Programme schreiben zu können. Syntaktische Details einer Programmiersprache sind aber nicht so wichtig wie das konzeptuelle Verständnis von Programmierkonzepten. Das heisst nicht, dass die Kenntnis unterschiedlicher Programmiersprachen nicht von Bedeutung wäre für *Computational Thinking*, aber *Computational Thinking* bevorzugt pädagogische Ansätze, die sich am Spracherwerb einer Muttersprache anlehnen. Beim Erlernen der Muttersprache fokussiert ein Kleinkind auf relevante, zielorientierte Kommunikation und nicht auf grammatische Regeln. Ganz ähnlich fokussiert *Computational Thinking* auf Ziele, wie zum Beispiel das Ziel, eine Simulation zu bauen, um Wissen zu erwerben. Wer einen Waldbrand simuliert, ist beispielsweise interessiert am Zusammenhang von Baumdichte und der Wahrscheinlichkeit, dass der Wald komplett abbrennen könnte, und nicht daran, ob eine «Schleife» (Loop-Anweisung) zur Implementation dieser Simulation verwendet wurde. Programmierkompetenz ist zwar nötig und auch erwünscht, aber wie das korrekte Befolgen einer IKEA-Montageanleitung einen Benutzer, eine Benutzerin nicht automatisch zum Schreiner macht, ist Programmierkompetenz nicht gleichzusetzen mit dem Verständnis von *Computational Thinking Patterns*.

Computational Thinking bedeutet, sich in einer eleganten Weise auf das Wesentliche zu konzentrieren, um ein möglichst breites Publikum für Informatikkonzepte zu interessieren. Aber gerade dieses Publikum empfindet traditionelles Programmieren häufig als «schwierig und langweilig». Oft sind nämlich Programme, die zwar eine relativ einfache Idee ausdrücken, nicht selten verblüffend lang und kompliziert, bis zu einem Punkt, an dem man die eigentliche Idee im Code kaum mehr erkennt.

Richtig und einfacher gemacht, könnte Programmieren aber durchaus interessant sein. Hitchcock hat einmal das Konzept von Drama in Filmen definiert als das reelle Leben, aus dem die langweiligen Sequenzen entfernt würden. Ganz in diesem Sinne hat Repenning in den letzten 20 Jahren Werkzeuge gebaut, die man als *Computational Thinking Tools* bezeichnen kann, weil sie den Fokus auf das Wesentliche richten. Mit *AgentCubes* kann beispielsweise ein *Pac-Man-Game* mit künstlicher Intelligenz mit nur gerade zehn Regeln gebaut werden.

Ein Computational Thinking Tool ist ein Werkzeug, das mehr als nur eine Programmierumgebung ist. Es wendet die Prinzipien von *Computational Thinking* auf den Programmierprozess selbst an. Ein *Computational Thinking Tool* soll auf eine möglichst effektive Art die Konsequenzen des eigenen Denkens visualisieren. Vor allem bei Anfängern führt die Verzögerung zwischen dem Zeitpunkt, an dem ein Programmierfehler gemacht wird, und der ersten Manifestation dieses Fehlers zu extremer Frustration. Je länger diese Verzögerung ist, desto wahrscheinlicher wird es, dass der Programmierer während dieser Zeit zusätzliche Entscheidungen trifft und im Code einbaut. Mit zunehmender Verzögerung wird es daher immer schwieriger, den Ursprung des Fehlers zu finden. Ganz extrem war dies bei der Programmierung mit Lochkarten. Oft dauerte es Stunden, bis man das Resultat des Programms einsehen konnte. Leider basieren auch heute immer noch viele Programmierumgebungen auf diesen Ansätzen. Zwar muss man sich nicht mehr mit einem Stapel von Lochkarten herumschlagen, aber ein Programm muss komplett sein, bevor man es kompilieren (übersetzen) und dann endlich laufen lassen kann. Live Programming geht in die richtige Richtung, indem es seinen Benutzern erlaubt, ein Programm inkrementell zu verändern, ohne das Programm neu starten zu müssen. Dadurch können die Benutzer schnell sehen und verstehen, was eine Veränderung wirklich bewirkt. Noch weiter geht das

Konzept von Conversational Programming in *AgentCubes*, wo ein Programm einen Schritt in die Zukunft ausführt, um dem Benutzer zu zeigen, was es machen wird. Schlussendlich geht es bei *Computational Thinking Tools* darum, eine sinnvolle Symbiose der Fähigkeiten von Menschen und Computern zu finden. Gerade mit den heutigen Computern, die über enorme Rechenleistungen verfügen, soll endlich der Programmierprozess des Menschen aktiv unterstützt werden.



Symbiose von Mensch und Computer

EINE COMPUTATIONAL-THINKING-STRATEGIE: SCALABLE GAME DESIGN

Das Projekt *Scalable Game Design*[1] unter der Leitung von Alexander Repenning untersucht, wie man informatische Bildung basierend auf *Computational Thinking* systematisch in Schulen integriert. Ziel ist es, Schülerinnen und Schüler für Informatik zu motivieren und durch die Benutzung von Game Design in Informatik auszubilden. Der Ansatz wurde im Rahmen einer der grössten Studien zur informatischen Bildung in den USA mit weit über 10'000 Teilnehmerinnen und Teilnehmern erprobt. Scalable Game Design basiert auf einer Vier-Punkte-Strategie:

1. **Durchdringung:** Leicht adaptierbare Lernmodule sollen früh in existierende Klassen integriert werden, so dass grosse und heterogene Gruppen von Kindern mit *Computational Thinking* in Berührung kommen. Informatische Bildung inklusive Programmieren soll nicht beschränkt werden auf Begabtenförderung, Sommerkurse oder Wahlfächer. *Computational Thinking* kann bereits im Kindergarten beginnen – mit oder ohne Computer. Ziel ist es, allen Schülerinnen und Schülern die Möglichkeit zu bieten, *Computational Thinking* zu entwickeln. *Computational Thinking* kann in verschiedenste Gebiete integriert werden – in MINT-Fächer (Mathematik, Informatik, Naturwissenschaften und Technik), aber auch in Sprachen, Gestalten und Kunst, Musik oder sogar Sport. Resultate: Diese Art informatische Bildung ist universell einsetzbar mit Lehrpersonen und Schüler/innen aus einem weiten Spektrum von Schulen. In den USA gab es Schulprojekte, die sowohl in ländlichen Gegenden als auch in Grossstadtschulen durchgeführt wurden – Pilotprojekte fanden sogar in Indianerreservaten statt. Durchschnittlich beteiligten sich zu 45 % Mädchen an der Studie.

1 www.scalablegamedesign.ch scalablegamedesign.cs.colorado.edu

2. **Motivation:** Die Grundidee besteht darin, dass Motivation die Voraussetzung ist, um Kompetenz zu erwerben. Umgekehrt führt die Aneignung von Kompetenzen nicht unbedingt zu Motivation. *Scalable Game Design* beinhaltet eine Reihe von sorgfältig abgestuften Spiel- und Simulationsprojekten. Die Schülerinnen und Schüler bauen schon nach kurzer Zeit selbst Computerspiele, die unterschiedliche Schwierigkeitsniveaus aufweisen und sich stufenweise von einfachen zu komplexen Programmierinhalten entwickeln. Programmiererfahrungen werden zu keinem Zeitpunkt vorausgesetzt. Resultate: Obwohl die meisten Schülerinnen und Schüler *Scalable Game Design* in obligatorischen Kursen lernen, wollen 74 % der Jungen und 64 % der Mädchen in den USA weitermachen.

3. **Kompetenz:** Mit computerunterstützten Instrumenten können die Projekte und Programme der Lernenden analysiert werden, so dass Kompetenzen objektiv messbar sind. Diese Ergebnisse beinhalten Lernkurven und den Wissenstransfer verschiedener *Computational Thinking Patterns* – von Computerspielen bis hin zu wissenschaftlichen Simulationen. Resultate: Ein Werkzeug für Auswertungen mit dem Namen *Computational Thinking Pattern Analysis* hat gezeigt, dass Schülerinnen und Schüler Ideen wie Kollision und Diffusion als Konzepte erkennen und einsetzen können. Dies nicht nur, um Spiele zu bauen, sondern auch um Simulationen abzubilden.

4. **Pädagogik:** Die Weiterbildung für Lehrpersonen beinhaltet unter anderem neue pädagogische Konzepte, die das stufenweise Erlernen von Programmierinhalten fördert und schnelle Lernerfolge auch für jene ohne Programmiererfahrung ermöglicht. Ziel ist es, möglichst viele Schülerinnen und Schüler zur Teilnahme zu motivieren und das Vermitteln von Programmierkenntnissen mit eigenständigem Arbeiten in Einklang zu bringen. Resultate: Es hat sich gezeigt, dass die Verwendung einer geeigneten Pädagogik ein Hauptfaktor ist, um das Interesse von Mädchen und Knaben gleichermassen zu wecken.

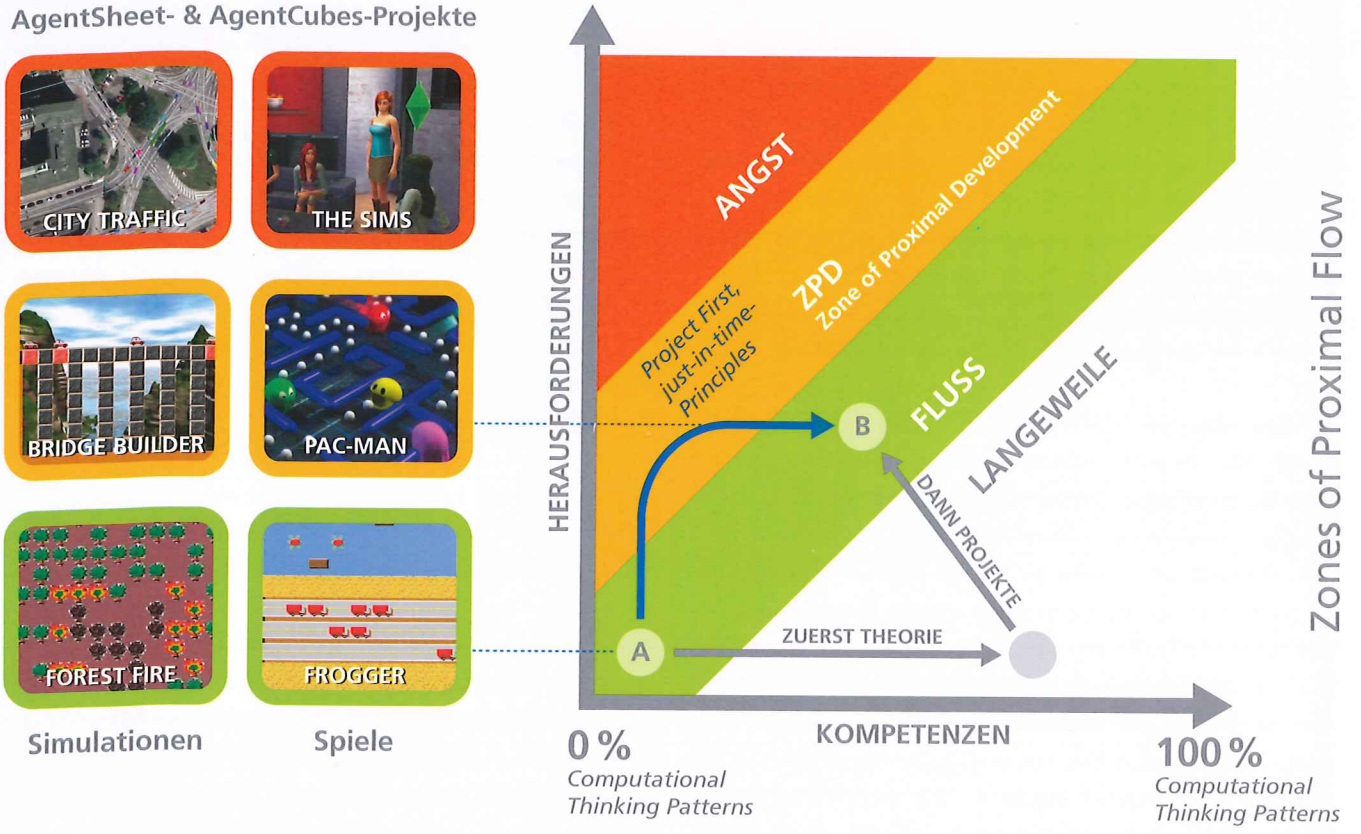
Lerntheoretische Grundlagen

Die Strategie von *Scalable Game Design* ist als Konzept der «Zones of Proximal Flow» (Zone des proximalen Flows) zusammengefasst (Abbildung Seite 16). Demnach sind Individuen besonders motiviert, wenn sie sich in der Zone des sogenannten «Flow» [Csíkszentmihályi] befinden, wo Herausforderung und individuelle Kompetenzen genau ausbalanciert sind. Tennis ist dafür ein gutes Beispiel.

Wenn ein Anfänger gegen einen Profi wie Roger Federer spielen würde, wäre die Partie sehr unausgewogen. Der Anfänger hat praktisch keine Tenniskompetenz. Gegen einen Profi zu spielen ist eine riesige Herausforderung. Der Anfänger ist daher nicht im Flow, sondern in der Zone der Angst (Herausforderung > Kompetenz). Umgekehrt, aus Sicht eines Profis, ist das Spiel gegen einen Anfänger alles andere als eine Herausforderung. Der Profi landet in der Zone der Langeweile (Herausforderung < Kompetenz).

Vygotsky, der viel zum Thema soziales Lernen forschte, hat den Begriff «Zone of Proximal Development» (Zone der proximalen Entwicklung) geschaffen. Folgt man Vygotsky, gelangt ein Lernender in die Zone der proximalen Entwicklung, wenn er an die Grenze des eigenen Verständnisses stösst. Wenn genau dann eine Lehrperson dem Lernenden hilft, lernt man am besten. Die «Zones of Proximal Flow» stellen eine Kombination von Flow und der Zone der proximalen Entwicklung dar. Sie definieren verschiedene Lernzonen. Die Frage ist nun, wie jemand dazulernt. Ein Lernender an Punkt A, mit wenig Kompetenz und niedrigen Herausforderungen, möchte zu Punkt B kommen. Traditionell folgt man einem indirekten Weg (blauer Pfad): Zuerst werden den Lernenden Grundlagen, zum Beispiel im Bereich der linearen Algebra, ohne Bezug zu projektorientierten Herausforderungen vermittelt. So lernen Schülerinnen und Schüler beispielsweise, wie man mit Matrizen umgeht, ohne konkrete Anwendungen zu verstehen. Erst viel später, möglicherweise im Studium, werden sie vielleicht im Rahmen eines Computer-Graphics-Kurses mit Anwendungen linearer Algebra konfrontiert. Leider mussten sie jahrelang den Umweg durch die Zone der Langeweile machen.

Der «Project First, Just-in-time-Principles»-Ansatz, der im Rahmen der Strategie von *Scalable Game Design* verwendet wird, ist grundlegend anders. Er beginnt unmittelbar mit einem Game-Projekt, ohne dass Schülerinnen und Schüler irgendwelche Programmierkompetenzen haben müssen. Er bringt die Lernenden sehr schnell an die Grenze ihrer Kompetenzen, aber navigiert sie mit entsprechender pädagogischer Unterstützung sorgfältig durch die Zone der proximalen Entwicklung und wieder zurück in die Zone des Flow für eine optimale Lernerfahrung.


AgentSheet- & AgentCubes-Projekte
CITY TRAFFIC
THE SIMS
BRIDGE BUILDER
PAC-MAN
FOREST FIRE
FROGGER
Simulationen
Spiele
HERAUSFORDERUNGEN
ANGST
ZPD
Zone of Proximal Development
Project First, just-in-time-Principles
FLUSS
LANGEWEILE
B
A
DANN PROJEKTE
ZUERST THEORIE
KOMPETENZEN
0 %
Computational Thinking Patterns
100 %
Computational Thinking Patterns
Zones of Proximal Flow

Das *Scalable-Game-Design*-Curriculum besteht aus zwei abgestuften Pfaden, die die Lernenden Schritt für Schritt neuen Herausforderungen aussetzen. Der erste Pfad fokussiert auf Spiele und beginnt mit einfachen 1980-*Arcade-Style*-Spielen wie Frogger. Aufwendigere Spiele wie *Pac-Man* führen bereits Konzepte von künstlicher Intelligenz ein, zum Beispiel wie finden die Geister den Weg durch das Labyrinth zum *Pac-Man*. In noch aufwendigeren Spielen werden sogar psychologische Modelle wie Maslows Bedürfnispyramide umgesetzt. Der zweite Pfad fokussiert auf Simulationen. Die unterschiedlichen Herausforderungen dieser Spiele und Simulationen können als Kompetenzprofile verstanden werden, die auf *Computational Thinking* Pattern basieren. Diese beschreiben typischerweise die Interaktion zwischen Objekten auf einer Ebene, unabhängig von den Details irgendwelcher Programmiersprachen, aber auch unabhängig von Anwendungen wie Computerspielen oder Simulationen. Ein einfaches Beispiel für ein solches *Computational Thinking Pattern* ist eine Kollision von zwei Objekten. Wenn der Frosch im *Frogger*-Spiel beispielsweise versucht, die befahrene Strasse zu überqueren, bedeutet die Kollision zwischen dem Frosch und einem Auto den Tod für den Frosch. Das gleiche *Computational Thinking Pattern* kommt auch in Simulationen sehr häufig vor. So ist zum Beispiel die Kollision von Molekülen sehr wichtig in der Chemie und die Kollision von Menschen, die mit einem ansteckenden Virus aufeinandertreffen, kann benutzt werden, um das Konzept einer Epidemie zu simulieren.

Computational Thinking Patterns

Die explizite Benutzung von *Computational Thinking Patterns* ist sehr wichtig für eine Wissensvermittlung, die den Transfer durch *Computational Thinking* einschliessen soll. Generell gilt, dass Transfer von Wissen nicht einfach passiert, auch wenn aus der Sicht der Lehrperson eine Analogie von Ideen zu existieren scheint. Also, wenn ein Schüler oder eine Schülerin ein Spiel bauen kann, bedeutet dies noch lange nicht, dass dieselbe Schülerin eine Simulation kreieren könnte, auch wenn die grundsätzlichen Konzepte des Spieles und der Simulation ähnlich sind.

Die Liste der *Computational Thinking Patterns* beinhaltet relativ einfache Objektinteraktionen:

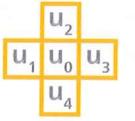
- **Kollidieren:** Zwei oder mehrere Objekte prallen aufeinander, z. B. kann im *Frogger*-Spiel der Frosch von einem Auto überfahren werden.
- **Stossen/Schieben:** Eine Person schiebt eine Schubkarre.
- **Ziehen:** Ein Zug zieht mehrere Waggons.
- **Generieren:** Etwas Neues entsteht, sprich wird generiert. Der Geldautomat generiert Geldscheine.
- **Absorbieren:** Ein Staubsauger absorbiert Krümel vom Boden.
- **Transportieren:** Ein Skateboard transportiert einen Teenager.

Daneben existieren anspruchsvollere *Computational Thinking Patterns*, die in Spielen und Simulationen anwendbar sind. Diffusion beispielsweise ist ein sehr mächtiges *Computational Thinking Pattern*. Schon 1952 hat der berühmte Mathematiker und Informatiker Alan Turing Diffusion definiert als die Bewegung von [chemischen] Agenten (Teilchen) aus Regionen hoher Konzentration hin zu Regionen geringerer Konzentration. Er nutzte Computer, um die Diffusion zu berechnen. In einem zweidimensionalen Gitter kann man Diffusion durch folgende Gleichung darstellen:

$$u_{0,t+1} = u_{0,t} + D\sum_{i=1}^{n}(u_{i,t} - u_{0,t})$$

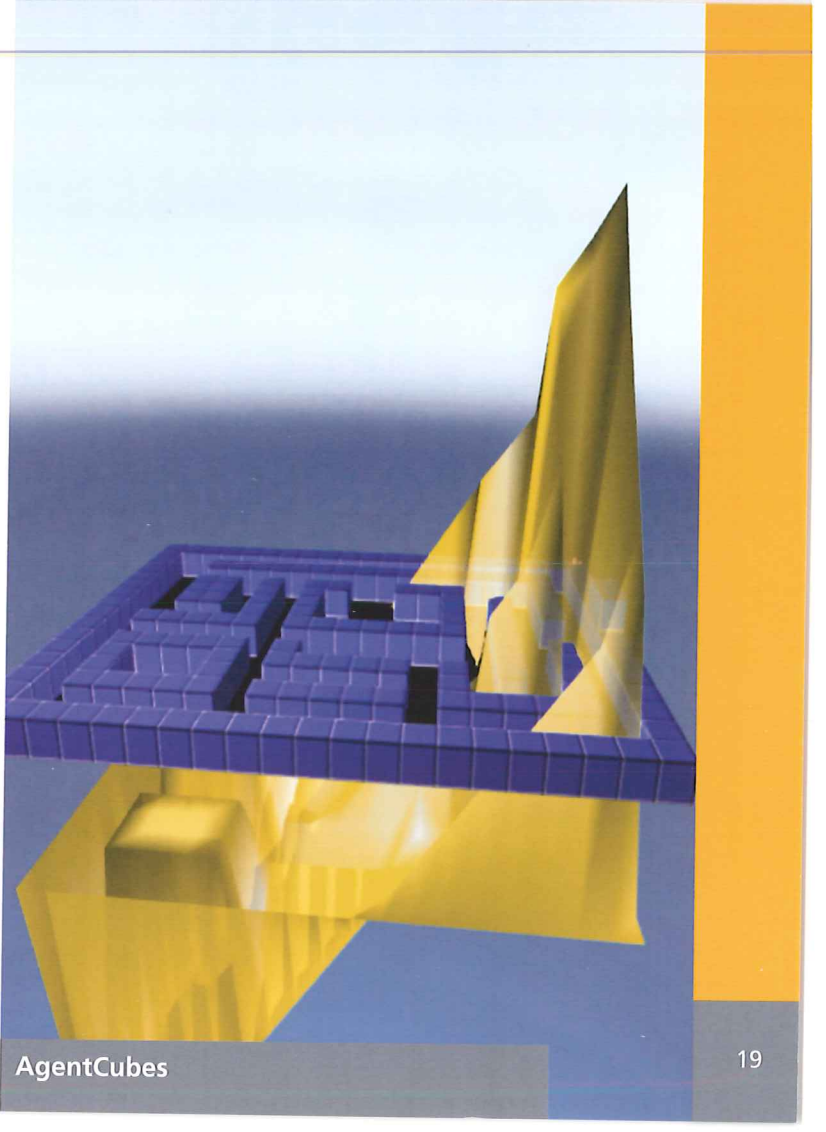
WOBEI:

n = Anzahl benachbarte Agenten, die als Eingabe für die Diffusionsgleichung benützt werden
$u_{0,t}$ = Diffusionswert des Agenten im Zentrum
$u_{i,t}$ = Diffusionswert des Nachbaragenten ($i>0$)
D = Diffusionskoeffizient [0..0,5]



Diese auf den ersten Blick sehr komplexen Konzepte können Schülerinnen und Schülern zugänglich und verständlich gemacht werden, so dass sie in der Lage sind, komplizierte Probleme elegant zu lösen. In einem *Pac-Man*-Spiel kann Diffusion verwendet werden, damit die Geister den *Pac-Man* mittels künstlicher Intelligenz überall im Labyrinth

finden können. Diffusion kann in diesem Beispiel erklären, wie *Pac-Man* einen Geruch verbreitet, dem die Geister folgen. Schon 10-jährige Kinder können dies umsetzen. Dies bedeutet, dass einerseits der Einfluss von Motivation massiv unterschätzt wird: Die Motivation beim Erstellen von Computerspielen ist oft so gross, dass Kinder, die sich nicht für Computer oder Programmieren interessieren, plötzlich mathematisch analytische Fähigkeiten zeigen, die niemand für möglich gehalten hätte. Grund dafür ist, dass sie Probleme lösen, die sie tatsächlich interessieren, und damit Mathematik als fantastisches Werkzeug erkennen. Andererseits spielt ein *Computational Thinking Tool* wie *AgentCubes* eine entscheidende Rolle: Es unterstützt durch Visualisierung die Problemlösung. In *AgentCubes* kann man zum Beispiel 3-D-Plots einblenden, die die Geruchsverteilung in der Spielwelt als Wert der Diffusion aufzeigen. Eine bergähnliche Erhebung über dem *Pac-Man* zeigt in diesem Fall die Verteilung der Diffusion, und so kann der Anwender den Suchalgorithmus als Konzept verstehen, das von den Geistern eingesetzt wird, um *Pac-Man* zu finden (Abbildung rechts).



AgentCubes

EIN SCHULMODELL

Wie lässt sich *Computational Thinking* mit dem Schweizer Schulmodell vereinen? Der Fokus liegt vor allem auf der Primarstufe. Als Kernkompetenz muss *Computational Thinking* hier beginnen. Auch aus der Perspektive von Motivation ist die Ausbildung auf Primarstufe der Knackpunkt. Im Alter von etwa elf Jahren entscheiden schon viele Kinder fälschlicherweise, dass sie nicht geeignet sind für Mathematik oder Informatik. Wer diese Schlussfolgerung einmal gezogen hat, ist später kaum mehr bereit, diese Themen aufzunehmen.

Primarschulen

Wenn man *Computational Thinking*, wie vorne beschrieben, in die Kategorien Problemformulierung, Repräsentation einer Lösung und Ausführung der Lösungsrepräsentation einteilt, sieht man, dass diese oder zumindest sehr ähnliche Konzepte in den Lehrplänen schon existieren und erfolgreich umgesetzt werden. Das ist wichtig, denn Primarlehrpersonen sind keine Fachlehrpersonen und müssen mit wenigen Ausnahmen alle Fächer unterrichten, von Lesen bis Sport. Folglich soll das informatische Bildungskonzept integrativ und nicht exklusiv sein. Informatische Bildung auf Primarstufe soll also nicht isoliert stehen, sondern in die anderen Fächer integrierbar sein, um von den Lehrpersonen nicht als Konkurrenz zu andern Unterrichtsfächern betrachtet zu werden. Es handelt sich also um ein «übergreifendes» Unterrichtsprinzip. Als Kernkompetenz des 21. Jahrhunderts, ähnlich wie Lesen und Schreiben, kann *Computational Thinking* dazu benutzt werden, verschiedenste Bereiche zu unterstützen:

- **Sprachen/Gestalten/Musik/Sport:** Der Designprozess eines Spiels benutzt die grammatikalische Analyse von Spiel- respektive Simulationsbeschreibungen bezüglich Hauptwörtern und Verben, um so die Objekte und ihre Interaktionen zu verstehen. Das ist sehr nützlich im Sprachunterricht inklusive Fremdsprachen. Das Kreieren von 2-D- oder 3-D-Spiel- und Simulationsobjekten beinhaltet Elemente, die für den Gestaltungsunterricht relevant sind. Durch die Kombination von computerunterstützter Musiksynthesis (z. B. durch MIDI) und Programmierkonzepten entsteht das synergetische

Konzept von *Computational Music Thinking*. Im Sport können Spiele wie *Frogger* ausgelebt werden (*Computer-Science-Unplugged*-Konzept), ganz ohne die Benutzung von Computern.

- **MINT (Mathematik, Informatik, Naturwissenschaften, Technologie):** Schon einfache Spiele verbinden wichtige mathematische Konzepte mit Spielkonzepten: In einem *Frogger*-Spiel soll ein Frosch eine befahrene Strasse überqueren, ohne mit den Autos zu kollidieren. Der Schwierigkeitsgrad des Spiels hängt direkt mit der Frequenz und der Wahrscheinlichkeit zusammen, mit der neue Fahrzeuge auftauchen. Simulationen wie zum Beispiel die Simulation eines Waldbrandes oder einer Schlammlawine lassen Schüler naturwissenschaftliche Prinzipien erfahren, anstatt sie nur auswendig zu lernen.

- **Informatik:** *Computational Thinking* ist auch kompatibel mit traditionellem Programmierunterricht. *Computational Thinking Patterns* können von den Lernenden in verschiedensten Programmiersprachen angewandt werden. *AgentSheets* und *AgentCubes* online schlagen eine Brücke zum imperativen Programmieren, indem sie eine Funktion anbieten, die der Anwender verwenden kann, um sich den Quellencode (Java bzw. JavaScript) anzusehen.

- **ICT und Medien:** Durch das Erstellen von Spielen und Simulationen im Kontext von *Computational Thinking* werden auch viele Punkte der Medienbildung, insbesondere der Umgang mit Medien abgedeckt. Durch das Programmieren erhalten die Kinder nicht zuletzt auch die Gelegenheit, von der Konsumhaltung wegzukommen, indem sie selbst kreieren. Ganz wichtig ist dabei die Vorstellung, dass der *Computational-Thinking*-Teil von ICT und Medien das Programmieren explizit einschliesst und sich nicht auf Tastaturschreiben oder Textverarbeitung beschränkt.

Sekundarstufe 1

Die Sek1 sollte auch bezüglich *Computational Thinking* eine Weiterführung der Primarschule sein. So erlaubt beispielsweise das Verständnis von Maslows Bedürfnispyramide die Erstellung von sehr anspruchsvollen Spielen wie etwa *The Sims*. Der Schwerpunkt sollte auf Simulationen gelegt werden, die vor allem in naturwissenschaftlichen Fächern zum Zug kommen. Zudem ist die Basis der Werkzeuge, die naturwissenschaftlich eingesetzt werden, erheblich zu erweitern. Kalkulationstabellen, die typischerweise von selbst programmierten Simulationen produziert werden, können z. B. als Werkzeug für die Analyse und Visualisierung von grossen Datenmengen benutzt werden. Das Gesamtkonzept auf dieser Stufe sollte immer noch integrativ sein, erweitert durch informatikfokussierte Kurse, die von Fachlehrpersonen unterrichtet werden.

Sekundarstufe 2

Viele Schulen der Sekundarstufe 2 unterrichten bereits das Ergänzungsfach Informatik durch qualifizierte Lehrpersonen mit formalen Informatikkenntnissen. Die Teilnahme, vor allem von Schülerinnen, ist jedoch sehr gering. Solange eine flächendeckende informatische Bildung auf Primar- und Sek-1-Stufe nicht existiert, müssen Sek-2-Schulen obligatorische Informatikkurse anbieten, die für die grosse Mehrheit der Schülerinnen und Schüler geeignet sind. Diese Kurse sollten sich auf die Motivation mit spannenden und genderneutralen Ansätzen wie etwa *Game Design* fokussieren.

HASLERSTIFTUNG

Hasler Stiftung
Hirschengraben 6
CH-3011 Bern
Tel. +41 31 381 41 41
Fax +41 31 381 67 00

www.haslerstiftung.ch
www.fit-in-it.ch