Rainer Osterwalder / Marcel Taisch

Unterrichtseinheit zum Thema Induktion im Informatikunterricht

Voraussetzungen der Lernenden

Die Unterrichtseinheit eignet sich für Schüler*innen ab Ende des 1. Jahres des obligatorischen Fachs Informatik (OFI) d.h. Ende 1. MAR- oder 2. MAR-Stufe.
Die Lernenden haben den Einstieg ins Programmieren mit Tigerjython und Turtlegrafiken erfolgreich abgeschlossen.
Die Beweismethode der vollständigen Induktion, mit "Induktionsannahme", "Induktionsverankerung" und "Induktionsschritt" kennen die Lernenden noch nicht.

Zielsetzung:

Das Programmieren mit Hilfe der Turtle ist für viele Schüler*innen attraktiv, da sie dabei ein "instant Feedback" bekommen. Diese Methode fördert auch das Programmieren nach dem Prinzip von "**Trial and Error**". Fehler werden sofort erkannt, das Programm kann schnell abgeändert und kontrolliert werden, ob es nun funktioniert. Häufig erreichen die Schüler*innen ihr Ziel durch progressive Annäherung.
Dies Vorgehen bringt sicherlich Vorteile - v.a. im motivationalen Bereich. Wird das Programmieren anspruchsvoller, sollten die Lernenden besser geeignete Konzepte kennen lernen.
Für das OFI wird hier häufig der Begriff des "**Computational Thinking**" eingeführt. Dabei soll die "informatische Denkweise" nicht nur zur strukturierten Lösung von Informatikproblemen führen, sondern auch helfen Probleme aus anderen Fachbereichen zu bearbeiten. Dies wird gerne als ein Grund für den allgemeinbildenden Charakter des Fachs Informatik herangezogen.

In dieser Unterrichtseinheit sollen die Denkfiguren des **Computational Thinking** spielerisch angewandt werden. Dabei kommen folgende Prinzipien vor:
Decomposition: Ein grosses Problem wird in mehrere, kleinere, einfach zu handhabende Probleme aufgeteilt und gelöst. Dabei geht es darum, zuerst ein kleineres, konkretes Problem zu lösen und dieses dann auf immer allgemeinere Lösungen zu erweitern.
Pattern Recognition: Hierbei geht es darum, wiederkehrende Muster und ähnliche Teillösungen zu erkennen und sinnvoll einzusetzen.
Abstraction: Dabei beschränkt man sich auf das Wichtigste und lässt Irrelevantes weg. Das hilft auch bei der Decomposition und Pattern Recognition. Zudem unterstützt sie dabei, das Ziel eines komplexeren Problems, nicht aus den Augen zu verlieren.
Algorithm Design: Die Lösung für eine ganze Klasse von Problemen wird in klar definierte einzelne Schritte aufgeteilt und logisch geordnet, so dass sie effizient und automatisiert gelöst werden können.

Ohne **induktives Vorgehen** wäre das Verfahren des Computational Thinking nicht möglich. Durch die spielerische Anwendung dieser Denkstrukturen führen wird die Lernenden an eine **erfolgreiche Erfahrung des induktiven Vorgehens** heran. Auf diesen Erfahrungen kann später in einem Spiral-Curriculum aufgebaut werden, wenn es beispielsweise um die Einführung der Beweismethode der vollständigen Induktion geht.

(**Nebenbei:** Es lohnt sich an dieser Stelle auch, den Lernenden zu zeigen, wie sie diese Form der Problemlösung auch für Probleme ausserhalb der Informatik oder der Mathematik sinnvoll anwenden können.)
Wir werden die verschiedenen Denkstrukturen anhand des "Nim-Spiels" durcharbeiten, das **induktive Vorgehen** aufzeigen und die Erkenntnisse auf verwandte Probleme anwenden.

(Um eine Google Suche nicht zu einfach zu gestalten, nennen wir das Spiel «der letzte Stift gewinnt». Zudem soll derjenige gewinnen, der den letzten Stift nehmen kann. Beim Original ist derjenige, der den letzten Stift nehmen muss, der Verlierer.)

Aufgaben:

Das Spiel: "Der letzte Stift gewinnt"

Du hast eine beliebige Anzahl Stifte. (Zum Beispiel 21).

Regeln:

- A beginnt, dann kommt B zum Zug. Dies geht abwechslungsweise so weiter, bis einer gewinnt.
- Jeder Spieler darf bei seinem Zug 1, 2 oder 3 Stifte entfernen.
- Wer den letzten Stift entfernen kann, hat gewonnen!

Ziel ist es, dieses Spiel so zu programmieren, dass der Computer möglichst oft gewinnt.

Wir werden das zusammen angehen!

Decomposition:

a. Wir teilen die "grosse Aufgabe" in "bewältigbare" kleiner Aufgaben. Findest du solche Aufgaben?
Wir lassen die Schüler*innen in Partnerarbeit, solche Teilaufgaben zusammenstellen. Danach sammeln wir die Ergebnisse.
Erwartete Antworten könnten sein: «Darstellung (grafisch?)», «User-Eingabe», «Computer-Zug», «Computer-Taktik (Gewinnstrategie)», «verschiedene Startmengen», «zufälliger Beginner», «Gewinner bekanntgeben» ...
Die Ideen werden gesammelt (Tafel, Monitor, Flip-Chart...). Sie werden später wieder aufgegriffen.

b. Wir wollen uns zuerst um die Computer-Taktik, also die Gewinnstrategie kümmern. Wie kommen wir zu dieser Strategie?
Erwartete Antworten könnten sein: «Wir programmieren es so, dass der Computer zufällig 1,2 oder 3 Stifte wählt und sehen, was passiert», «wir beginnen einmal mit einer kleineren Anzahl Stifte», «wir spielen es so lange, bis wir eine Gewinnstrategie haben» ... (auch hier werden verschiedene Strategien der Lernenden sichtbar). Wir lenken dabei die Lernenden auch hier auf die Strategie der Decomposition. Wie könnten wir dabei kleinere, einfach zu lösende Teilprobleme lösen?

c. Wir nehmen die Idee auf, von einer mit einer kleineren überschaubaren Anzahl Stiften zu beginnen.
Zunächst gehen wir davon aus, dass immer Spieler A beginnt und dass immer die bestmögliche Variante gewählt wird.
Beginnt mit 1, dann 2, 3 bis 12 Stiften und zeichnet auf, wo A unter diesen Bedingungen gewinnt, respektive verliert (w = winner, l = looser).

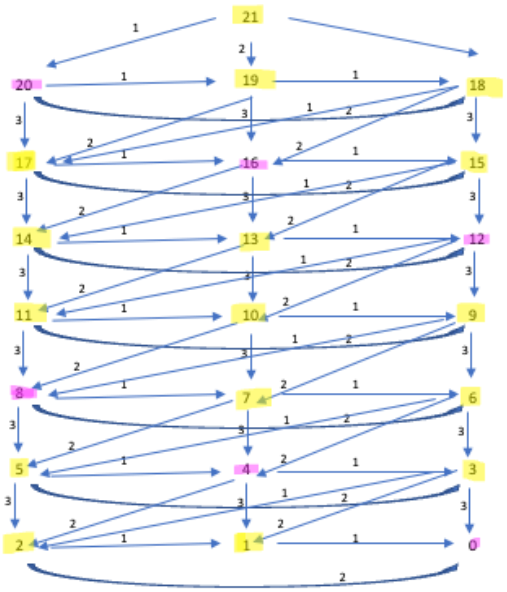
Beispiel:

Pattern Recogintion

a. Findest du ein Muster? Was heisst das für die Gewinnstrategie?
b. Wenn ihr eine Gewinnstrategie habt, dann spielt ihr das Spiel mehrere Male mit Stiften durch und testet die Gewinnstrategie! Haltet eure Beobachtungen fest.
   Die Lernenden werden herausfinden, dass A immer gewinnt, ausser wenn er bei einer durch 4 teilbaren Zahl zum Zug kommt. Wenn sie es mehrmals durchspielen und dabei auch von unterschiedlichen Startmengen ausgehen, werden sie feststellen, dass A keine Gewinnstrategie hat, wenn die Startmenge durch 4 teilbar ist.

Abstraction

a. Gilt die gefundene Gewinnstrategie nun für alle möglichen Spiele? Zeichne für die Startmenge 21 einen vollständigen Baum auf und zeichne die Gewinnstrategie ein.



b. Kannst du daraus eine allgemeingültige Formel ableiten?

T_4 => A verliert
T_8 => A verliert
T_{12} => A verliert
...

T_{4n} => A verliert
$T_{4(n+1)}$ => A verliert

Gewinnstrategie für A:
Gegner dazu bringen, dass er bei T = T4 (n+1) ziehen muss.

Damit ist die erste Teilaufgabe gelöst.

Beweis mittels Induktion:

Induktionsannahme:

Stelle eine Vermutung, respektive eine Hypothese an, wer in welcher Konfiguration gewinnen wird?

$n \in \mathbb{N}_0$

H(n): Wer am Zug ist gewinnt in den Konfigurationen 4n+1, 4n+2, 4n+3
Wer **nicht** am Zug ist gewinnt in allen Konfigurationen 4n

Durch diese Hypothesen sind zwei Induktionsannahmen entstanden!

Induktionsverankerung (IV)

Angenommen deine Vermutungen stimmen für die Zahlen 1, 2, 3, und 4, dann begründe in Worten, warum dies auch für 5, 6, 7, 8 so ist.

Die Schüler stellen fest, dass die Induktionsannahmen für 1, 2, 3 und 4 zutreffen.

Induktionsschritt (IS):

Du hat erkannt, dass deine Annahme für die Zahlen 1, 2, 3, 4 gilt. Überprüfe das nun mit den nächstgrösseren Zahlen. Überlege dir dabei, wie du das in einem mathematischen Term darstellen kannst.

Hier macht es Sinn, die beiden Induktionsannahmen getrennt zu betrachten.

(IA) Wer am Zug ist gewinnt in den Konfigurationen 4n+1, 4n+2, 4n+3
(IV) Wer am Zug ist gewinnt bei 1, 2, 3
(IS) $n \rightarrow n+1$

(IA) Wer **nicht** am Zug ist gewinnt in allen Konfigurationen 4n
(IV) Wer bei 4 ziehen muss, verliert.
(IS) $4n \rightarrow 4(n+1)$

Wer am Zug ist gewinnt bei 4(n+1)+1, 4(n+1)+2, 4(n+1)+3 und verliert bei 4(n+1).
Der Spieler der am Zug ist, nimmt bei 4(n+1)+1 = 4n+5 einen Stift weg. Dadurch kommt der Gegner bei 4n+4 = 4(n+1) zum Zug und verliert.

Anwendung bei veränderten Spielregeln

a. Du hast gesehen, dass du damit eine Gewinnstrategie für das Spiel mit der Wegnahme von 1,2 oder 3 Stiften bewiesen. Wie gilt diese Gewinnstrategie auch noch, wenn du maximal 4 Stifte wegnehmen kannst? Teste dies analog zum obigen Beispiel aus und Beweise deine Gewinnstrategie, wie im vorherigen Beispiel.

(IA) Wer am Zug ist, gewinnt in den Konfigurationen 5n+1, 5n+2, 5n+3, 5n+4
(IV) Wer am Zug ist, gewinnt bei 1, 2, 3, 4
(IS) $n \rightarrow n+1$

(IA) Wer **nicht** am Zug ist, gewinnt in allen Konfigurationen 5n
(IV) Wer bei 5 ziehen muss, verliert.
(IS) $5n \rightarrow 5(n+1)$

Wer am Zug ist gewinnt bei 5(n+1)+1, 5(n+1)+2, 5(n+1)+3, 5(n+1)+4 und verliert bei 5(n+1).
Der Spieler der am Zug ist, nimmt bei 5(n+1)+1 = 5n+6 einen Stift weg. Dadurch kommt der Gegner bei 5n+5 = 5(n+1) zum Zug und verliert.

b. Kannst du die Gewinnstrategie für jede beliebige maximale Anzahl der Wegzunehmenden Stifte verallgemeinern?

$n \in \mathbb{N}_0$
$x \in \mathbb{N}$, x sei die maximal zu entfernende Anzahl Stifte.

(IA) Wer am Zug ist, gewinnt in den Konfigurationen:
(x+1)n+1, (x+1)n+2, (x+1)+3 … (x+1)n+x
(IV) Wer am Zug ist gewinnt bei 1, 2, 3, …, x
(IS) $n \rightarrow n+1$

(IA) Wer **nicht** am Zug ist gewinnt in allen Konfigurationen (x+1)n
(IV) Wer bei x+1 ziehen muss, verliert.
(IS) (x+1)n → (x+1)(n+1)

Die Beweisführung erfolgt analog zum ursprünglichen Beispiel.

Anwendung beim Programmieren

Algorithm Design

a. Wir haben jetzt die Gewinnstrategie erkannt. Nun gehen wir zum **Programmieren** über. Auch hilft uns die Strategie des Computational Thinking:
Welche kleinen, einfacher zu lösenden Teilprobleme kannst du dabei identifizieren?
Mögliche Antworten: Anzeige der Anzahl Stifte (ev. grafisch), Anzeige der Züge, User-Zug, Computer-Zug, Startwert setzen, Beginner auslosen, Usereingabe überprüfen, Anzeige des Gewinners …

b. **Abstraction**: Welche Teilprobleme müssen zwingend gelöst werden, um das Spiel zu spielen. Welche sind eher «nice to have»?
Anzeige der Anzahl Stifte und der Züge im Terminal, User-Zug, Computer-Zug (mit Gewinnstrategie), Startwert (mindestens fix), Gewinner anzeigen, Spiel beenden.

c. **Programmiere** die notwendigen Teile und setze diese geschickt zusammen, so dass du das Spiel spielen kannst. Wenn du damit fertig bist, besprichst du dein Programm mit der Lehrperson und beginnst damit dein Programm weiter auszubauen. Vorgeschlagene Reihenfolge für den Ausbau (Im Beispielprogramm rechts grün beschriftet):
   - Überprüfung der Usereingabe
   - Variable Eingabe des Startwerts
   - Zufällige Startreihenfolge
   - Zugfälliger ComputerZug, wenn keine Gewinnstratege vorhanden
   - Grafische Anzeige der Stifte
   - … eigene Ideen

Mögliche Lösung (so wie sie durch die Schüler*innen aufgrund der Programmierkenntnisse erstellt werden könnte).

NimmSpiel_1.py

```
#Startwert
anzahlStifte= 21                                        #variables Festlegen des Startwerts

#Anzeige der Stifte
def stiftAnzeige(anzahlStifte):                         #Grafische Anzeige
    print ("Es liegen", anzahlStifte, "Stifte auf dem Tisch")
    userZug(anzahlStifte)

#UserZug
def userZug(anzahlStifte):
    nimmSpieler = inputInt ("Gib eine Zahl zwischen 1 und 3 an!")#Kontrolle der richtige Eingabe
    print ("Spieler nimmt",nimmSpieler)
    if (anzahlStifte-nimmSpieler) == 0:
        print("Sieger ist User!")
        return
    computerZug(anzahlStifte - nimmSpieler)

#ComputerZug
def computerZug(anzahlStifte):
    if anzahlStifte % 4 != 0:
        czug = anzahlStifte % 4
    else: czug = 1                                      #zufällige Zahl
    print("Computer nimmt",czug)
    if (anzahlStifte - czug) == 0:
        print("Sieger ist Computer!")
        return
    stiftAnzeige(anzahlStifte - czug)


stiftAnzeige(anzahlStifte)                              #Randomisieren, wer starten darf
print("Game over")
```

(**Nebenbei:** Mit dieser Lösung, kann man können die Schüler*innen mit dem eleganten Einsatz der **Modulo-Rechnung** beim Programmieren aufmerksam machen. Vielen wird diese

Möglichkeit nicht bewusst sein und sie werden die Gewinnstrategie mehr oder weniger umständlich programmieren.)

Diese Unterrichtseinheit kann für die unterschiedlichen Niveaus sehr gut skaliert werden. Hier wurde das Vorgehen für die weniger versierten Schüler*innen skizziert. Gute Schüler*innen können den Auftrag, das Spiel zu programmieren auch selbstständig lösen. Umso wichtiger ist es dann, ihr Vorgehen mit ihnen zu besprechen und das induktive Vorgehen aufzuzeigen.

Damit sollten nun die Schüler*innen in der Lage sein, ein Spiel aus der Geleichen Klasse selbständig bearbeiten zu können:

Das Spiel: "Bachet'sches Spiel"

Regeln:

- Man beginnt mit einer zufälligen Zahl kleiner 30.
- Die Spieler addieren abwechselnd eine selbst gewählte ganze Zahl zwischen 1 und 10 zu dieser Zahl.
- Gewonnen hat der Spieler, der als Erster 100 erreicht.

Decomposition:

- Man beginnt mit einer zufälligen Zahl kleiner 10.
- Die Spieler addieren abwechselnd eine selbst gewählte ganze Zahl zwischen 1 und 10 zu dieser Zahl.
- Gewonnen hat der Spieler, der als Erster 20 erreicht.

Spielt das Spiel mit den obigen Regeln gegen eine Mitschülerin oder einen Mitschüler. Beantworte nachher noch die untenstehenden Fragen.

Kommentar für die Lehrperson:
Das Gegeneinander-Spielen fördert den Gruppengedanken und soll zudem den Spassfaktor erhöhen.

Frage 1: Bei welcher Zahl des Gegners weisst du garantiert, dass du gewinnen wirst? Stelle eine Vermutung auf.
→ Induktionsannahme (IA)

Lösung:
Bei der Zahl 9. Die 9 ist eine Schlüsselzahl. Wenn der Gegner bei 9 dran ist, kann er höchstens auf 19 kommen und muss mindesten 1 wählen und somit auf 10 kommen. Damit kannst du mit dem nächsten Zug sicher gewinnen.

Pattern Recogintion

Frage 2: Nun ändern wir die Regeln ein wenig (siehe Regeln unten).
Bei welcher Zahl des Gegners weisst du garantiert, dass du gewinnen wirst? Stelle eine Vermutung auf.
→ Induktionsannahme (IA)

Regeln

- Man beginnt mit einer zufälligen Zahl kleiner 10.
- Die Spieler addieren abwechselnd eine selbst gewählte ganze Zahl zwischen 1 und 10 zu dieser Zahl.
- Gewonnen hat der Spieler, der als Erster 30 erreicht.

Lösung:
Bei der Zahl 19. Die 19 ist eine Schlüsselzahl. Wenn der Gegner bei 19 dran ist, kann er höchstens auf 29 kommen und somit kannst du mit dem nächsten Zug gewinnen.

Abstraction

Frage 3: Kannst du aus deinen Vermutungen, welche du bei Frage 1 und Frage 2 aufgestellt hast, ein Muster ableiten?
→ Induktionsannahme (IA)

Frage 4: Nun spielen wir das original Bachet'sche Spiel (siehe Regeln unten). Bei welcher Zahl des Gegners weisst du garantiert, dass du gewinnen wirst? Überprüfe, ob deine Vermutung stimmt, indem du gegen eine Mitspielerin / einen Mitspieler spielst.
→ Induktionsverankerung (IA)

Regeln

- Man beginnt mit einer zufälligen Zahl kleiner 30.
- Die Spieler addieren abwechselnd eine selbst gewählte ganze Zahl zwischen 1 und 10 zu dieser Zahl.
- Gewonnen hat der Spieler, der als Erster 100 erreicht.

Lösung:
Bei der Zahl 89. Die 89 ist eine Schlüsselzahl. Der Gegner kann maximal noch auf 99 kommen und somit hat man gewonnen.

Patern Recognition

Analyse

Das Ziel des Spiels ist es, als erstes 100 zu erreichen. Das bedeutet, dass der letzte Zug nur erreicht werden kann, wenn die Zahl zwischen 90 und 99 liegt. Wenn man dem Gegner 89 als Zahl überlässt, kann er das Spiel nicht gewinnen. Die 89 ist dabei eine Schlüsselzahl. Die weiteren Schlüsselzahlen liegen um eine Differenz von 11 auseinander: 78, 67, 56, 45, 34, 23 und 12 (und 1).

Verallgemeinerung
Das Prinzip lässt sich auf modifizierte Regeln anwenden. Bei einem Ziel *z* (im Original 100) und einer Zugbreite von 1 bis *x* (im Original 10) sind die Schlüsselzahlen $z-(x+1)$, $z-2(x+1)$, $z-3(x+1)$, ...

Algorithm Design
Analog zum Spiel «Der letzte Stift gewinnt», können die Schüler*innen nun daran gehen, das «Bachet'sche Spiel» zu programmieren.

"Der letzte Stift gewinnt" vs. "Bachet'sches Spiel"

Decomposition:

Nun hast du die zwei Spiel «Der letzte Stift gewinnt» und das «Bachet'sche Spiel» kennengelernt.

Frage 5: Kannst du Gemeinsamkeiten zwischen den beiden Spielen entdecken?

Lösung:

- **Bei beiden Spielen geht es darum, ein gewisse Anzahl Stifte zu erreichen (Ziel). Im ersten Spiel geht es darum, durch Subtraktion das Ziel 0 zu erreichen. Im zweiten Spiel muss versucht werden, durch Addition das Ziel 100 zu erreichen.**
- **Bei beiden Spielen zielt die Gewinnstrategie darauf ab, den Gegner dazu zu bringen, bei einer «Schlüsselzahl» ziehen zu müssen.**
- **Die Schlüsselzahlen beim «Der letzte Stift gewinnt» liegen bei:**
  $z+(x+1)n$

  wobei:
  $n \in \mathbb{N}_0$
  x ∈ ℕ, x sei die maximal zu subtrahierende Anzahl Stifte.
  $z \in \mathbb{N}_0$**, z sei das Ziel, hier auf 0 Stifte**

- **Beim „Bachet'schen Spiel" werden die Schlüsselzahlen wie folgt ermittelt:**
  $z-(x+1)n$

  wobei:
  $n \in \mathbb{N}_0$
  x ∈ ℕ, x sei die maximal zu addierenden Anzahl Stifte.
  $z \in \mathbb{N}_0$**, z sei das Ziel, hier auf 100 Stifte**