Vorkurs Informatik (D-ITET)

Malte Schwerhoff

September 2018

http://lec.inf.ethz.ch/itet/informatik0/2018/

3. Exkursion: Monte-Carlo-Simulation

Zahlenraten durch Würfeln

- Zahl zwischen 1 und 6 durch Würfeln „erraten“
- Frage: Wie oft würfeln um mit ≥ 95% richtig zu liegen?
- Mathematik:
  - Chance, mit n Würfeln *nicht richtig* zu liegen: $\frac{5}{6} \cdot \frac{5}{6} \cdot \frac{5}{6} \cdot \ \cdots \ \cdot \frac{5}{6} = \left(\frac{5}{6}\right)^n$
  - Chance, mit n Würfeln *richtig* zu liegen: $1 - \left(\frac{5}{6}\right)^n$
  - $\mathrm{n} = 6 \quad \rightarrow \quad 66\%$
    $10 \rightarrow 83\%$
    $13 \rightarrow 90\%$
    $16 \rightarrow 94\%$
    $17 \rightarrow 95\%$

Monte-Carlo-Simulation

- Analytisch unmöglich/schwierig lösbare Probleme, z.B. Wetterprognose, Neutronenfluss im Kernreaktor ...
- ... mittels *wiederholter Zufallsexperimente* simulieren ...
- ... und so *Lösungen numerisch annähern*
- Mathematik: Gesetz der grossen Zahlen

Monte-Carlo-Simulation für's Zahlenraten

- Wahrscheinlichkeit, Zahl durch Würfeln zu erraten, mittels Monte-Carlo-Simulation numerisch annähern

- Vorgehen:
  1. Eine Variablen, E: Anzahl Erfolge (Zahl nach n Würfen erraten)
  2. Experiment: Zahl wählen, n mal Würfeln, falls Erfolg dann merken
  3. Experiment V Mal ausführen (für grosses V)
  4. Wahrscheinlichkeit, mit n Würfeln richtig zu liegen: $\frac{E}{V}$

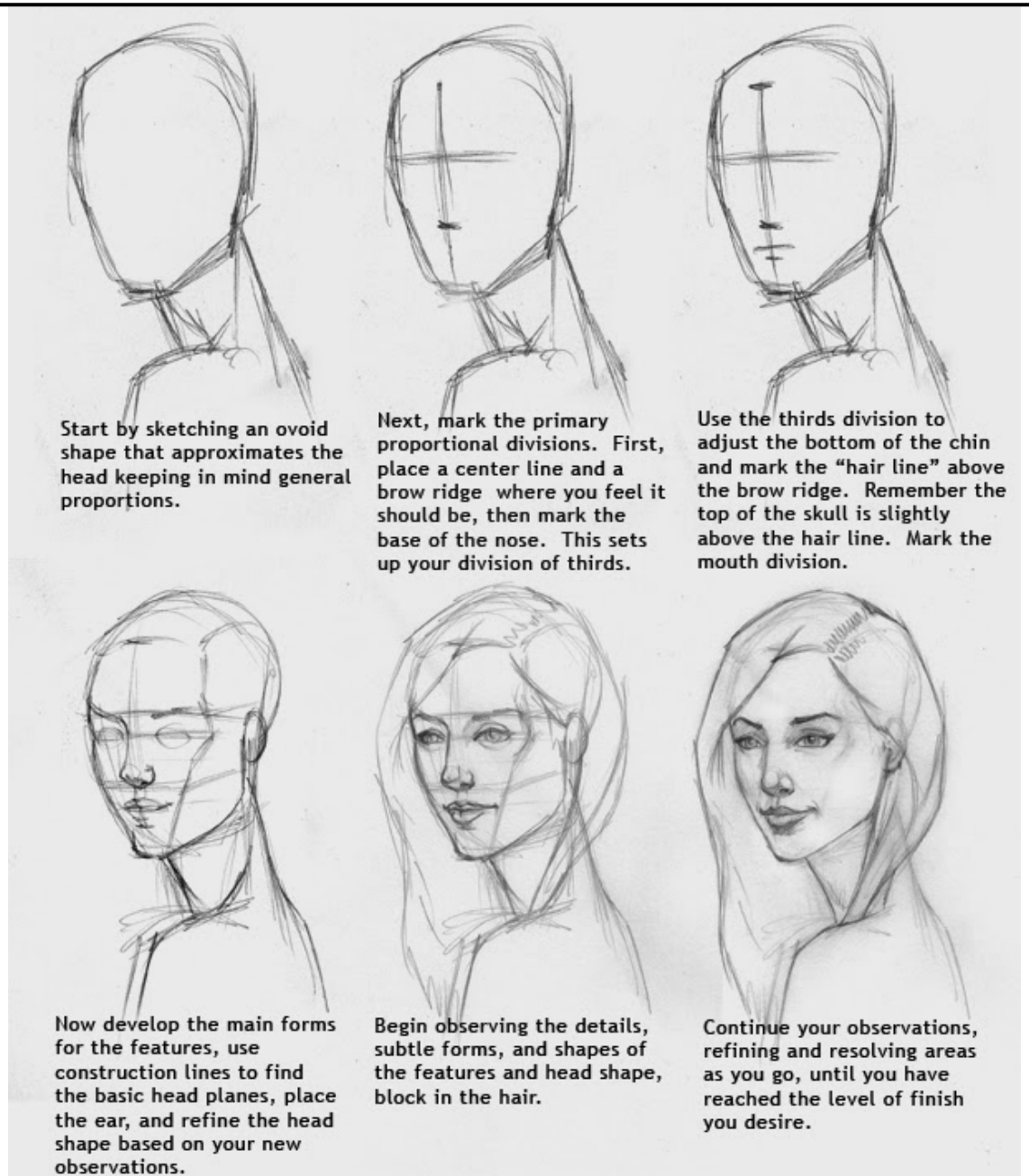
Monte-Carlo-Simulation für’s Zahlenraten

```
#include <iostream>
#include "project.h"

int main() {
    int attempts = 17;
    int successes = 0;
    int experiments = 100; // Choose value, run several times, compare obtained output

    // Repeat the experiment (number guessing) sufficiently often
    for (int experiments_made = 0; experiments_made < experiments; experiments_made += 1) {
        int number_to_guess = choose_a_number(6); // Choose a number from [1,6]

        // Throw the dice at most "attempts" times
        for (int attempts_made = 0; attempts_made < attempts; attempts_made += 1) {
            int guess = choose_a_number(6); // Make a random guess

            if (guess == number_to_guess) {
                successes = successes + 1; // Record the successful guess …
                attempts_made = attempts;  // … and stop the current experiment
            }
        }
    }

    std::cout << (successes * 100.0 / experiments) << "%\n";
}
```

4. Projekt 2: Hangman

sowie *Schrittweise Verfeinerung*
als Implementierungsstrategie

Schrittweise Verfeinerung



http://drawpaint-sg.blogspot.com/2014/07/learn-how-to-draw-face.html

Hangman als Pseudocode

Setup: Choose the word to guess

Game:

- Let the player guess repeatedly
- Until too many incorrect guesses have been made
- Or until the word has been guessed

Hangman als Pseudocode: Schrittweise Verfeinerung

Setup: Choose the word to guess

Game:

Let the player guess repeatedly

Until too many incorrect guesses have been made

Or until the word has been guessed

Hangman als Pseudocode: Schrittweise Verfeinerung

Choose WORD to be some English word (with N characters, e.g. N = 5)

Initialise UNCOVERED to be a sequence of N underscores (e.g. _____)

Game:

    Let the player guess repeatedly

    Until too many incorrect guesses have been made

    Or until the word has been guessed

Hangman als Pseudocode: Schrittweise Verfeinerung

Choose WORD to be some English word (with N characters, e.g. N = 5)

Initialise UNCOVERED to be a sequence of N underscores (e.g. _____)

Game:

Let the player guess repeatedly

Until too many incorrect guesses have been made

Or until the word has been guessed

Hangman als Pseudocode: Schrittweise Verfeinerung

Choose WORD to be some English word (with N characters, e.g. N = 5)

Initialise UNCOVERED to be a sequence of N underscores (e.g. _____)

Allow at most A failed attempts (incorrect guesses)

Game:

Get the next guess G (e.g. L)

If G occurs in WORD: …
Otherwise: Record a failed attempt

If all attempts have been used up: Output ☹ and stop

Or until the word has been guessed

Hangman als Pseudocode: Schrittweise Verfeinerung

Choose WORD to be some English word (with N characters, e.g. N = 5)

Initialise UNCOVERED to be a sequence of N underscores (e.g. _____)

Allow at most A failed attempts (incorrect guesses)

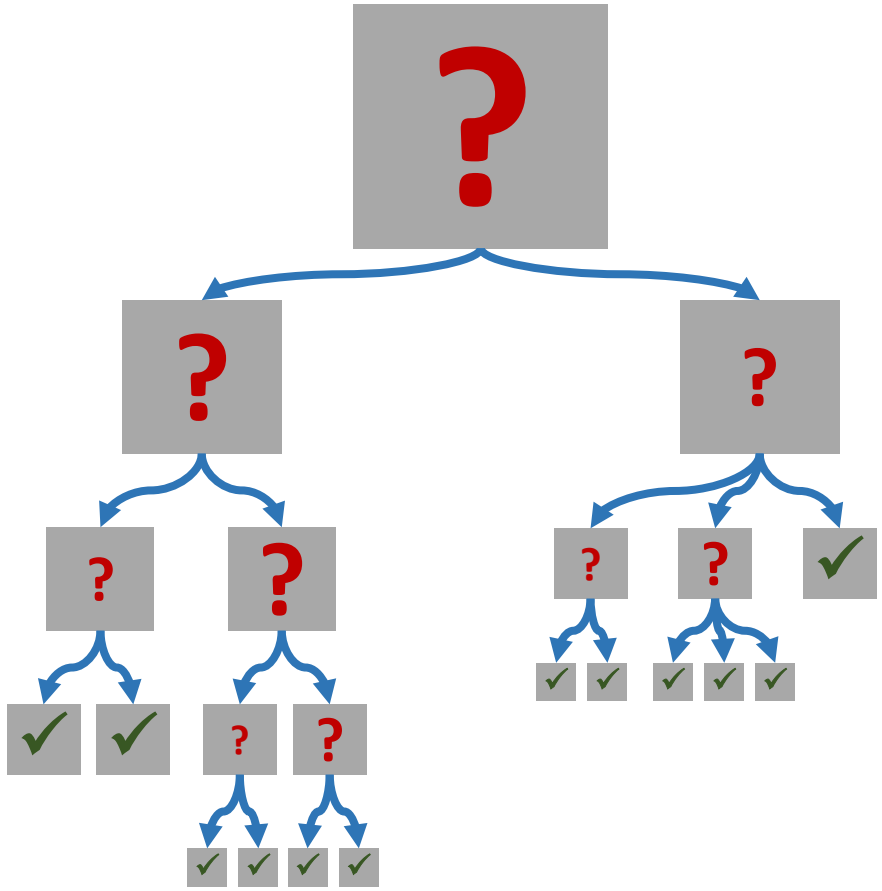
Play the game. In each round:

- Output UNCOVERED and how many attempts are left (e.g. H E _ _ O, 3)
- Get the next guess G (e.g. L)
- If G occurs in WORD: Uncover all characters G in UNCOVERED (e.g. H E L L O)
  Otherwise: Record a failed attempt
- If all attempts have been used up: Output ☹ and stop
- If all characters in UNCOVERED have been uncovered: Output ☺ and stop

Schrittweise Verfeinerung

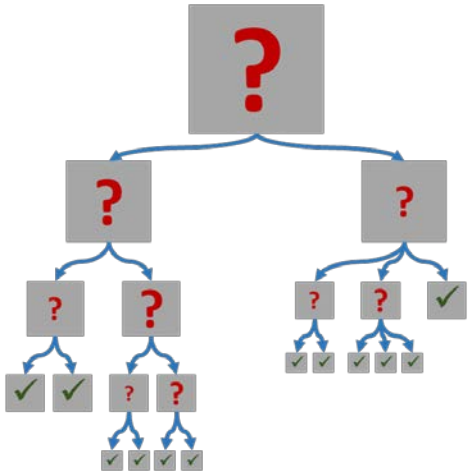
- Schrittweise Verfeinerung (stepwise refinement) ist eine *iterative Top-Down-Strategie*:

  1. Zerlege Aufgabe (Hangman, Wörterbuchsuche) in Teilaufgabe/-schritte
  2. Wähle eine Teilaufgabe aus → Verfeinern durch weiteres Zerlegen (gehe zu Schritt 1) und hinzufügen von Details
  3. Stoppe, wenn alle Teilaufgaben vollständig ausgearbeitet sind: alle Details vorhanden, keine offenen/ungelösten Teilaufgaben



Schrittweise Verfeinerung

- Vorteile:
  - Langsamer Einstieg
  - Systematisches, strukturiertes Vorgehen
  - Komplexität/Detailniveau steigt schrittweise an
  - Einfacher, den Überblick zu behalten



- Aber: Ist eine *Strategie*: Prinzip verstehen reicht nicht, Anwendung erfordert Übung

- Nicht Informatik-spezifisch, sondern allgemein anwendbar

Mögliche Programmverfeinerung

- „Skizze“ des Programs als Pseudocode: grobe Aufteilung, graduell verfeinern
- Pseudocode als Anfangskommentare nutzen, diese dann graduell zu Code verfeinern
- Mit den Variablen anfangen: Welche Daten brauche ich?
- Noch nicht realisierte, generelle Funktionalitäten durch fixe Werte simulieren um den Code laufenzulassen und testen zu können, z.B.:

```
std::cin >> word;// Get arbitrary word
uncovered = ???  // Replace characters
                 // with underscores
std::cin >> guess;
if (/* guess occurs in word */) {
    ... // Does this line work?
}
```

```
word = "HELLO";       // Fixed
uncovered = "_____";  // Fixed

std::cin >> guess;
if (/* guess occurs in word */) {
    ... // Does this line work?
}
```

- ...