Algorithmic Thinking and Structured Programming (in Greenfoot)

Teachers:
Renske Smetsers-Weeda
Ana Tanase
Sjaak Smetsers

Course

- Algorithmic Thinking:
  Solving computational problems

- Structured Programming:
  Object Oriented programming
  in Java using the
  Greenfoot environment

Not just with a PC,
Also with pen-and-paper

Course expectations

- Moral 1: **Don’t give up**
  - programs usually don’t run perfectly the first time, you will make mistakes
  - expect to make mistakes
  - learn from them
- Moral 2: **Work smart**
  - think ahead (like an architect)
  - build strong and sturdy
  - reuse your solution in following exercises (instead of rebuilding)

Introduction

- 3 teachers:
  - 2 teachers / master students
  - 1 lecturer RU

- What brings us here?
  - We love computer science education
  - Scientific research on learning computer science

Introduction

- HAVO / VWO?
- TTO?
- Programming experience?

Organization

- Masterclass
- Beginners course
- Course is in English
  - If English becomes a problem, please let us know.
  - Try to speak as much English as possible!
- 14 lessons: 2 hours a week
- Homework
  - At least 1 hour a week
  - Magister (deadline: Wednesday 8:30)

Final Grade

- Homework: must be a pass
- 3 Quizes: each 10% of final mark (Dec, Jan, Feb)
- Test: 70% of final mark (beginning of April)

- Extra credit (max 10%):
  - Outstanding work on Dodo's Race (final project)
  - Advanced students who complete extra Sokoban project (assignment 8)

Today's Lesson plan

- 10 min Introduction
  - Course goal & expectations
  - Today's lesson goal
- 35 min Computational Thinking
- 10 min Greenfoot introduction
- 50 min Get Dodo to work: Assignment 1
- 10 min Wrapping up
  - Saving work
  - Plenary reflection

Today’s Lesson

- Computing is about….

  … solving problems (for people).

- Problem solving concepts:
  - Algorithms
  - Efficiency

21st century skill: computational thinking

- **Working in a structured manner:**
  - Breaking problems down into subproblems
  - Design, solve and test solutions to subproblems
  - Combing these (sub)solutions to solve problem
- **Analyzing** the quality of a solution
- **Reflecting** about the solution chosen and proces
- **Generalizing** and re-use of existing solutions

Locked-in syndrome

- Patient is ‘locked-in’ body:
  - Totally paralyzed
  - All mental abilities intact
  - But can only blink

  - It can happen to anyone, suddenly (stroke)
  - Doctors can’t do much
  - Rehabilitation (if possible) up to 20 years

- Can you come up with a way to communicate?

Example: count blinks

A: 1 blink
B: 2 blinks
C: 3 blinks
...
Z: 26 blinks

Algorithm: count blinks

A: 1 blink
B: 2 blinks
C: 3 blinks
…
Z: 26 blinks

Algorithm: precise description of solution:

which steps (and in which order)

This algorithm has 2 parts:

- The patient: blinking (correct) number of times
- The helper:
  - Counts number of blinks
  - Writes letter down when blinking stops

Improved algorithm

Improved algorithm:

- The helper: Reads out letter
- The patient: Blinks when correct
- The helper: Writes down letter

Locked-in: finding solutions

5 minutes:

- Get in pairs
- Decide on a better way to communicate
- Can you come up with a solution that really works?
- Try it out!

  Communicate the message “JAVA” to each other
- Write down:
  - The algorithm…
  - It is better because….
  - When does it (not) work? Problems? Challenges?

Locked-in: sharing solutions

Describe:

- The algorithm
- Why is your solution better?
- Problems / Challenges?

Algorithm: count blinks

A: 1 blink
B: 2 blinks
C: 3 blinks
…
Z: 26 blinks

Problems/ Challenges:

- Word/sentence end: punctuation
- Blink by accident?
- LOTS of blinks (for example: puzzel)
- What to do if you miscount?
- Numbers and smilies?

Efficiency: examining solutions

- How long does it take? How to measure?
  - Don’t use time (not stable)
  - Use how much work needed: number of blinks/Q’s

- Best case scenario: What is the fewest blinks/Q’s needed?
- Worst case scenario: What is the most blinks/Q’s needed?

- Example for a 4-letter word:
  - Best case: AAAA is 4x1=4 blinks
  - Worst case: ZZZZ is 4x26=104 blinks
  - Average: 54 blinks

A: 1 blink
B: 2 blinks
C: 3 blinks
…
Z: 26 blinks

Locked-in: examine your solution

- Best case scenario: fewest blinks needed?
- Worst case scenario: most blinks needed?

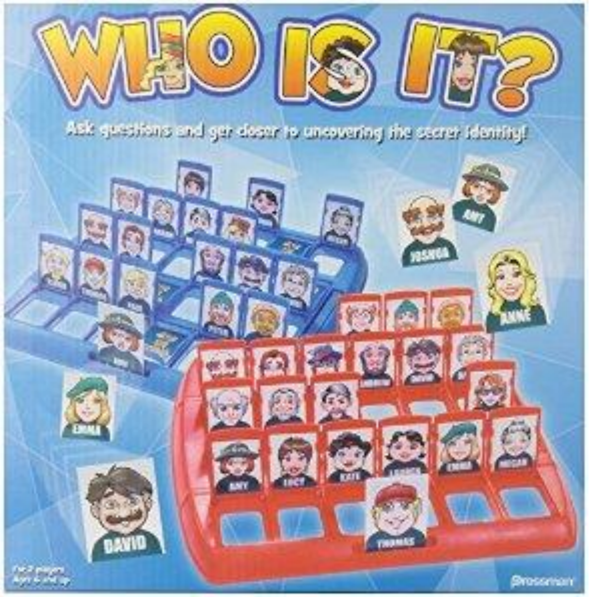
- Example for a 4-letter word:
  - Best case: AAAA is 4x1=4 blinks
  - Worst case: ZZZZ is 4x26=104 blinks
  - Average case: 54 blinks

A: 1 blink
B: 2 blinks
C: 3 blinks
…
Z: 26 blinks

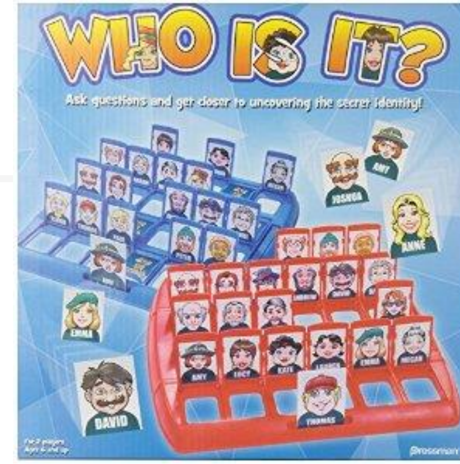
2 minutes: Determine how well your solution works in best and worst case.

Possible improvements

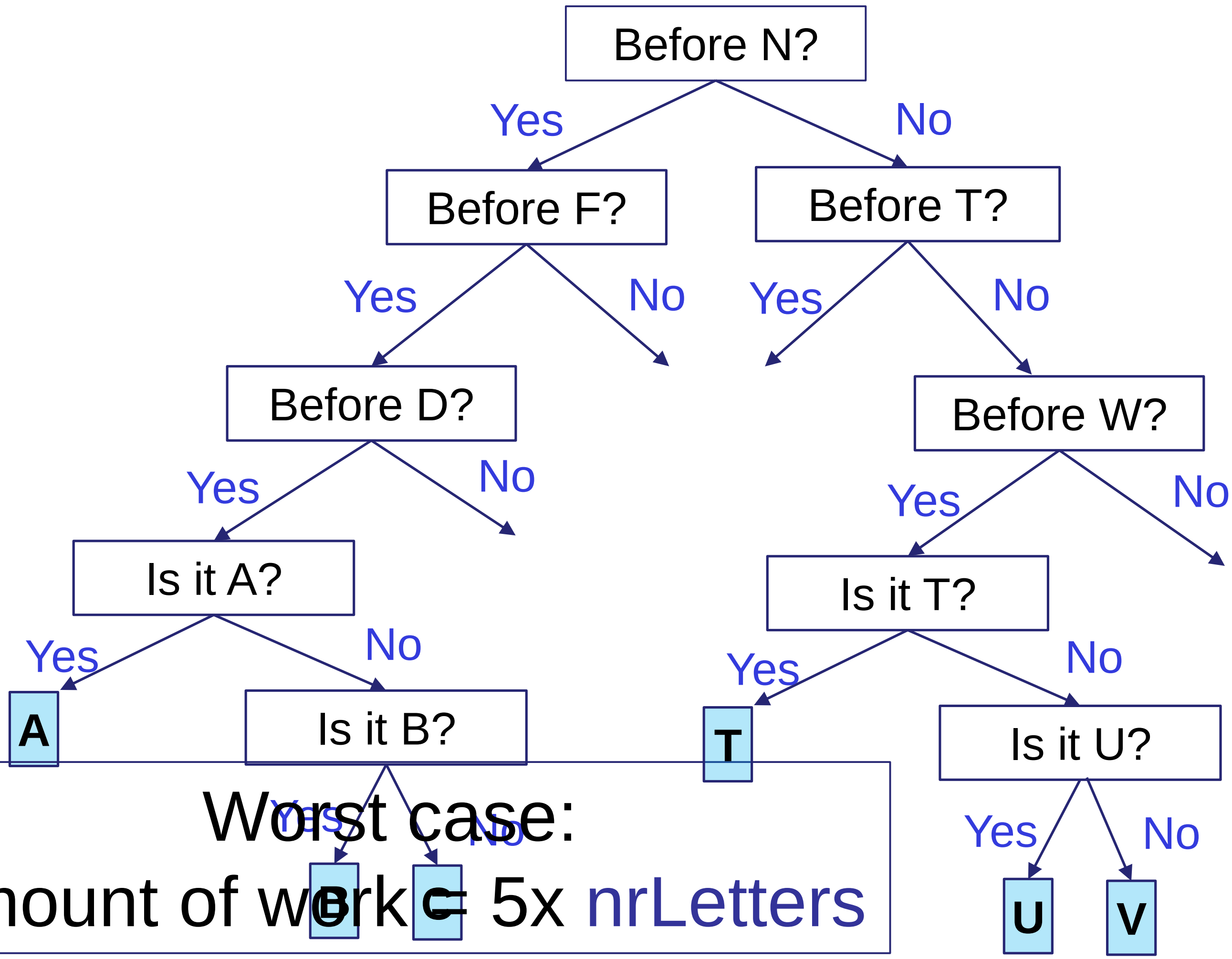
- More modes: short/long blinks
- Word prediction:
  - antel -> antelope
  - T9 (only 10 possibilities)
- Most frequent letters first (Huffman coding)
- Dividing possibilities in half
  - Man / woman
  - Hair / bald
  - Glasses / no glasses



Transfer of a solution



Using Who-Is-It strategy for Locked-in solution:



Worst case:
Amount of work = 5x nrLetters

Search algorithms

Worst–case:

- First algorithm: work = 26 x nr letters
- Improved algorithm: work = 5 x nr letters

Imagine Google searching through data:

- First algorithm: work = 1 million steps
- Improved algorithm: work = 20 steps

Locked-in: summing up

- We developed an algorithm
  - Precise steps that both people agree on to communicate
- We evaluated algorithms
  - How much work is needed
  - Limits: how good/bad it could possibly be

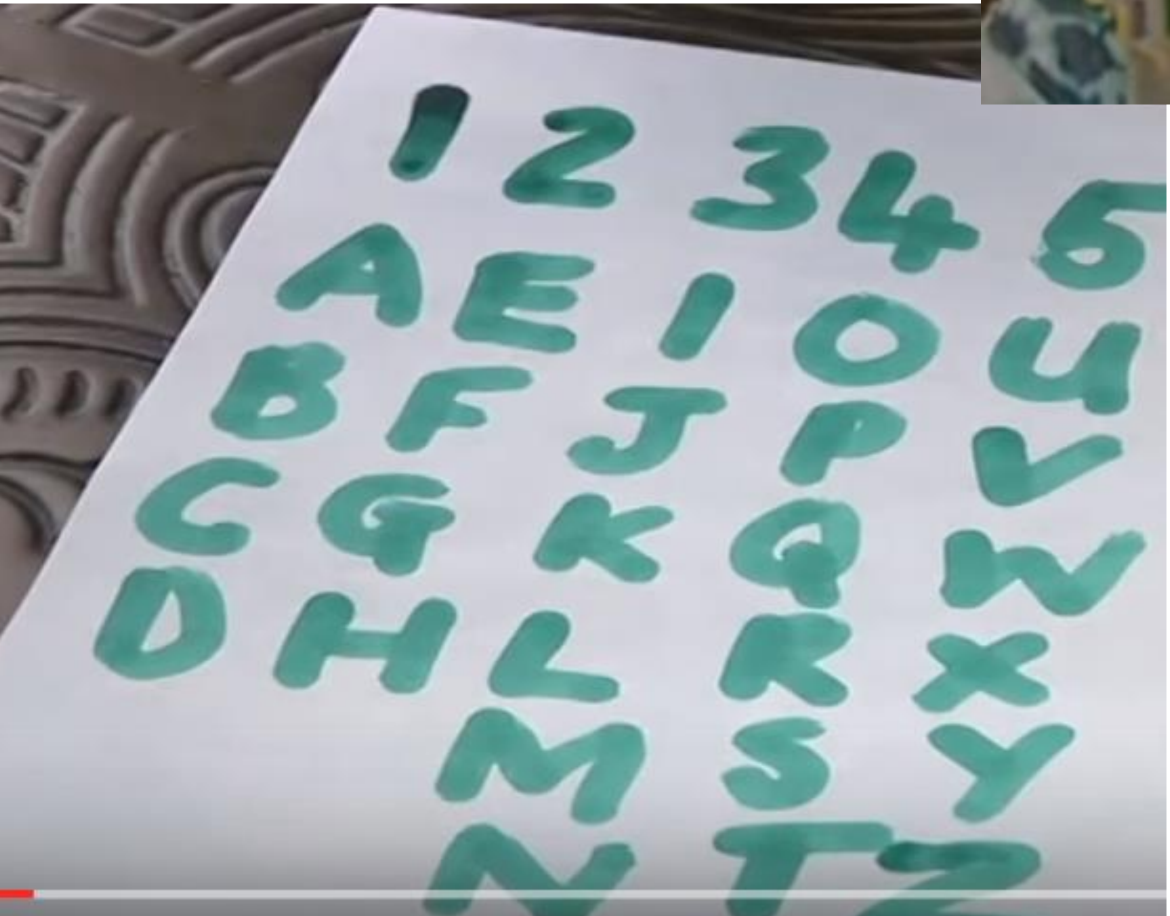
- Problem similar to how 2 computers communicate over a network: they can only send 0s and 1s

Locked-in: real solutions

- 0:00 – 0:46
- https://www.youtube.com/watch?v=WQIWc3uE4LU

- 1.25 – 1.55
- https://www.youtube.com/watch?v=A3uEMyVnThI

Other real solutions



Computational thinking

- Finding creative solutions
- Reuse solutions from other problems
- Describing steps precisely
- Before building a solution, think about:
  - Efficiency
  - Assumptions / conditions
  - Does it solve the problem? (final situation)
- It's not just about computers…

Computing is about… solving problems for people

Greenfoot and Java

Greenfoot environment:

- Visualize and test your algorithms
- Gives immediate feedback

- You write real Java code

Mimi the Dodo

- Demo

Where we are going

And the end of the course you will be able to:

- program in Java
- use Java docs
- reuse other’s work

… and make just about anything your creative mind can think of!

Where we are going

Final assignment: Dodo's race.

Who can come up with the best algorithm and make the smartest Dodo?

How?

1) Algorithmic Thinking

2) Structured Programming

} Course Goals

Assignments: how to work

- Read the theory
- Do the exercises (all code and ‘IN’ must be handed in)

- Work in pairs (same strength)
- First read and think about answer individually
- Discuss answer together
- Switch ‘driver’ every exercise (so, about every 10 min)

- Expect to get stuck occasionally
- Stuck? Explain to your partner what you are trying to do and why you think it doesn’t work
- Can’t figure it out together => raise your hand

Pair programming

- Why?
  - Discuss problems together
  - You can help and learn from each other
  - Less mistakes, smarter solutions, faster
  - More fun
- How?
  - Together: discuss algorithm, debug
  - Driver: types (code & answers to hand IN questions)
  - The other: thinks about strategy, draws flowcharts, reviews code, advises, writes answers to questions
- Switch ‘driver’ every exercise or 15 minutes

Assignment 1

1. Get into pairs
2. Open (Word) document for hand‘(IN)’ questions
3. Other questions: jot down on instruction paper
4. Make sure you have a place to save your work
5. Download and unzip the scenario at http://www.cs.ru.nl/~S.Smetsers/Greenfoot/Kandinsky/

- **Hand in on Magister before Wednesday 8:30**

Wrapping up

Save your work! Discuss how/when to finish off and who will turn it in.

Homework:

- Finish Assignment 1: until and including 5.4
- Instructions on saving and handing in: 7 and 8
- **In Magister before Wednesday 8:30**

Reflection:

- What did you learn today?

Any other questions?