# Games

Musa Al-hassy

McMaster University
alhassm@mcmaster.ca

January 9, 2017

**Programming requires a great investment in thinking from the get-go!**

"Thinking about thinking is the hardest sort of thinking there is, which makes you think." – Philomena Cunk's Moments of Wonder - Philosophy

# Closing the curve

Consider the 2-player game that needs a grid-paper and pencils, with game play:

1. The players alternate moves.
2. Player 1 moves by drawing | or – between two adjacent dots; player 2 moves by drawing a dotted line between two adjacent dots.
3. A player may not write over another players move.
4. Player 1 wins if he can get a completely closed curve.
5. Player 2, because he goes second, has an easier takes: he wins if he can stop player 1 from getting a closed curve.

Question: is there a strategy that guarantees a win for either player, no matter how big the board is? If so, what is it?

# Go!

What about a wining strategy for the 2-player game that again needs a grid-paper but uses black and white coffee beans, with game play:

1. The players alternate moves.
2. Each player moves by placing a bean of their own colour onto an unoccupied intersection point.
3. A group of stones of an identical colour are removed from the board by the opponent and are considered 'points' for them if they can successfully surround all *adjacent* intersections to that group with their own colour.
4. A player's total score is the sum of the area —intersection points— they have completely surrounded and the points they obtain from any captured enemy beans. The player with the greater score wins.
5. The game concludes when both players 'pass' —i.e., agree that there are no more profitable moves to make.

( The Interactive Way To Go ; Hikaru no Go )

# Numbers not divisible by two just seem odd to me!

## Expected Properties

1. even 0
2. odd 1
3. even $n \not\equiv$ odd $n$ —every number is *either* even or odd
4. $\neg$even $n \equiv$ odd $n$
5. $\neg\neg p \equiv p$ —double negation

What about

$$\text{even } (n + m) \equiv \text{even } n \equiv \text{even } m$$

where '$\equiv$' is equality on the Booleans —note that it is the only associative equality operation!

# Prison Break!

- A merciful king tells his 100 prisoners that tomorrow they will play a game where the prisoners will be lined up in a row, the first prisoner at the front and the last at the back.
- Unsurprisingly, each prisoner will be able to see all the prisoners in-front of them.
- The kind king will place a black or white hat on each prisoner, randomly, and whomever guesses the colour of their hat correctly earns their freedom *immediately*; otherwise earns an *immediate* death. After all the hats are placed, he begins with the prisoner at the end.
- The prisoners can only say "black" or "white", any other word or gesture earns them immediate death. They are in an open field, so their is no shenanigans for utilizing walls or mirrors.

Question: is their a 'winning-strategy' for the prisoners? That is, can they concoct a plan where-by all of them, or nearly all of them, get out free?

# The Poor Perplexed King!

- Upset with how (nearly) all the prisoners escaped; the kind king asks how they cleverly figured it out. Fear of playing the game again, the prisoners refused to answer.
- The kind king take on an ambitious gamble with the prisoners.
- He proposes they play again and if they succeed, he offers to give them each great riches. If more than one of them loses, then they promise to tell him how they won the first game.
- The king states he'll use five different colours this time around!
- They accept!!

Question: what is the prisoners' 'winning-strategy'? What if the king uses $N$-colours instead?

# Take!

What winning strategies are there in the 2-player game "Nim"?

1. There are 2-heaps of black beans, with $n_1$ and $n_2$ beans in each heap.
2. Players alternate turns.
3. A player moves by selecting a non-empty heap and removing any non-zero number of beans from it.
4. The game ends when all heaps are empty.
5. The player who removes the last bean wins.

Question: what if there are $K$-many heaps?

# Game Play

We play a 1-player game whose pieces involve a coffee can containing some black beans and white beans and whose game-play is as follows:

1. Randomly select two beans from the can.
2. If they are the same colour, throw them out, but put another black bean in. ( Enough extra black beans are available to do this. )
3. If they are different colours, place the white one back into the can and throw the black one away.

# Game Analysis

- Game-state: what data characterizes a particular position in the game?
- Progress: how does the game-state change? In particular, what value is decreased each turn?
- Termination: does the game ever actually finish? If so, how many beans are left in the can when the game finishes?

# Invariants

Post-condition: what can be said, if anything, about the colour of the final bean based on the number of white beans and the number of black beans initially in the can?

Perhaps there is a *a simple property of the beans in the can that remains true as beans are removed* and that, together with the fact that only one bean remains, can give the answer. Since the property will always be true, we will call it an *invariant* —since it does *not vary* as the game progresses.

# Know the properties!

These problems have extremely simple solutions, but the solutions are extremely difficult to find by simply trying test cases. The problems are easier if one looks for properties that remain true. And, once found, these properties allow one to see in a trivial fashion that a solution has been found.

## Principle

Know the properties of the objects that are to be manipulated by a program. The more properties you know about the objects, the more chance you have of creating an efficient algorithm.

# The outer parts

1. Formalise '*G*ivens' and '*R*equireds' of the problem.
2. Obtain an invaraint $P$ and initalise the varaibles to make it true.
3. Bridge from invaraint to post-condition: solve for $B$ in
$$P \wedge \neg B \implies R$$
4. If $\neg B$ holds then we're done, otherwise we construct a loop to obtain it.

## So far

$\{G\}$
intialisation
$\{\text{invaraint } P\}$
; **do** $B \to$ ??? **od**
$\{P \wedge \neg B\}$
$\{R\}$

Okay, great, we don't know what's in the loop and that's fine, that's not a problem right now —maybe a problem for future-us, not current-us.

# The loop body

1. Solve for a "bound function" $bf$ in $P \wedge B \implies bf > 0$.
2. Make progress towards termination: solve for $S$ in

$$\{bf = c\}\ S\ \{bf < c\}$$

   where $c$ can be thought of as any "candidate number of iterations remaining".
3. Ensure that such a program $S$ *maintains* our invaraint!

$\{G\}$
intialisation
$\{$invaraint $P$ ; bound $bf\}$
; **do** $B \to \{P \wedge B \wedge bf = c\}\ S\ \{P \wedge bf < c\}$ **od**
$\{P \wedge \neg B\}$
$\{R\}$

# References

*Can you solve the prisoner hat riddle?* by Alex Gendler

`https://www.youtube.com/watch?v=N5vJSNXPEwA`

Animated problem statement and solution to a less-violent hat problem using 10 prisoners.

*Introductory Combinatorics* by Richard Brualdi

`https://www.pearsonhighered.com/program/Brualdi-Introductory-Combinatorics-5th-Edition/PGM17775.html`

Discusses the game of Nim, and its solution, in the introductory chapter.

LADM

See course site

Among other things, mentions the "approach to program construction"; Along with a host of "principles and heuristics" to remember and use.

*Theoretical Introduction to Programming* by Bruce Mills

A Springer Text

Discusses the coffee can problem on page 129