Control Issues & Refactoring
Power of De Morgan’s Theorem

the complement of the union of two sets is the same as the intersection of their complements; and
Power of De Morgan’s Theorem

the complement of the union of two sets is the same as the intersection of their complements; and

the complement of the intersection of two sets is the same as the union of their complements.
De Morgan’s Law in practice

- Simplify expression to avoid double negatives

- Instead of:

  ```java
  if (!printer.hasPower() && !printer.hasPaper()) {
  }
  ```

- Write:

  ```java
  if (!printer.hasPower() || printer.hasPaper()) {
  }
  ```
Other good rules

- Encode complex Boolean expressions in methods
  - Naming documents the meaning of the expression
  - Even if the method is only called in one place
    \[
    \text{boolean isSMS} = \text{isSMS}(\text{-1, -1, -})
    \]
- Use braces and parentheses to improve clarity
  - Helps quickly parsing expressions
Dijkstra’s Structured Programming

Which does not belong?

A) Iteration
B) Recursion
C) Selection
D) Sequence
Code Complexity Metrics

- Start with 1 for straight-line path through the code
- Add 1 for each: if, while, for, &&, ||, or equivalent
- Add 1 for each case in a switch statement

1-5 Routine is fine
6-9 Look to simplify
10+ Refactor into multiple routines

- Applicable for reasoning about nesting depth
Refactoring Examples

- Code is duplicated
- Routine too long / deeply nested
- Poor cohesion
- Changes require parallel modifications to multiple classes
- Routine uses more features of another class than its own
- Primitive data type is overloaded
- One class is overly intimate with another
- Data members are public
- Comments are used to explain difficult code
- Global variables are used
A* search Algorithm

- [https://en.wikipedia.org/wiki/A*_search_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)
Grid input

```json
{
    "dimension": 10,
    "start": {"x": 0, "y": 0},
    "end": {"x": 8, "y": 8},
    "obstacles": [
        {"x": 1, "y": 0},
        {"x": 1, "y": 1},
        {"x": 1, "y": 2},
        {"x": 1, "y": 3},
        {"x": 2, "y": 3},
        {"x": 3, "y": 3},
        {"x": 4, "y": 3},
        {"x": 5, "y": 3},
        {"x": 5, "y": 4}
    ]
}
```
Key Principles of Software Design

- Make intention clear
  - The purpose of each aspect of code is easily understood

- Encapsulation / Data hiding
  - bind together data and functions that manipulate the data, to keep both safe from outside interference and misuse.

- Cohesion (Strong cohesion is good)
  - degree to which the elements of a module belong together

- Coupling (Loose coupling is good)
  - the degree of interdependence between software modules
To Dos for Wed night

- Read Ch. 5 (Design in Construction)

- Start working on Astar for grids
  - Parse JSON
  - https://en.wikipedia.org/wiki/A*_search_algorithm