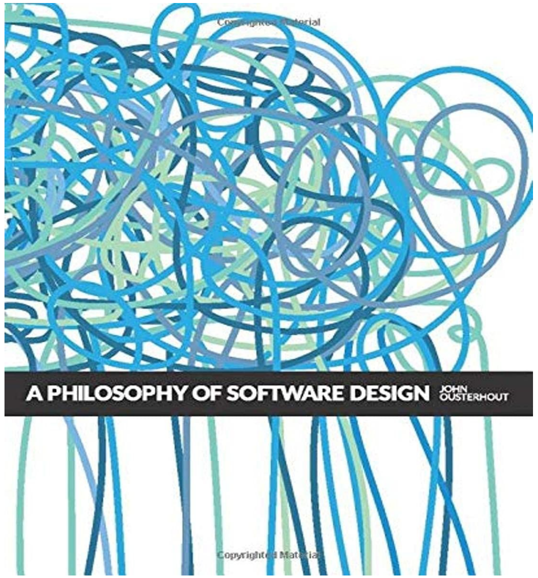


Lecture 31 (Software Engineering III)

Managing Software Complexity II

CS61B, Spring 2025 @ UC Berkeley

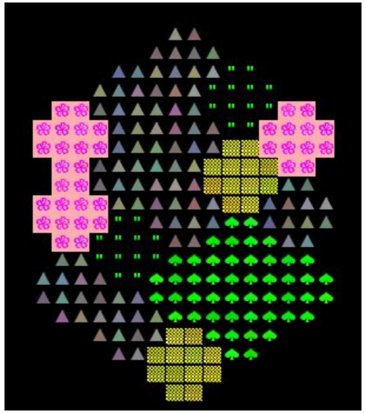
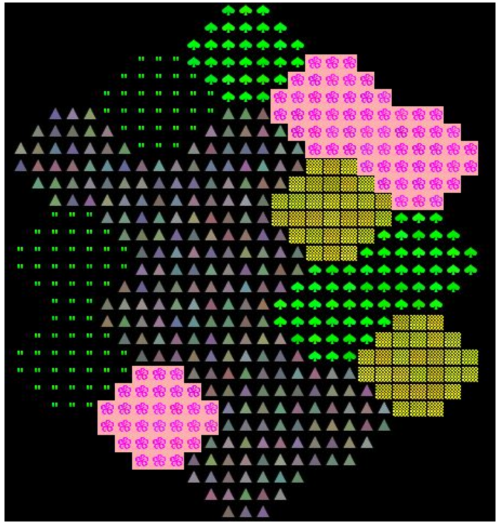
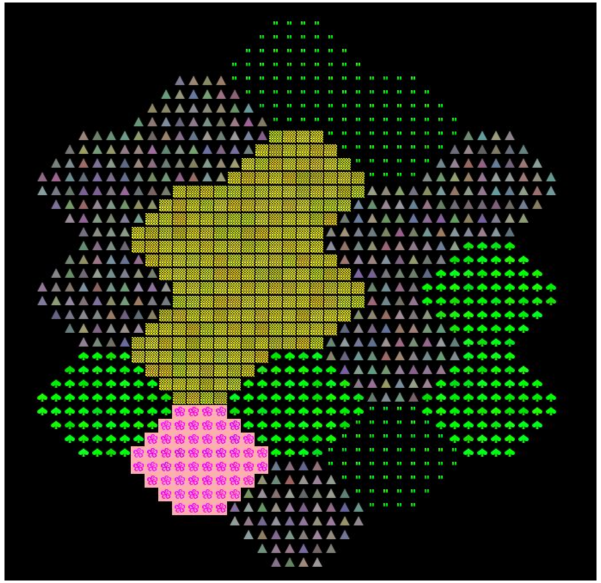
Slides credit: Josh Hug

HexWorld Continued

Lecture 31, CS61B, Spring 2025

Review

Last week we started implementing a world generator that could generate the worlds below.

Review

Based on your input, we created a Hexagon class with:

- Instance variables int x, int y, int s, Tile t
    - x: x coordinate of the top left corner
    - y: y coordinate of the top left corner
    - s: size of the hexagon
    - t: tile of the hexagon (e.g. flower, mountain, etc.)
- void addHexagon(TETile[][] world)
    - Adds the given hexagon to the passed-in world

In our main class, we also had:

- static void method drawHorizontalLine(TETile[][] world, int x, int y, int s, TETtile t)

Review

Main method so far:

```java
public static void main(String[] args) {
    TERenderer ter = new TERenderer();
    ter.initialize(60, 60);
    TETile[][] world = new TETile[60][60];
    for (int x = 0; x < 60; x += 1) {
        for (int y = 0; y < 60; y += 1) {
            world[x][y] = Tileset.NOTHING;
        }
    }
    drawHorizontalLine(world, 10, 10, 7, Tileset.MOUNTAIN);

    Hexagon h = new Hexagon(10, 10, 4, Tileset.TREE);
    h.addHexagon(world);
    ter.renderFrame(world);
}
```

Initialize graphics

Create empty world

Review

Main method so far:

```
public static void main(String[] args) {
    TERenderer ter = new TERenderer();
    ter.initialize(60, 60);
    TETile[][] world = new TETile[60][60];
    for (int x = 0; x < 60; x += 1) {
        for (int y = 0; y < 60; y += 1) {
            world[x][y] = Tileset.NOTHING;
        }
    }
    drawHorizontalLine(world, 10, 10, 7, Tileset.MOUNTAIN);

    Hexagon h = new Hexagon(10, 10, 4, Tileset.TREE);
    h.addHexagon(world);
    ter.renderFrame(world);
}
```

Initialize graphics

Create empty world

Why did we do this again?

Review

Main method so far:

```
public static void main(String[] args) {
    TERenderer ter = new TERenderer();
    ter.initialize(60, 60);
    TETile[][] world = new TETile[60][60];
    for (int x = 0; x < 60; x += 1) {
        for (int y = 0; y < 60; y += 1) {
            world[x][y] = Tileset.NOTHING;
        }
    }
    drawHorizontalLine(world, 10, 10, 7, Tileset.MOUNTAIN);

    Hexagon h = new Hexagon(10, 10, 4, Tileset.TREE);
    h.addHexagon(world);
    ter.renderFrame(world);
}
```

Initialize graphics

Create empty world

Testing the method!

Review

Main method so far:

```java
public static void main(String[] args) {
    TERenderer ter = new TERenderer();
    ter.initialize(60, 60);
    TETile[][] world = new TETile[60][60];
    for (int x = 0; x < 60; x += 1) {
        for (int y = 0; y < 60; y += 1) {
            world[x][y] = Tileset.NOTHING;
        }
    }
    drawHorizontalLine(world, 10, 10, 7, Tileset.MOUNTAIN);

    Hexagon h = new Hexagon(10, 10, 4, Tileset.TREE);
    h.addHexagon(world);
    ter.renderFrame(world);
}
```

Initialize graphics

Create empty world

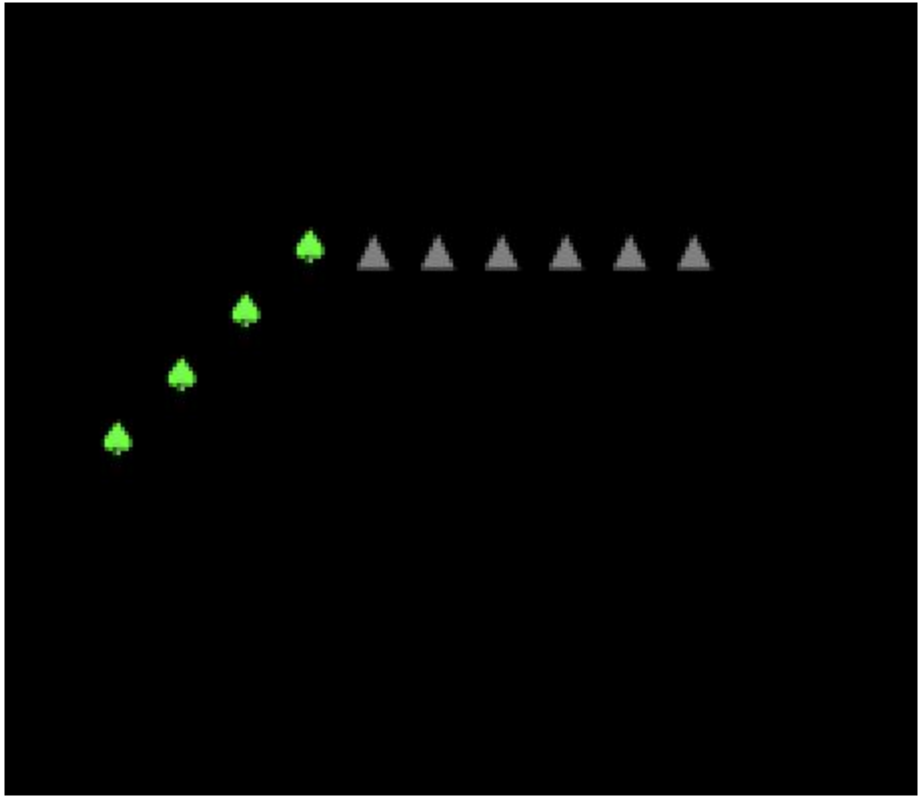
Add hexagon

Render world

Review

Output after running our code:

- Not exactly a hexagon.



Review: addHexagon

Below is our addHexagon code.

- What did we mean to do?
- What did we actually do?

```
// draws this hexagon to the world
public void addHexagon(TETile[][] world) {
   // top half
   // how many horizontal lines? s
   for (int lineNum = 0; lineNum < s; lineNum += 1) {
       world[x - lineNum][y - lineNum] = t;
   }

   // bottom half
}
```

Review: addHexagon

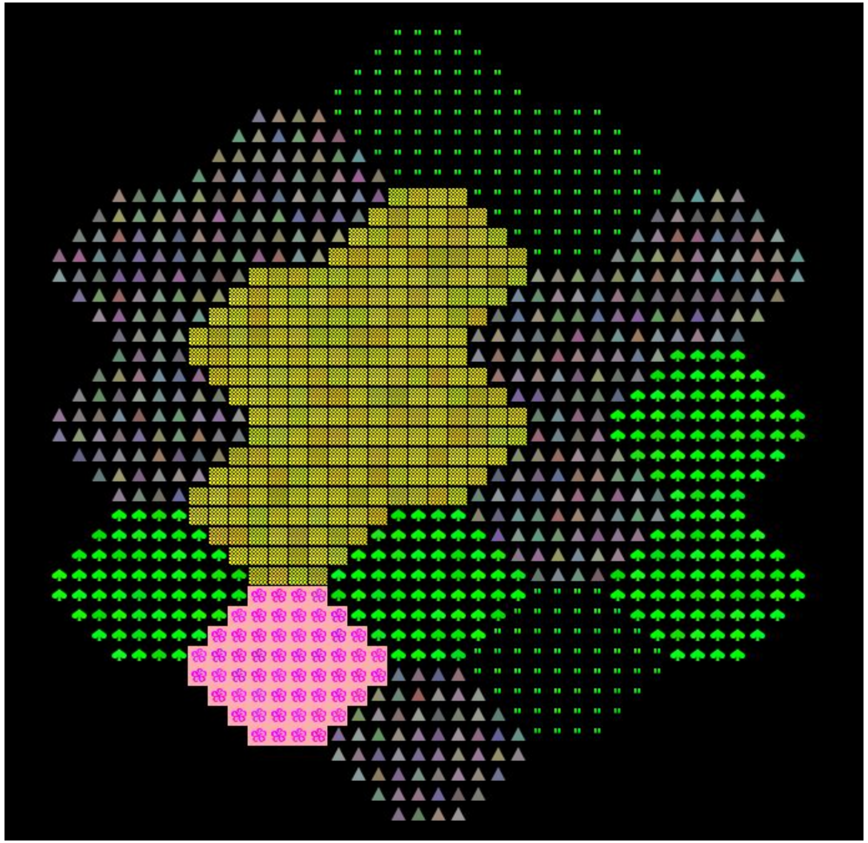
Let’s keep working.

```
// draws this hexagon to the world
public void addHexagon(TETile[][] world) {
   // top half
   // how many horizontal lines? s
   for (int lineNum = 0; lineNum < s; lineNum += 1) {
       world[x - lineNum][y - lineNum] = t;
   }

   // bottom half
}
```

Next Up

We’ve gotten single hexagons working. Now what else do we need?



Refactoring

Lecture 31, CS61B, Spring 2025

Tactical vs. Strategic Programming

Let’s try and identify places where we were “tactical”.

- Recall, in tactical programming: “Your main focus is to get something working, such as a new feature or bug fix.”

We’ll then refactor our code to be more strategic.

Potential Issues with our Code

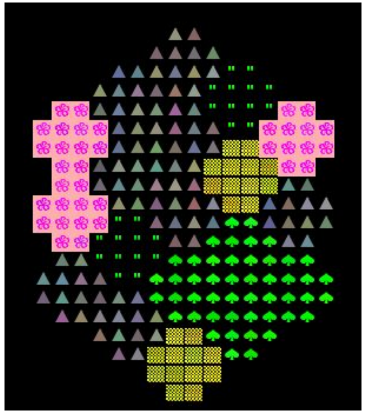
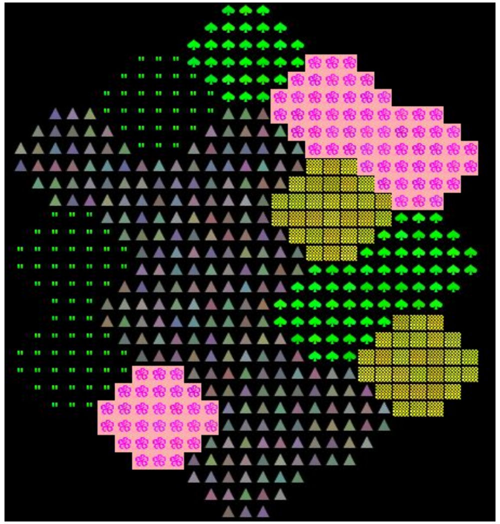
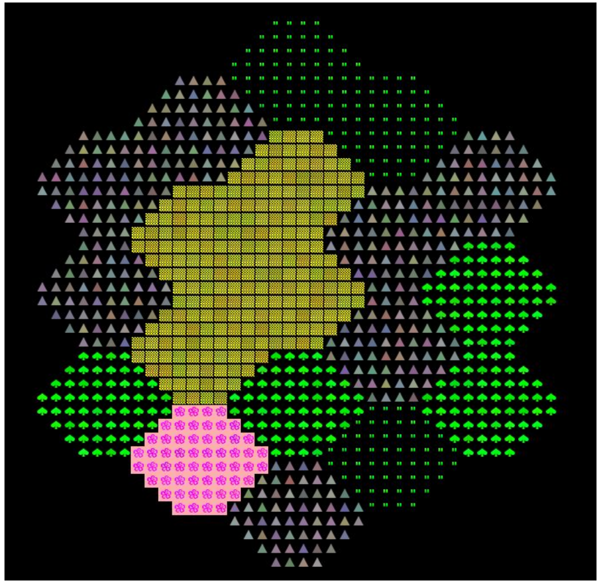
Adding a New Feature

Lecture 31, CS61B, Spring 2025

Goal: Generate Worlds

Now suppose we want the user to be able to press “d” which will delete a random hexagon.

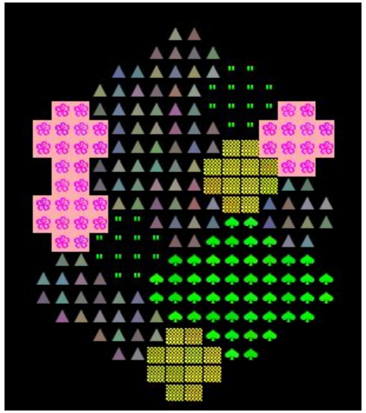
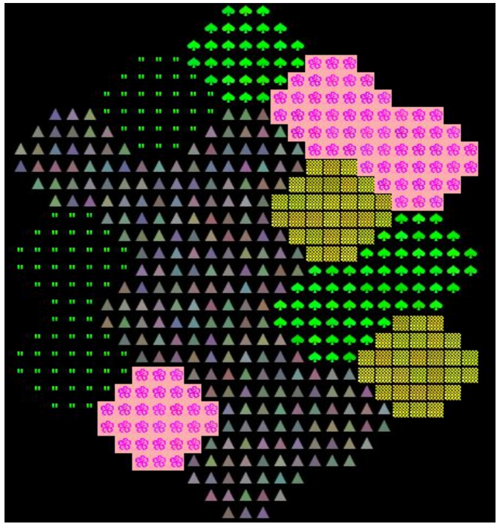
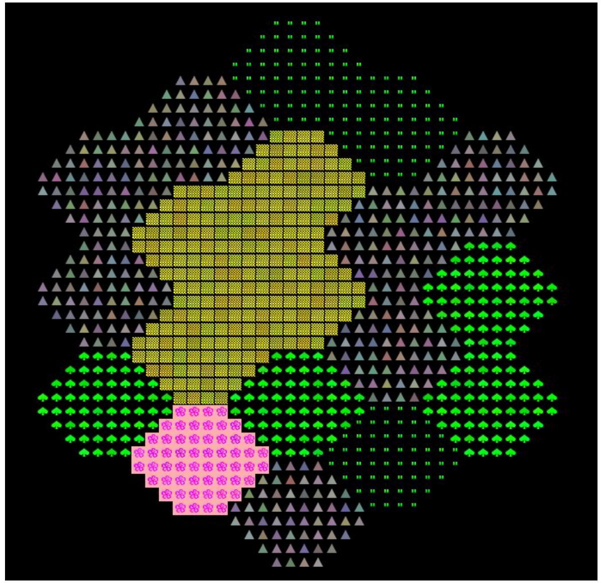
- What methods might we need now?

Implementation

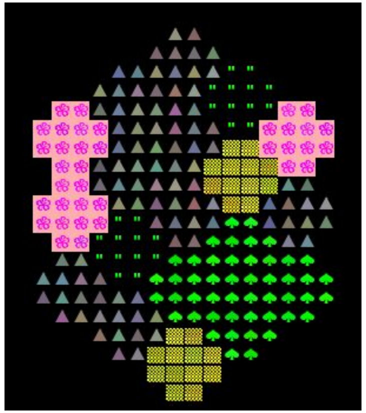
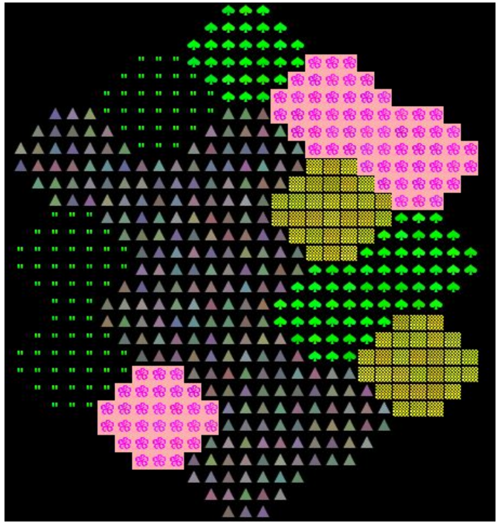
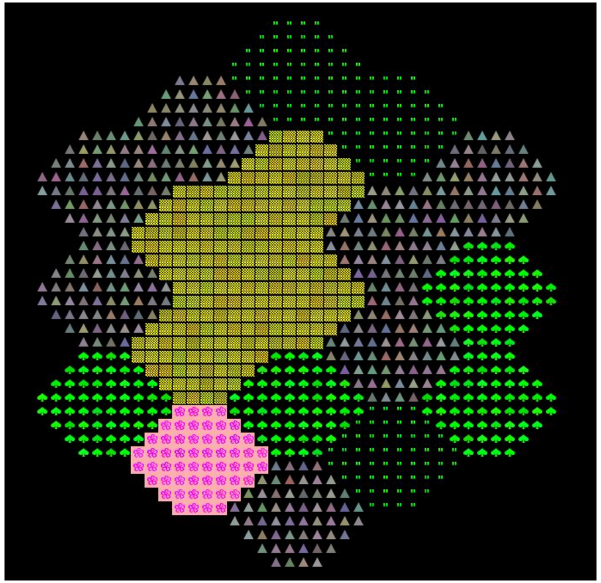
Now let’s go implement.

- I’ll use the LLM-powered text editor Cursor to help speed up the process.
  - On project 3, task 8, you can use LLMs too if you opt-in to the LLM pilot project. More shortly.

Other Features

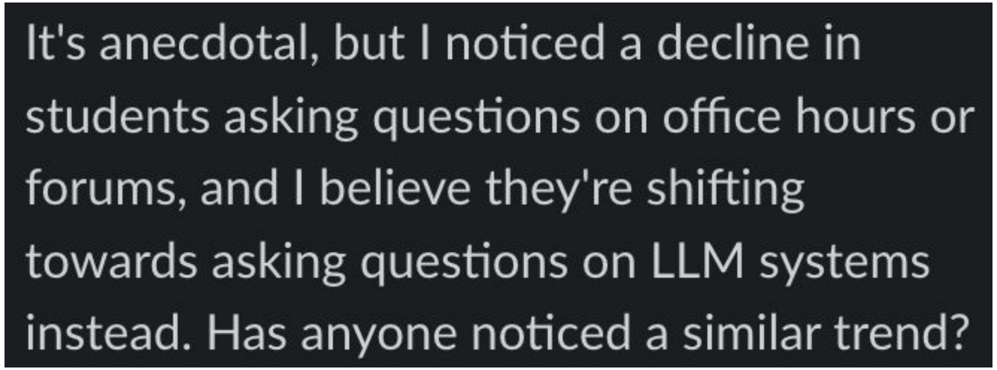
Time permitting, let's maybe add another feature.

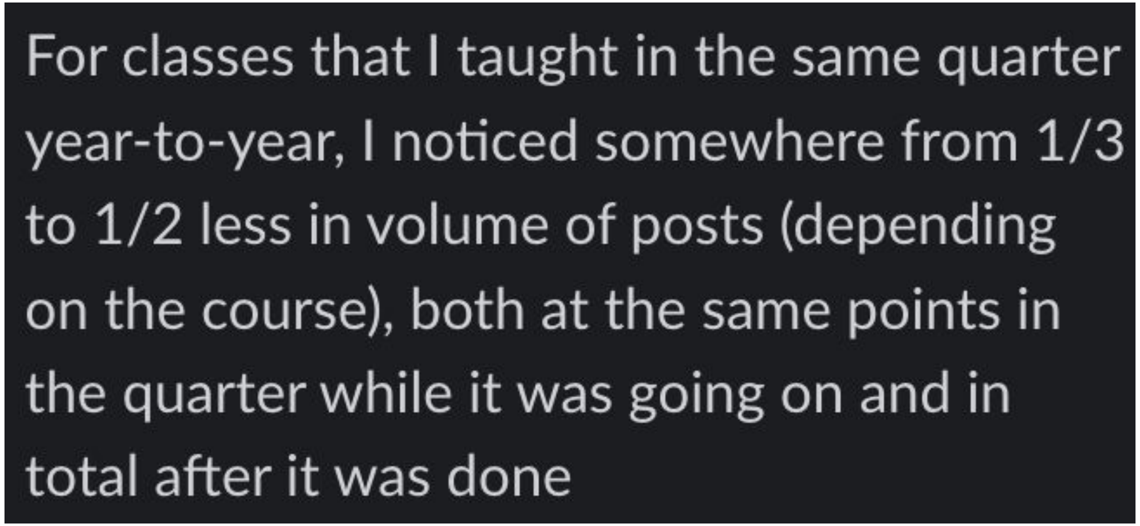




Notes on LLM Pilot, Plagiarism

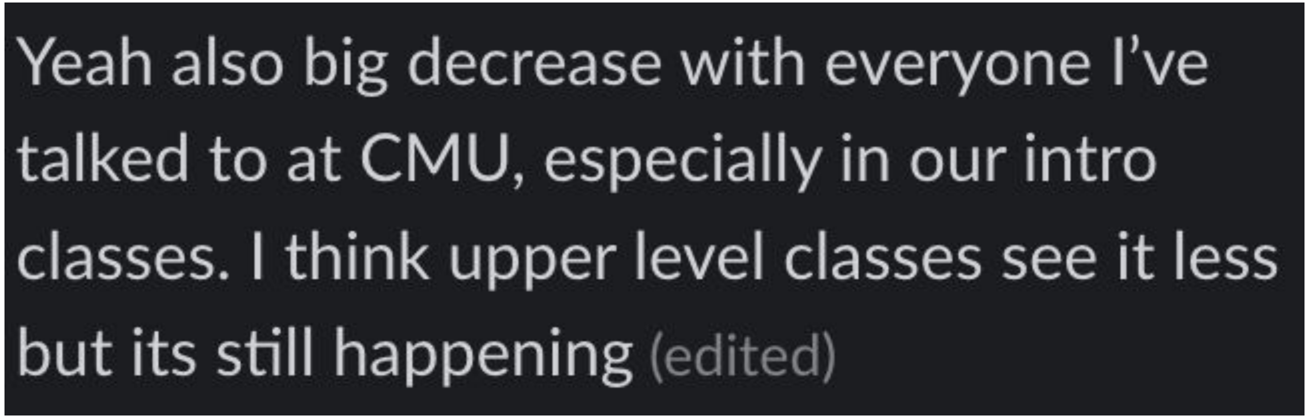
Lecture 31, CS61B, Spring 2025

LLMs and Education


It's anecdotal, but I noticed a decline in students asking questions on office hours or forums, and I believe they're shifting towards asking questions on LLM systems instead. Has anyone noticed a similar trend?


For classes that I taught in the same quarter year-to-year, I noticed somewhere from 1/3 to 1/2 less in volume of posts (depending on the course), both at the same points in the quarter while it was going on and in total after it was done

LLMs and Education



Yeah also big decrease with everyone I've talked to at CMU, especially in our intro classes. I think upper level classes see it less but its still happening (edited)

At Berkeley

Lots of folks at Berkeley have noticed office hours being much quieter than in past semesters across multiple courses.

- One exception in 61B: Project 2B

I'm currently mulling over how 61B should change. Should balance:

- Need to be able to write and understand code.
- Need to be able to use modern tools.

Project 3 LLMs

For the final task (task 8) of project 3, you can opt-in to using LLMs.

- https://sp25.datastructur.es/projects/proj3/ambition/

I'm also thinking about how to include LLMs in 61B next Fall.

- Fun fact: UCSD has an intro CS class (CSE 8A) that focuses on the use of LLMs.

Tactical vs. Strategic Programming Examples

Lecture 31, CS61B, Spring 2025

Build Your Own World

In the previous software engineering lecture, we talked about complexity.

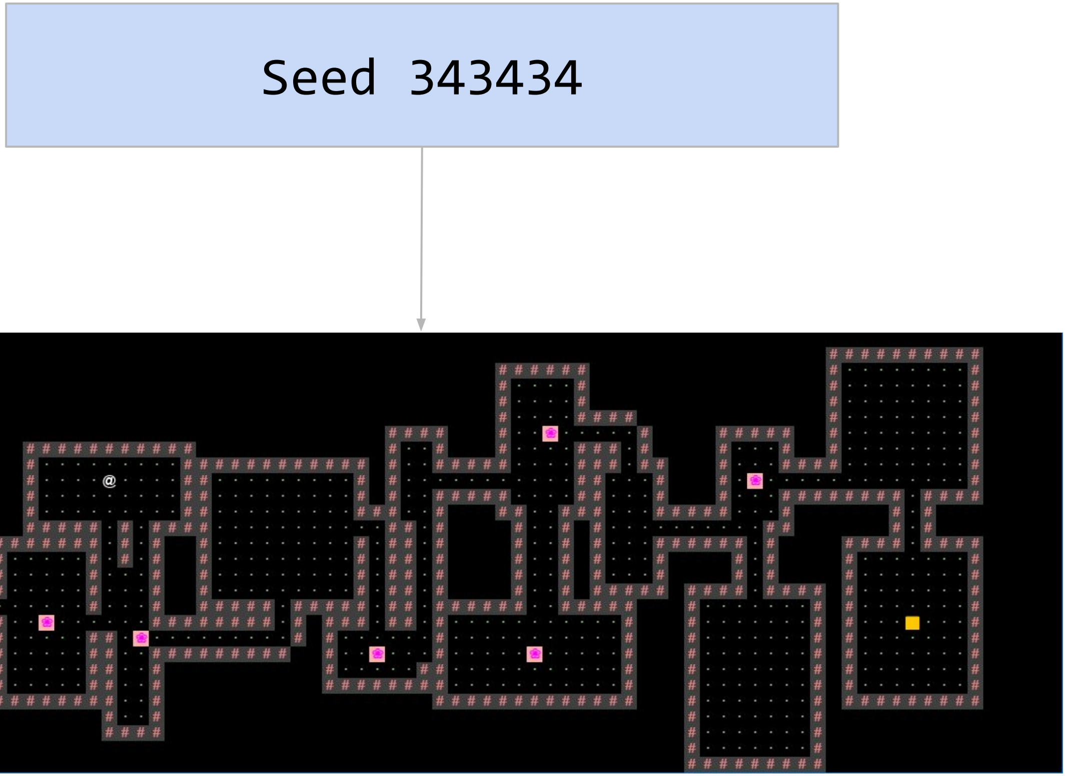
- "Complexity is anything related to the structure of a software system that makes it hard to understand and modify the system."
- None of the assignments in 61B have really given you enough room to create truly complex code.
  - Project 3 will give you a chance.

In Project 3, which you've hopefully started, you'll be building a system in two phases:

- World Generation
- Interactivity

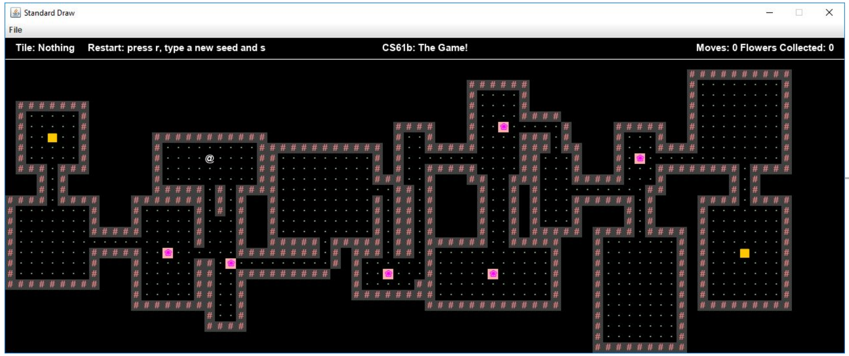
Part 1: World Generation

Given a random seed (long), generate a 2D world (TetTile[][]) with rooms and hallways.



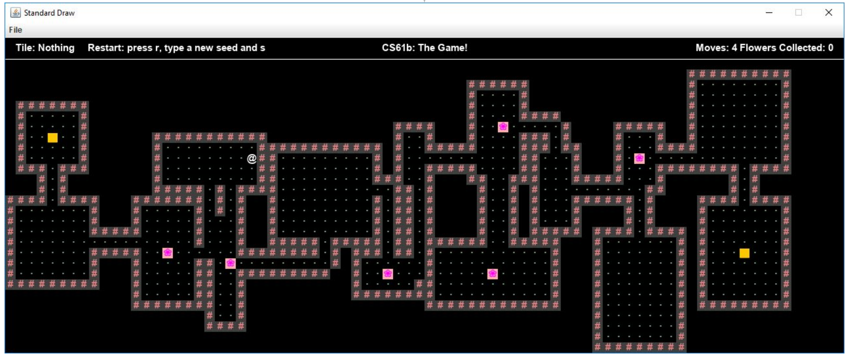
Part 1: World Generation

In part 2, you'll add the ability to walk around the world and interact with it.



Tile: Nothing Restart: press r, type a new seed and s CS61b: The Game! Moves: 0 Flowers Collected: 0

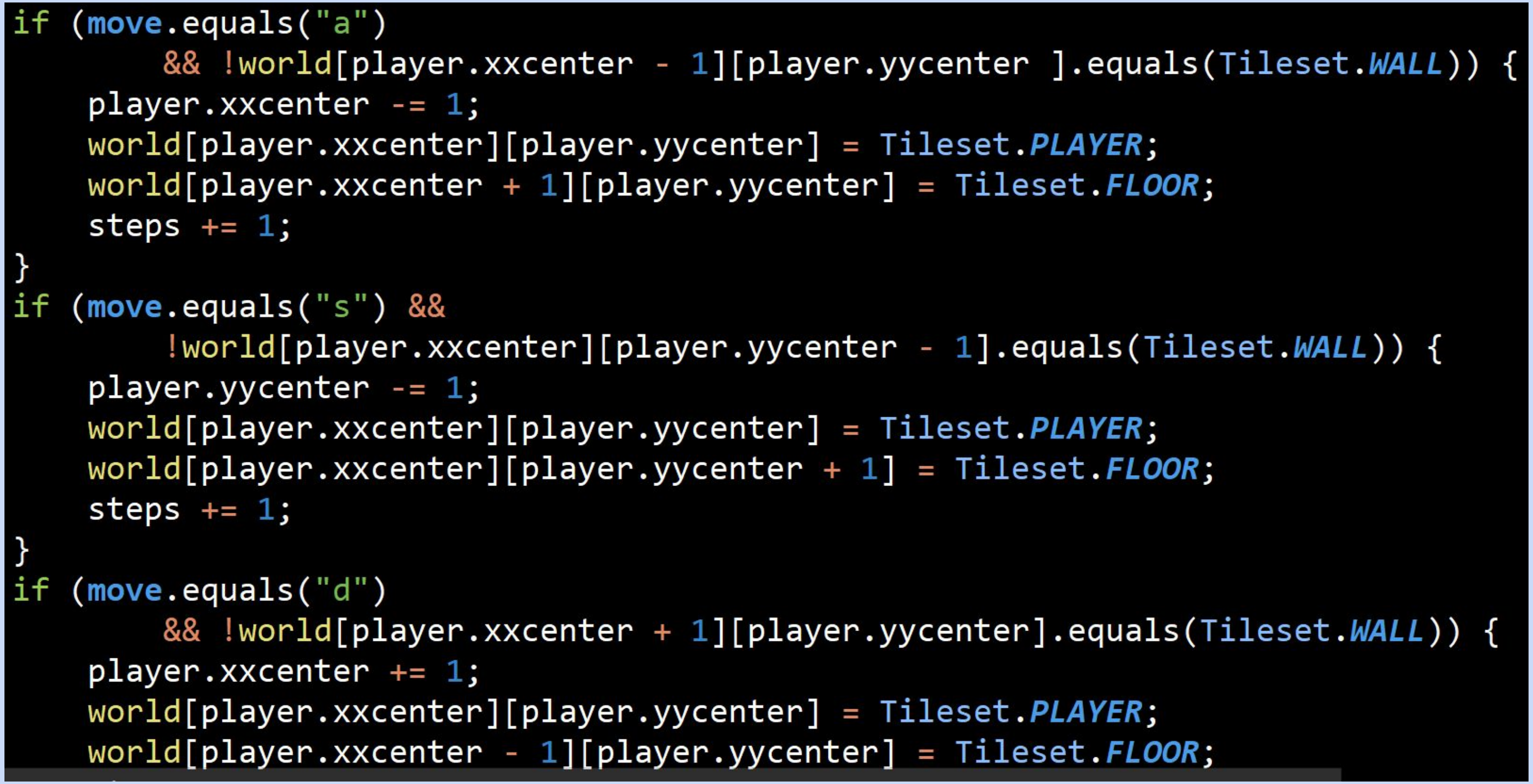
User types "dddd". Avatar moves four spaces east.



Tile: Nothing Restart: press r, type a new seed and s CS61b: The Game! Moves: 4 Flowers Collected: 0

Tactical vs. Strategic Programming

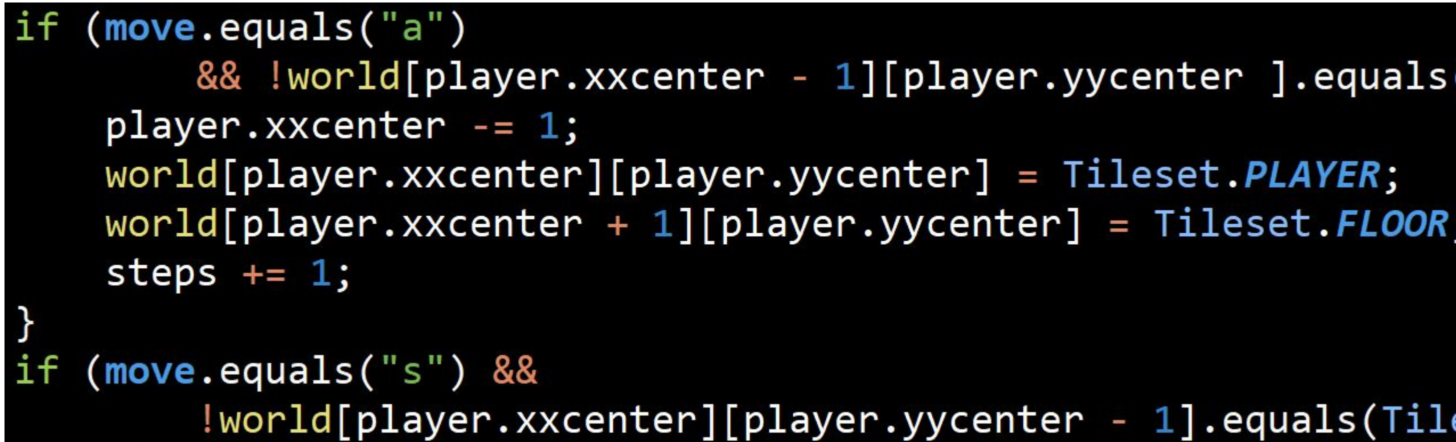
Let’s see an actual example where complexity got out of hand due to tactical programming. What is “complex” about this code?



```
if (move.equals("a")
        && !world[player.xxcenter - 1][player.yycenter ].equals(Tileset.WALL)) {
    player.xxcenter -= 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter + 1][player.yycenter] = Tileset.FLOOR;
    steps += 1;
}
if (move.equals("s") &&
        !world[player.xxcenter][player.yycenter - 1].equals(Tileset.WALL)) {
    player.yycenter -= 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter][player.yycenter + 1] = Tileset.FLOOR;
    steps += 1;
}
if (move.equals("d")
        && !world[player.xxcenter + 1][player.yycenter].equals(Tileset.WALL)) {
    player.xxcenter += 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter - 1][player.yycenter] = Tileset.FLOOR;
```

Tactical vs. Strategic Programming

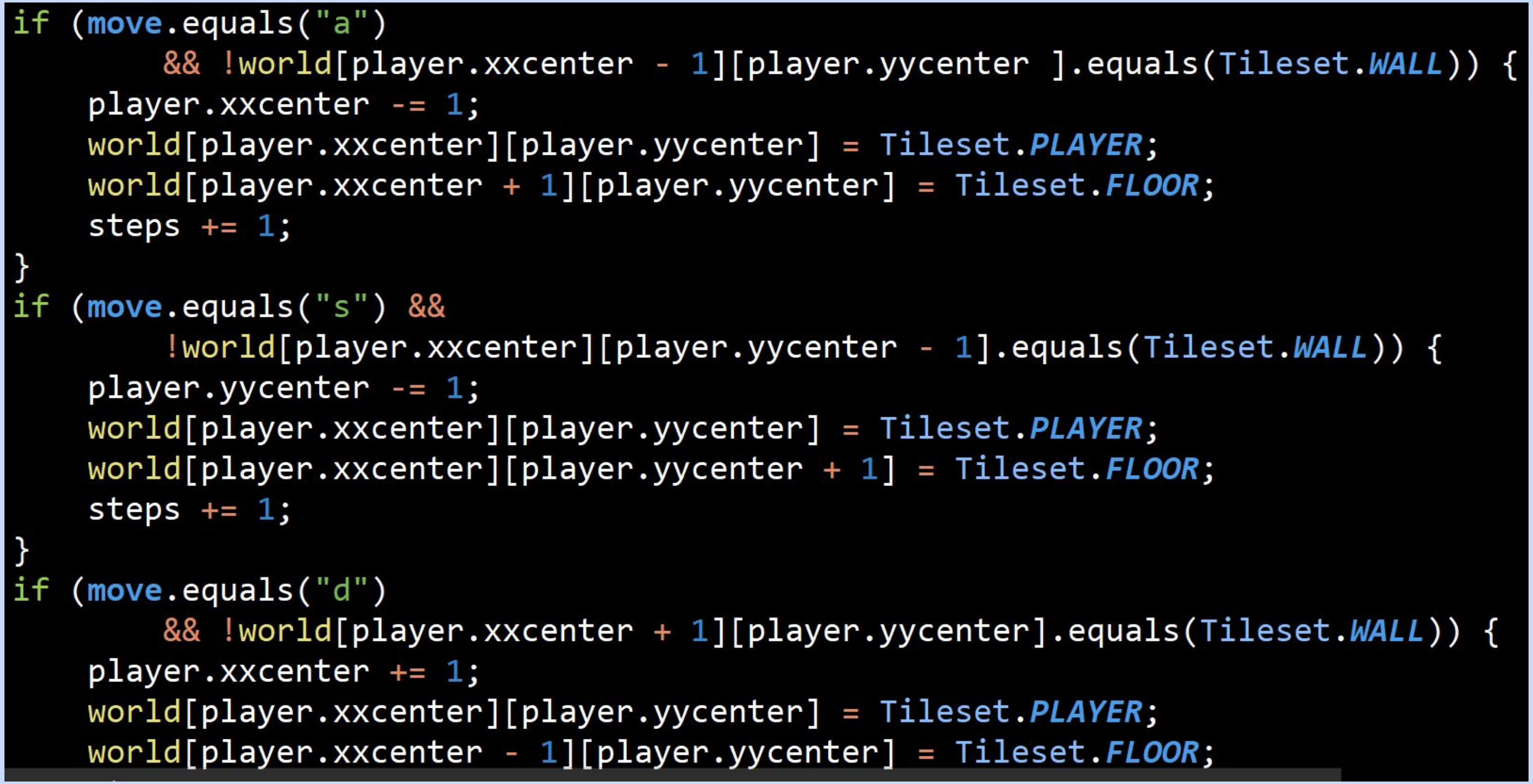
What is “complex” about this code?



```
if (move.equals("a")
        && !world[player.xxcenter - 1][player.yycenter ].equals
    player.xxcenter -= 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter + 1][player.yycenter] = Tileset.FLOOR
    steps += 1;
}
if (move.equals("s") &&
        !world[player.xxcenter][player.yycenter - 1].equals(Tile
```

Strategic Programming

Give some examples of changes you’d make to simplify this code.



```
if (move.equals("a")
        && !world[player.xxcenter - 1][player.yycenter ].equals(Tileset.WALL)) {
    player.xxcenter -= 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter + 1][player.yycenter] = Tileset.FLOOR;
    steps += 1;
}
if (move.equals("s") &&
        !world[player.xxcenter][player.yycenter - 1].equals(Tileset.WALL)) {
    player.yycenter -= 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter][player.yycenter + 1] = Tileset.FLOOR;
    steps += 1;
}
if (move.equals("d")
        && !world[player.xxcenter + 1][player.yycenter].equals(Tileset.WALL)) {
    player.xxcenter += 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter - 1][player.yycenter] = Tileset.FLOOR;
```

Strategic Programming

Give some examples of changes you’d make to simplify this code.

-

Ousterhout’s Take on Complexity

There are two primary sources of complexity:

- **Dependencies**: When a piece of code cannot be read, understood, and modified independently.
- **Obscurity**: When important information is not obvious.

Both of these happen in the code we just saw.

- You must remain vigilant to remain letting dependencies and obscurities from infesting your code.

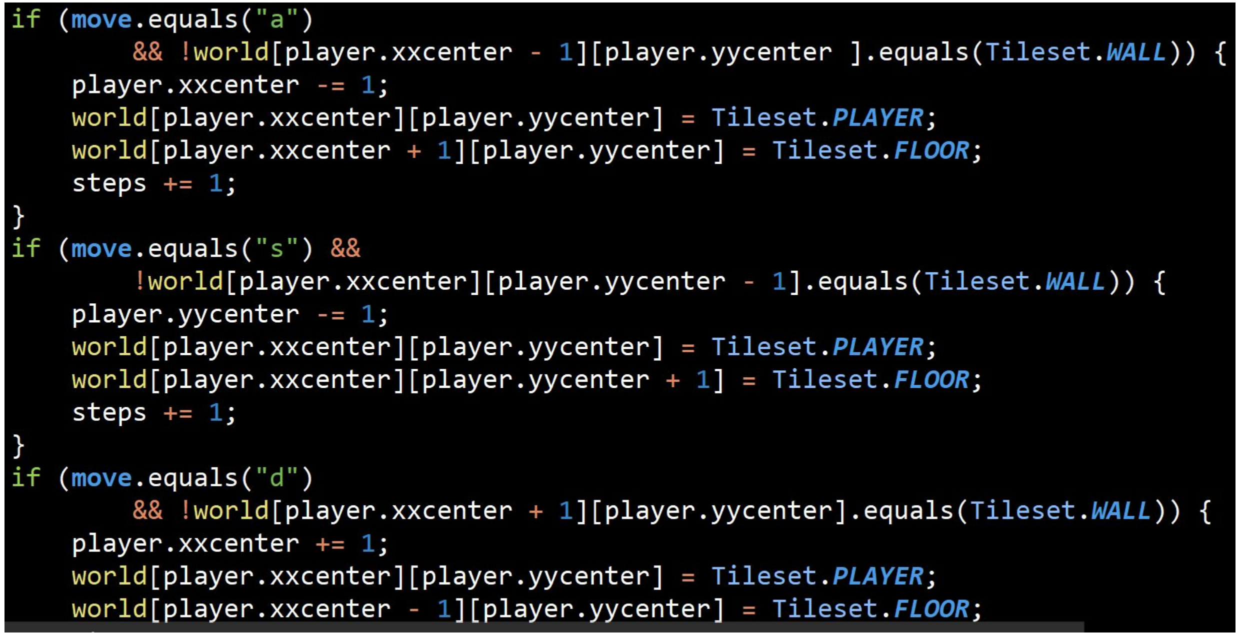
The End Result of Tactical Programming

What is “complex” about this code?

- Complex manual computation of west, east, north, and south.
- Lots of variables that need to be manipulated exactly so.
- Repetitive code (steps += 1, similar logic in many places).

obscurity!

dependencies!



```
if (move.equals("a")
        && !world[player.xxcenter - 1][player.yycenter ].equals(Tileset.WALL)) {
    player.xxcenter -= 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter + 1][player.yycenter] = Tileset.FLOOR;
    steps += 1;
}
if (move.equals("s") &&
        !world[player.xxcenter][player.yycenter - 1].equals(Tileset.WALL)) {
    player.yycenter -= 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter][player.yycenter + 1] = Tileset.FLOOR;
    steps += 1;
}
if (move.equals("d")
        && !world[player.xxcenter + 1][player.yycenter].equals(Tileset.WALL)) {
    player.xxcenter += 1;
    world[player.xxcenter][player.yycenter] = Tileset.PLAYER;
    world[player.xxcenter - 1][player.yycenter] = Tileset.FLOOR;
```

Don’t Be This Dog

Other Tips

Do not create mutable static variables.

- This is a very bad idea. It makes it very difficult to reason about your code.
- Example:
    - public static Position playerLocation : MASSIVELY BAD IDEA

Why is this such a bad idea?

Other Tips

Take some time to refactor occasionally.

- If you mess up and break your program while cleaning up your code, use git to go back to a known working state.
- Unit tests can help prevent these issues.
  - Though many things in this project are challenging to unit test. No (easy) way to test graphics.

Deep Modules

Lecture 31, CS61B, Spring 2025

Hiding Complexity

One powerful tool for managing complexity is to design your system so that programmer is only thinking about some of the complexity at once.

- By using helper methods, e.g `getNeighbor(WEST)` and helper classes, e.g. `DrawUtils`, you can hide complexity.

Modular Design

In an ideal world, system would be broken down into modules, where every module would be totally independent.

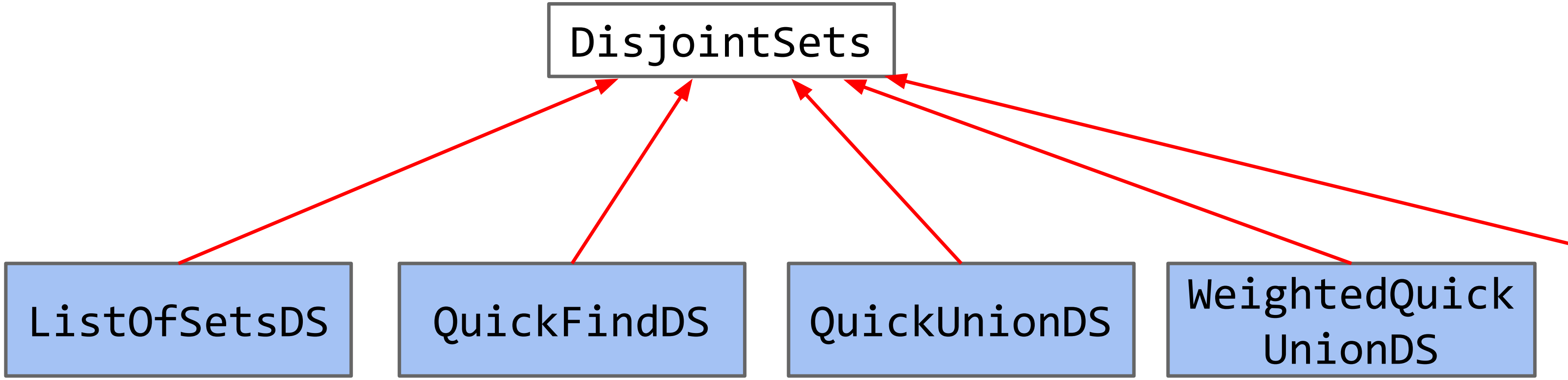
- Here, “module” is an informal term referring to a class, a package, or other unit of code.
- Not possible for modules to be entirely independent, because code from each module has to call other modules.
  - e.g. need to know signature of methods to call them.

In modular design, our goal is to minimize dependencies between modules.

Interface vs. Implementation

As we've seen, there is an important distinction between Interface and Implementation.

- Map is an interface.
- HashMap, TreeMap, etc. are implementations.



Interface vs. Implementation

Ousterhout: “The best modules are those whose interfaces are much simpler than their implementation.” Why?

- A simple interface minimizes the complexity the module can cause elsewhere. If you only have a `getNext()` method, that’s all someone can do.
- If a module’s interface is simple, we can change an implementation of that module without affecting the interface.
    - Silly example: If `List` had an `arraySize` method, this would mean you’d be stuck only being able to build array based lists.

Will pick up at the next software engineering lecture

Interface

A Java interface has both a formal and an informal part:

- Formal: The list of method signatures.
- Informal: Rules for using the interface that are not enforced by the compiler.
    - Example: If your iterator requires hasNext to be called before next in order to work properly, that is an informal part of the interface.
    - Example: If your add method throws an exception on null inputs, that is an informal part of the interface.
    - Example: Runtime for a specific method, e.g. `add` in `ArrayList`.
    - Can only be specified in comments.

Be wary of the informal rules of your modules as you build project 3.

- Static mutable variables result in **horrifically complex informal rules**.

Modules Should Be Deep

Ousterhout: “The best modules are those that provide powerful functionality yet have simple interfaces. I use the term *deep* to describe such modules.”

For example, a RedBlackBSTSet is a deep module.

- Simple interface:
  - Add, contains, delete methods.
  - Nothing informal that user needs to know (e.g. user doesn’t have to specify or know which nodes are red or black).
- Powerful functionality:
  - Operations are efficient.
  - Tree balance is maintained using sophisticated, subtle rules.

Information Hiding, Temporal Decomposition

Lecture 31, CS61B, Spring 2025

Information Hiding

The most important way to make your modules deep is to practice “information hiding”.

- Embed knowledge and design decision in the module itself, without exposing them to the outside world.

Reduces complexity in two ways:

- Simplifies interface.
- Makes it easier to modify the system.

Information Leakage

The opposite of **information hiding** is **information leakage**.

- Occurs when design decision is reflected in multiple modules.
  - Any change to one requires a change to all.
- Example:
  - Information is embodied in two places, i.e. it has “leaked”.

Information Leakage Example: An Avatar Class from Fall 2024

```java
public class Avatar extends GenericBuilder {
    public Coordinate pos;
    private TETile[][] visibleWorld;
    ...
    public void moveLeft() {
        Coordinate west = pos.westNeighbor();
        if (validCoordinate(west)) {
           if (tileMatch(west, Tileset.FLOOR)) {
               this.pos = west;
           } else if (tileMatch(west, Tileset.COIN)) {
               this.pos = west;
               foundCoinLastTurn = true;
               setTile(this.world, this.pos, Tileset.FLOOR);
          }
    }
}
```

What is “leaky” about this?

Information Leakage

Ousterhout:

- “Information leakage is one of the most important red flags in software design.”
- “One of the best skills you can learn as a software designer is a high level of sensitivity to information leakage.”

Temporal Decomposition

One of the biggest causes of information leakage is “temporal decomposition.”

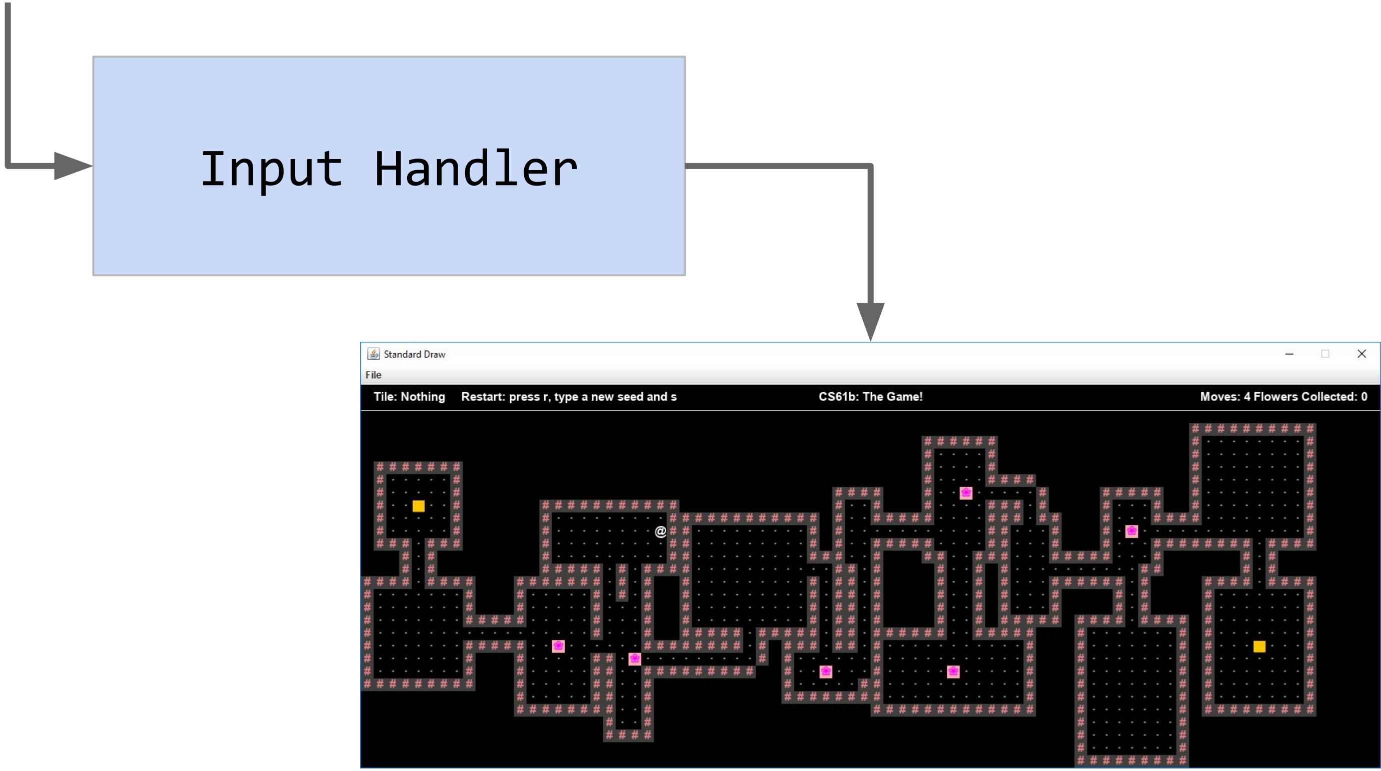
- In temporal decomposition, the structure of your system reflects the order in which events occur. Example, you will need to implement a save/load feature. In your system, the user:
  - Starts the program.
  - Enters a **random seed.**
  - **Moves around using WASD**.
  - **Saves the state** and quits.
  - Restarts the program.
  - **Loads** the state.

As suggested in lab 9, one approach to **saving and loading** is to simply record the **random seed and sequence of key presses**.

- A purely temporal decomposition will miss this opportunity to create a deep module that takes input and yields a world state.

Information Hiding

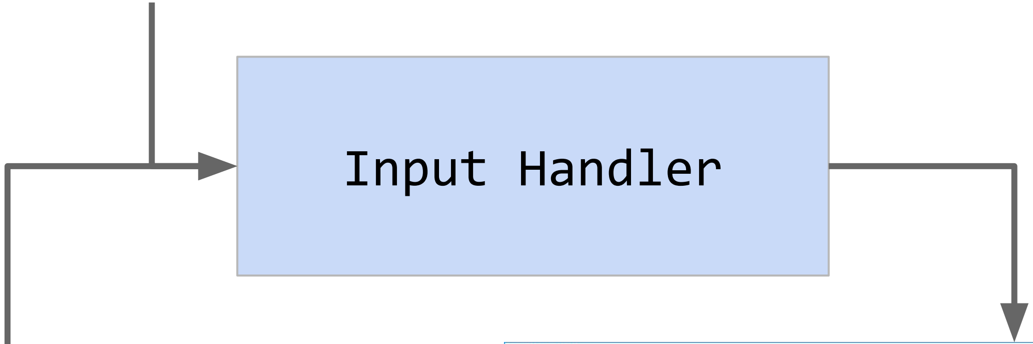
User enters random seed 239874, then pressed DDDD to move east east east east



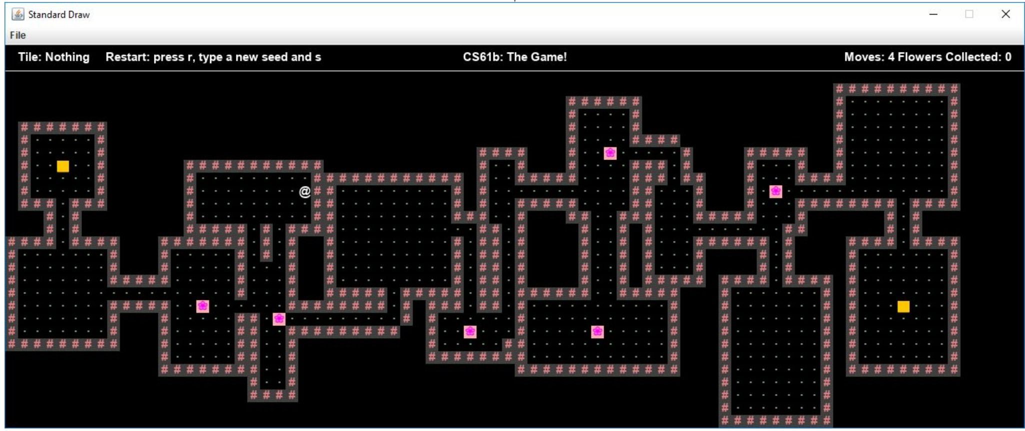
Tile: Nothing Restart: press r, type a new seed and s CS61b: The Game! Moves: 4 Flowers Collected: 0

Information Hiding

User enters random seed 239874, then pressed DDDD to move east east east east



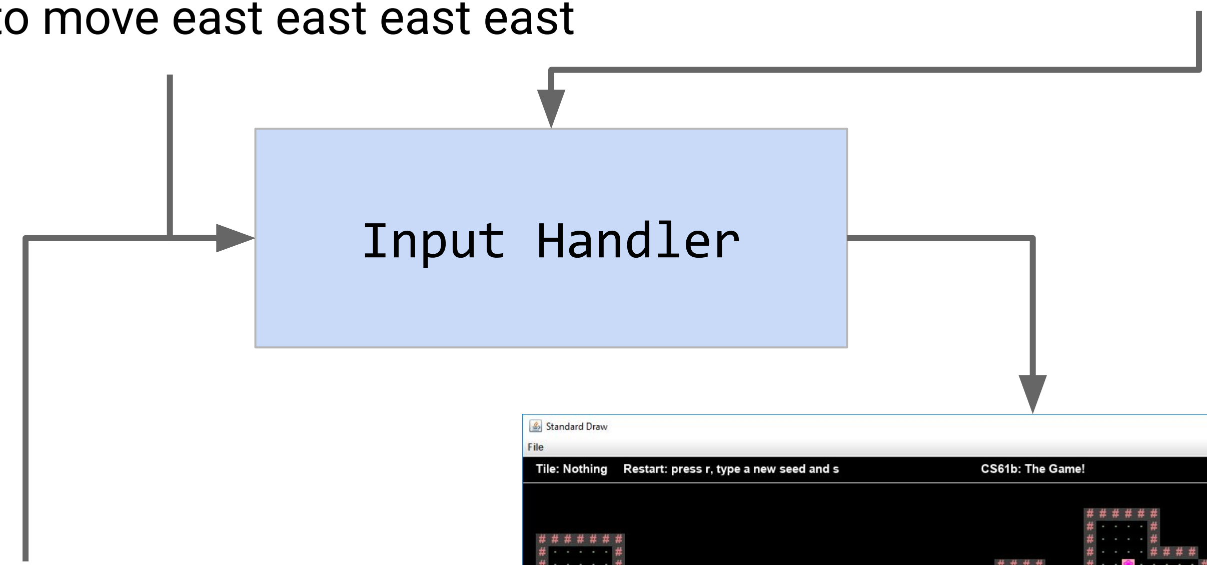
User loads file containing “234974DDDD”


Standard Draw
File
Tile: Nothing
Restart: press r, type a new seed and s
CS61b: The Game!
Moves: 4 Flowers Collected: 0
@

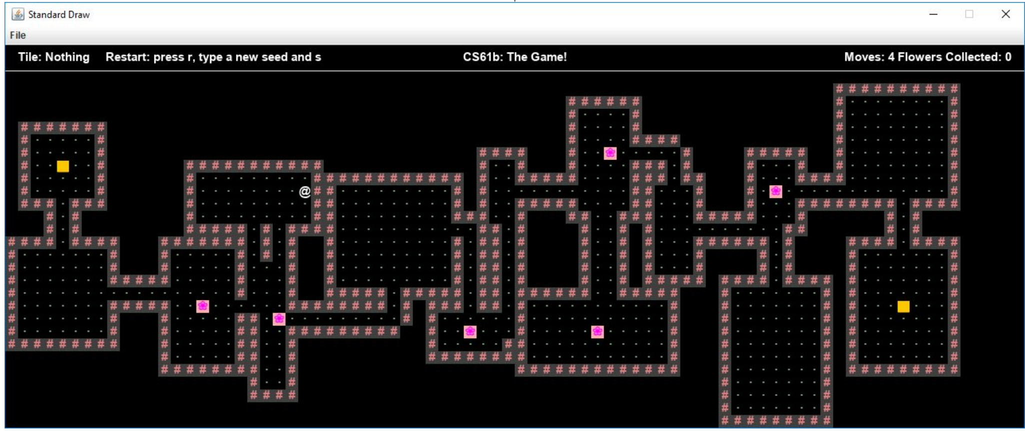
Information Hiding

User wants to replay their game from the beginning

User enters random seed 239874, then pressed DDDD to move east east east east



User loads file containing “234974DDDD”



Tile: Nothing   Restart: press r, type a new seed and s   CS61b: The Game!   Moves: 4 Flowers Collected: 0

Summary

Lecture 31, CS61B, Spring 2025

Summary and BYOW Suggestions

Some suggestions as you embark on BYOW:

- Build classes that provide functionality needed in many places in your code.
- Create “deep modules”, e.g. classes with simple interfaces that do complicated things.
- Avoid over-reliance on “temporal decomposition” where your decomposition is driven primarily by the order in which things occur.
  - It’s OK to use some temporal decomposition, but try to fix any information leakage that occurs!
- Be strategic, not tactical.
  - Refactor occasionally if your code gets too complicated.
- Most importantly: Hide information from yourself when unneeded!