

# Designing Abstractions



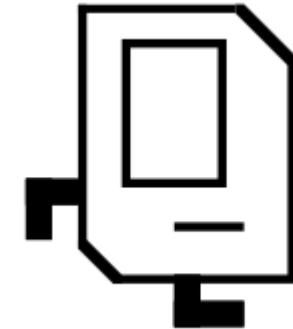
Roy Lichtenstein, *Bull Profile I - VI*

# Outline for Today

- ***Apply to Section Lead!***
  - A quick recruiting pitch.
- ***Review: What's an Abstraction?***
  - You're well acquainted with them - but let's take stock anyway.
- ***A Missing Container***
  - Some motivating problems.
- ***Creating Our Own Classes***
  - Expanding our toolkit.

---

# CS198 Section Leading



---

cs198@cs.stanford.edu

---

# Who should section lead?

---

For this round of applications, we are looking for applicants who are currently enrolled CS106B... and that's you!

We are looking for section leaders from all backgrounds who can relate to students and clearly explain concepts.

---

# What do section leaders do?

---

- Teach a weekly 1 hour section
- Help students in the LaLR
- Grade CS106 assignments
- Hold IGs with students
- Grade midterms and finals
- Get paid \$18.50/hour (more with seniority)
- Have fun!

---

# Time and requirements

---

You'll need to:

- Section lead for **two quarters!**
- Take CS198 for 3-4 units (1st quarter only)
- Attend staff meetings (Monday, 5:30-6:30PM)
- Attend workshops (Mon/Wed evenings, 1st quarter only)
- Fulfill all teaching, LaIR, and grading responsibilities

---

# Why section lead?

---

- “Learn to teach; teach to learn”
- Work directly with students
- Participate in fun events
- Join an amazing group of people
- Leave your mark on campus

---

# Participate in fun events

---



- LaIR Formal
- Special D
- Movie Nights
- BAWK
- Lecturer Hangouts
- New SL Picnic
- Swag
- And more!

# Apply Now

---

**Applications are open!**

Deadlines:

**Friday, February 17th at 11:59PM PT** for students who are currently enrolled in CS106B

Online application: [cs198.stanford.edu](http://cs198.stanford.edu)

Contact us: [cs198@cs.stanford.edu](mailto:cs198@cs.stanford.edu)

---

# Designing Abstractions

# *ab·strac·tion*

[...]

the process of considering something independently of its associations, attributes, or concrete accompaniments.

Source: Oxford Languages

Language profile

Road network

RNA translation

# Map

Fire simulation

Terrain heightmap

Human pyramid weights

# Set

# Grid

# Queue

Worker schedule

Collection of languages

Neighboring cities

Looping piece of music

Information on flood propagation

Building a rich vocabulary of abstractions makes it possible to ***model and solve*** a wider class of problems.

## ***Question One:***

How do we create new abstractions to model ideas not precisely captured by the standard container types?

## ***Question Two:***

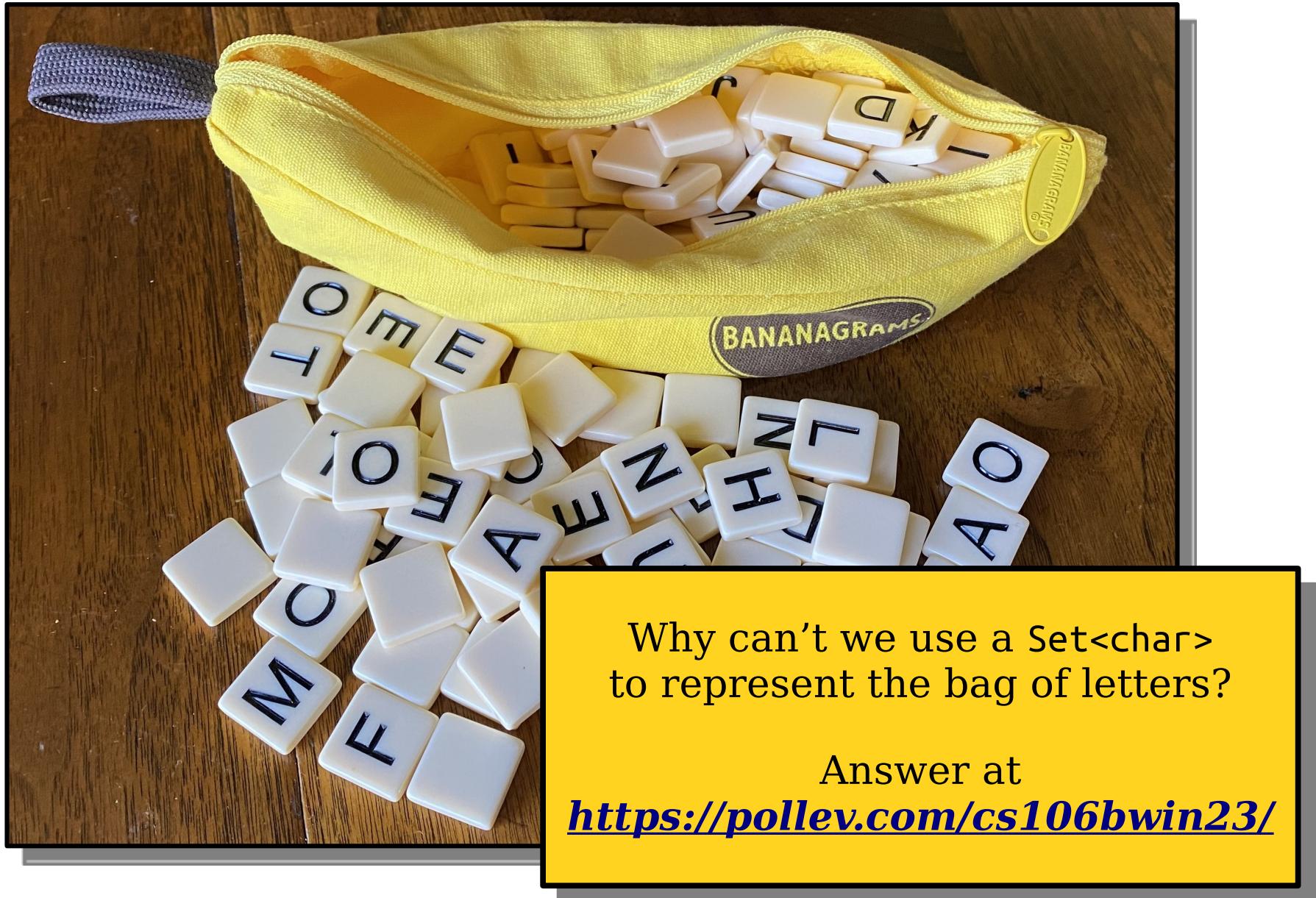
How do the abstractions we've been using so far work, and how can we use that knowledge to build richer abstractions?

# Expanding Our Abstraction Vocabulary

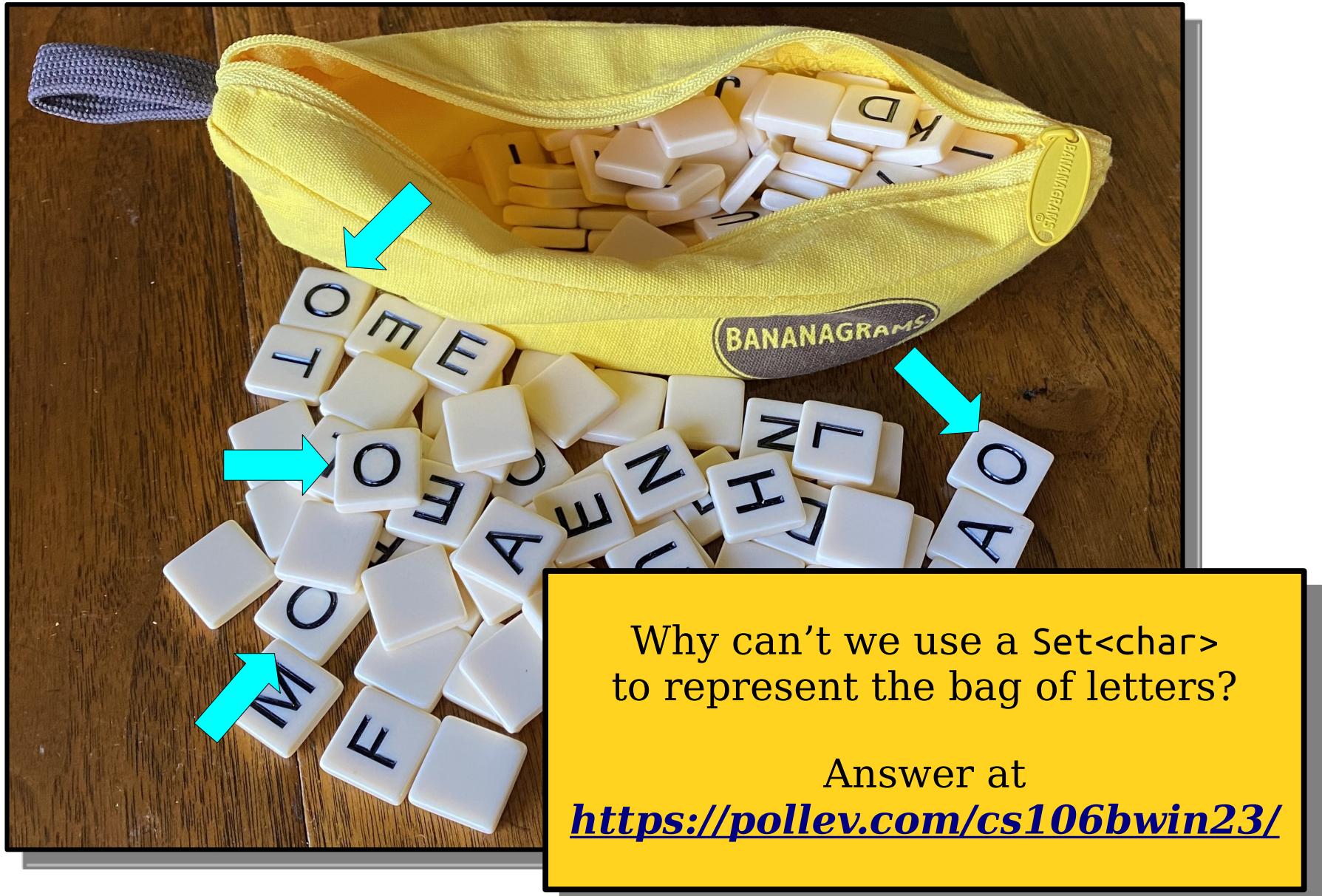
# Bananagrams Tiles



# Bananagrams Tiles



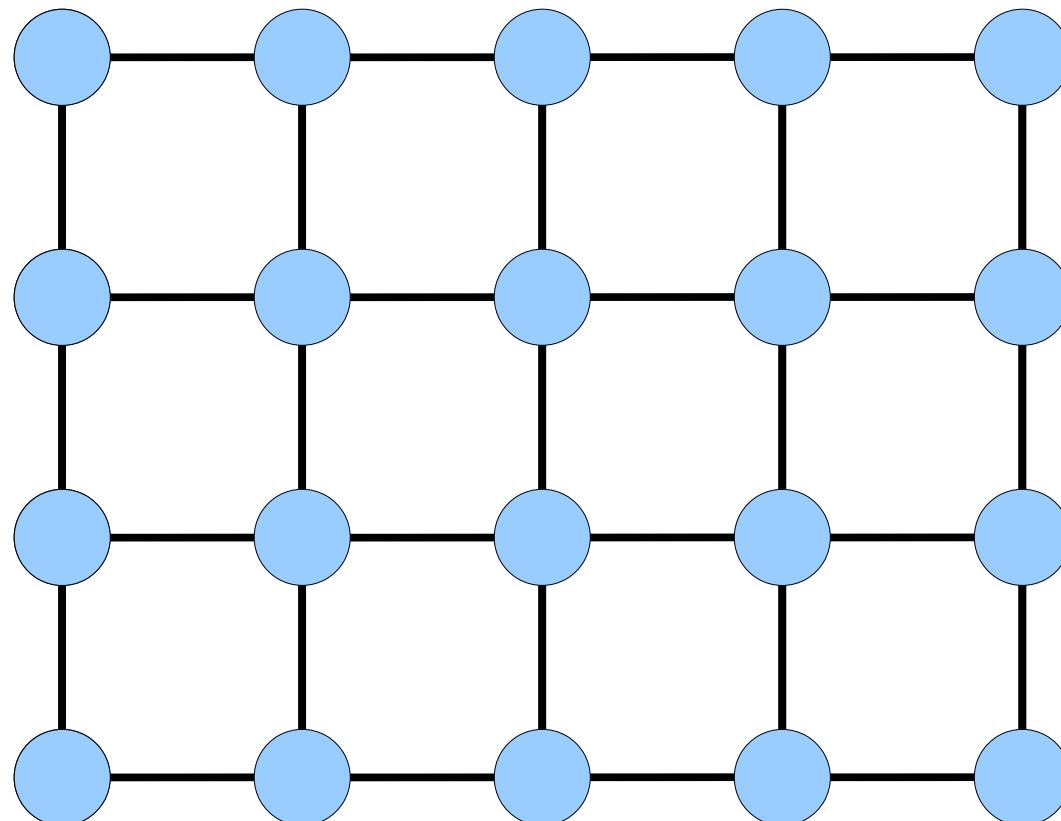
# Bananagrams Tiles



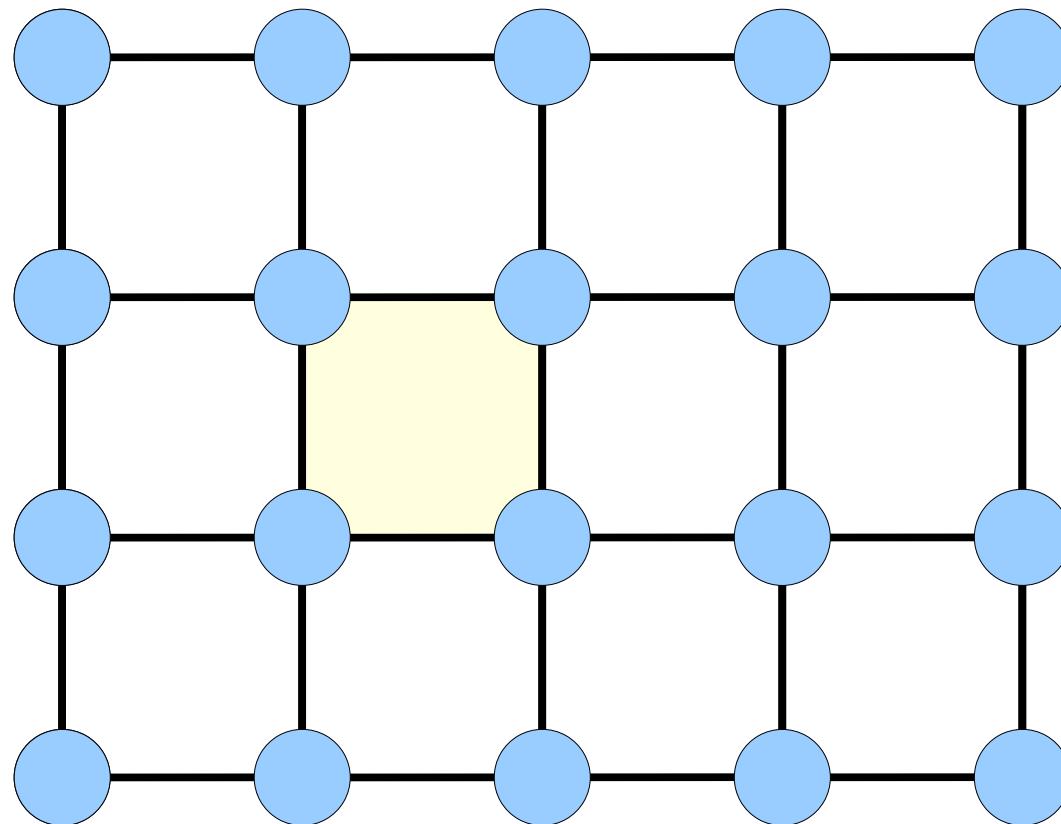
Why can't we use a `Set<char>` to represent the bag of letters?

Answer at  
<https://pollev.com/cs106bwin23/>

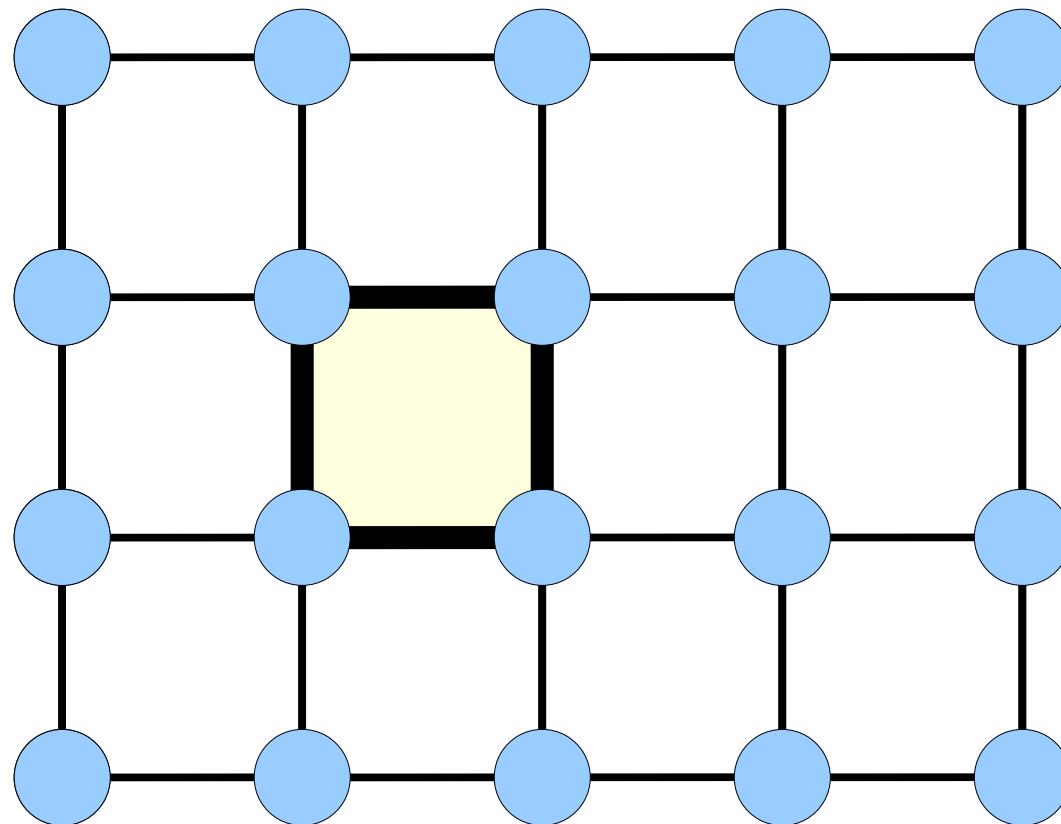
# Creating a Maze



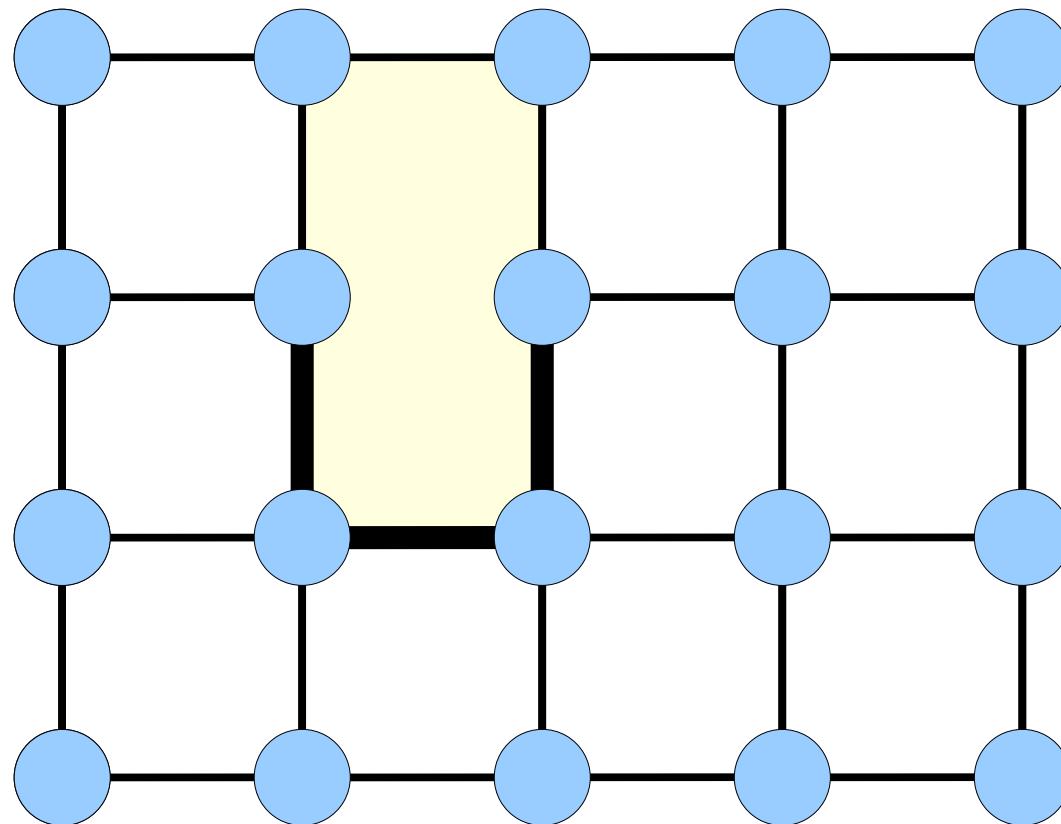
# Creating a Maze



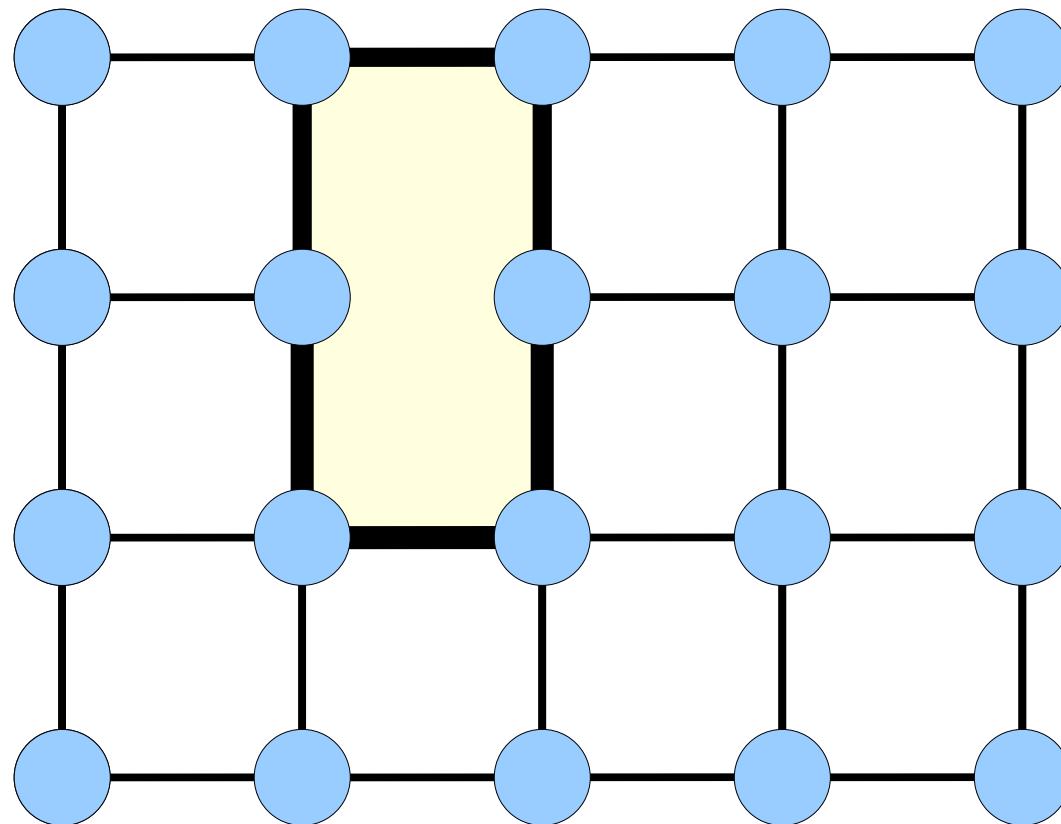
# Creating a Maze



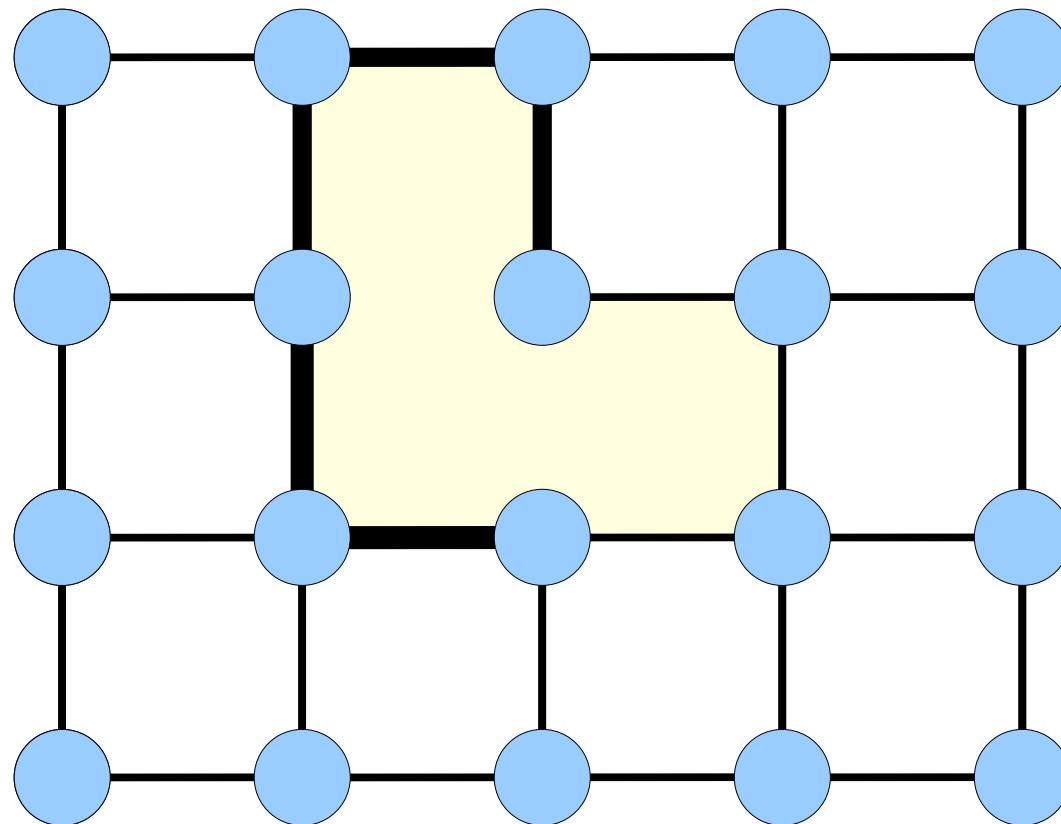
# Creating a Maze



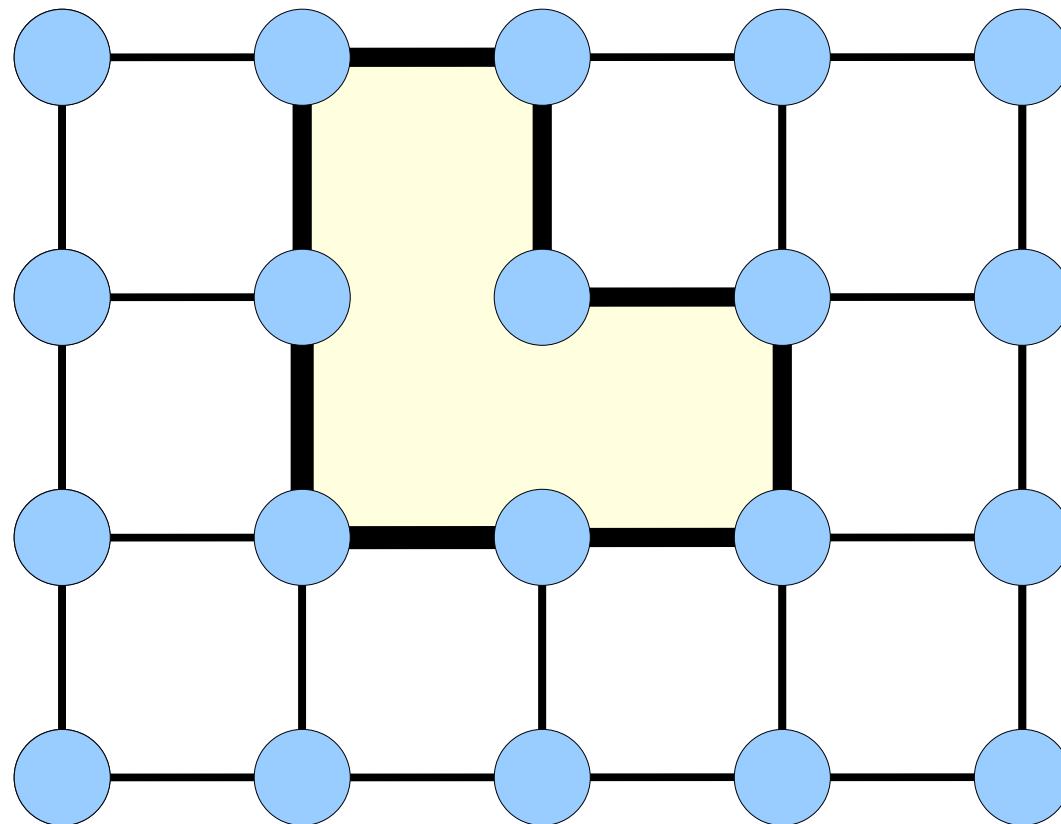
# Creating a Maze



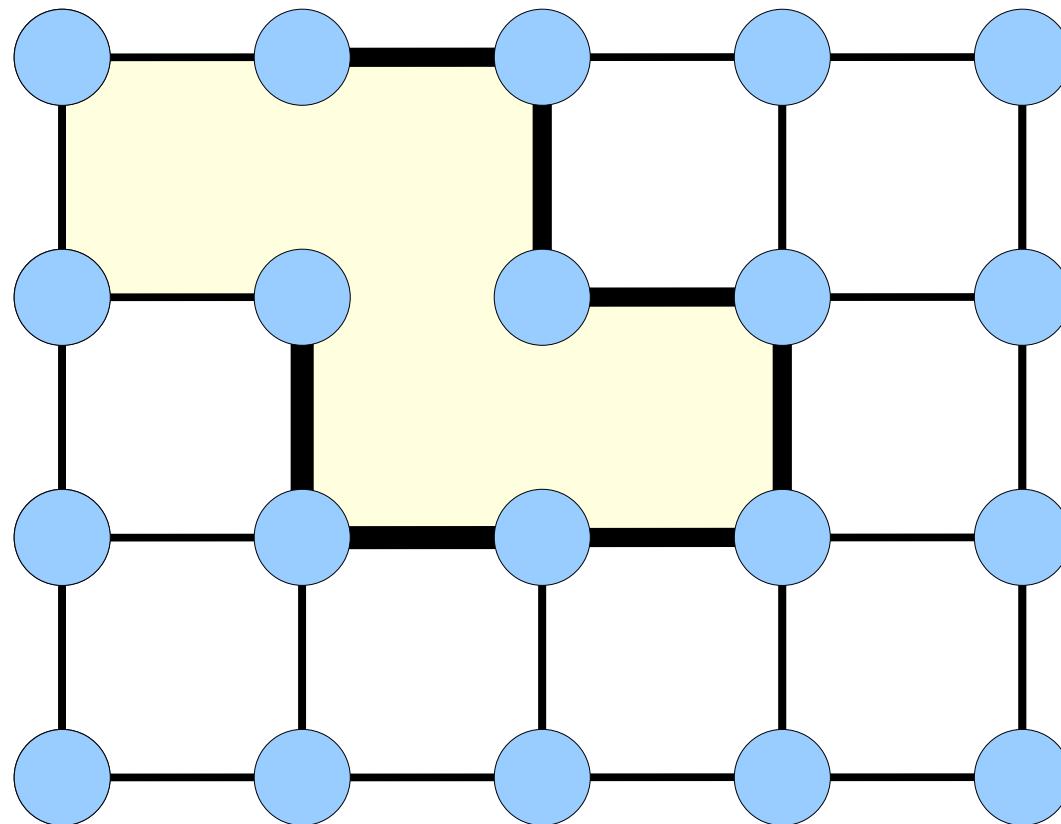
# Creating a Maze



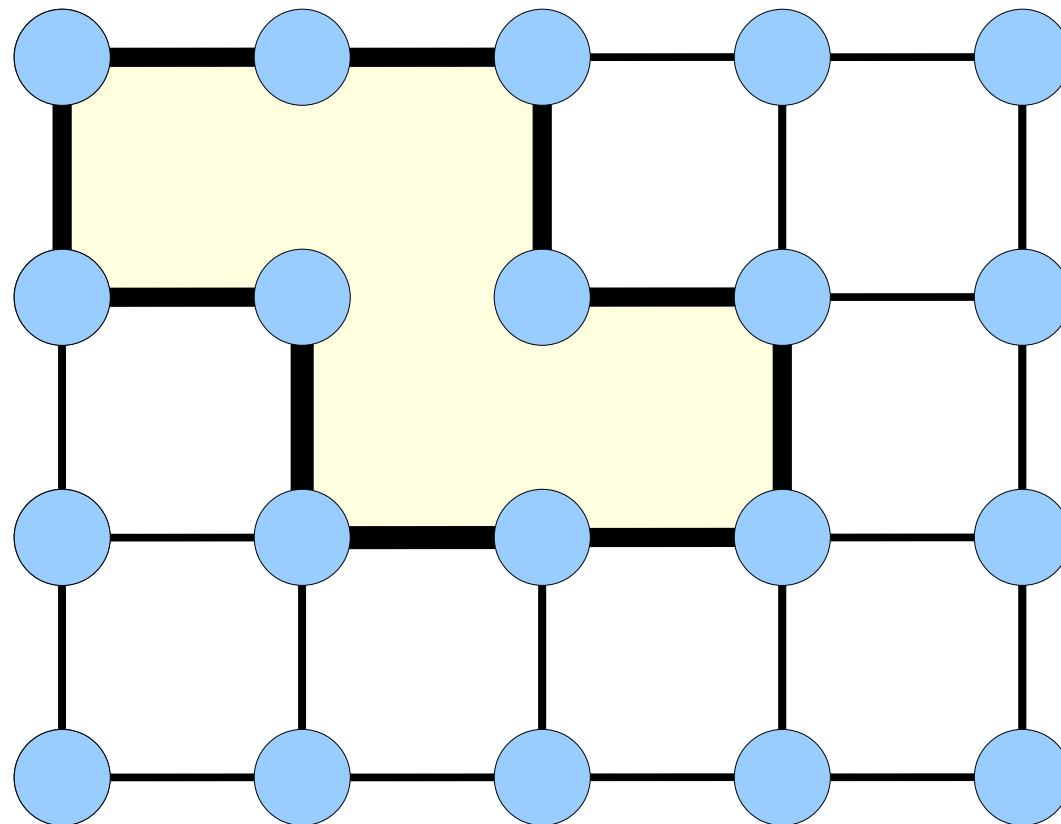
# Creating a Maze



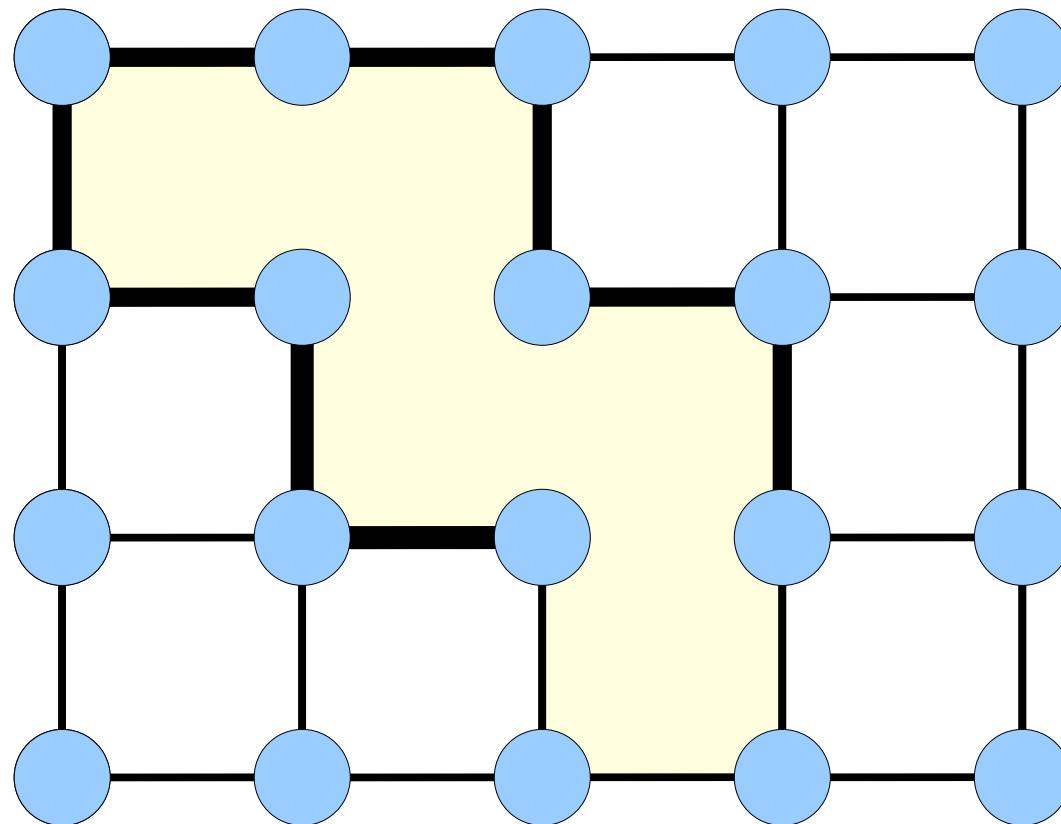
# Creating a Maze



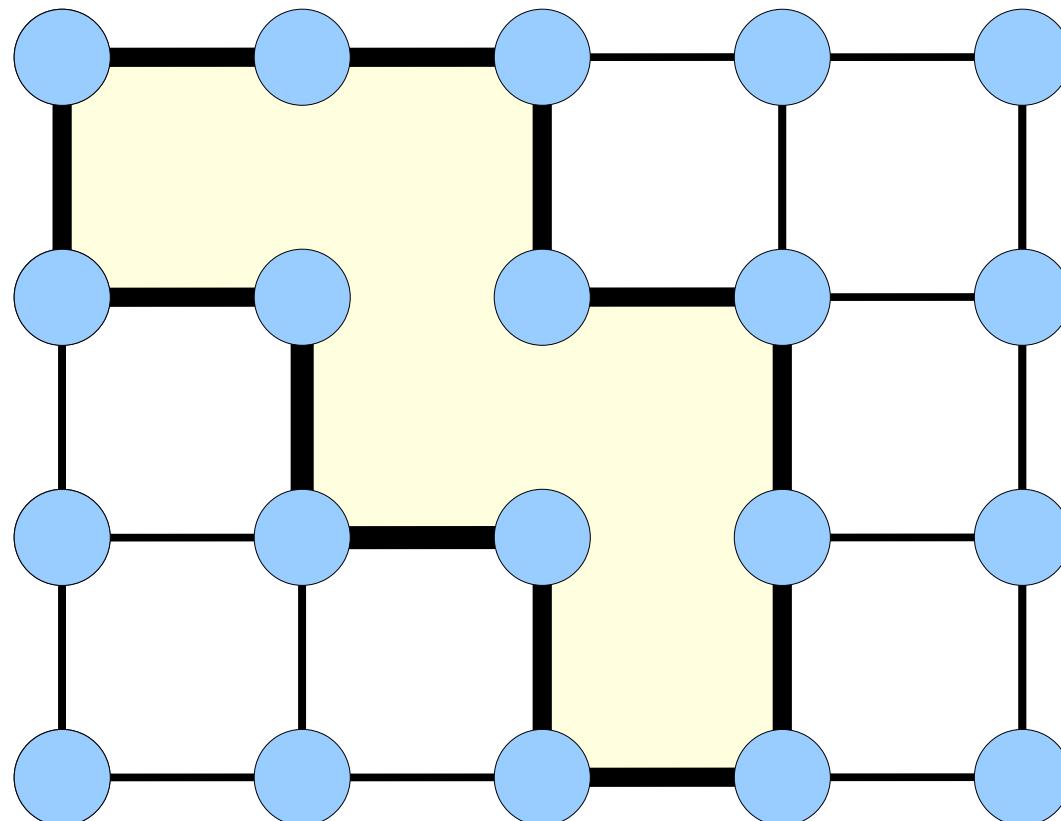
# Creating a Maze



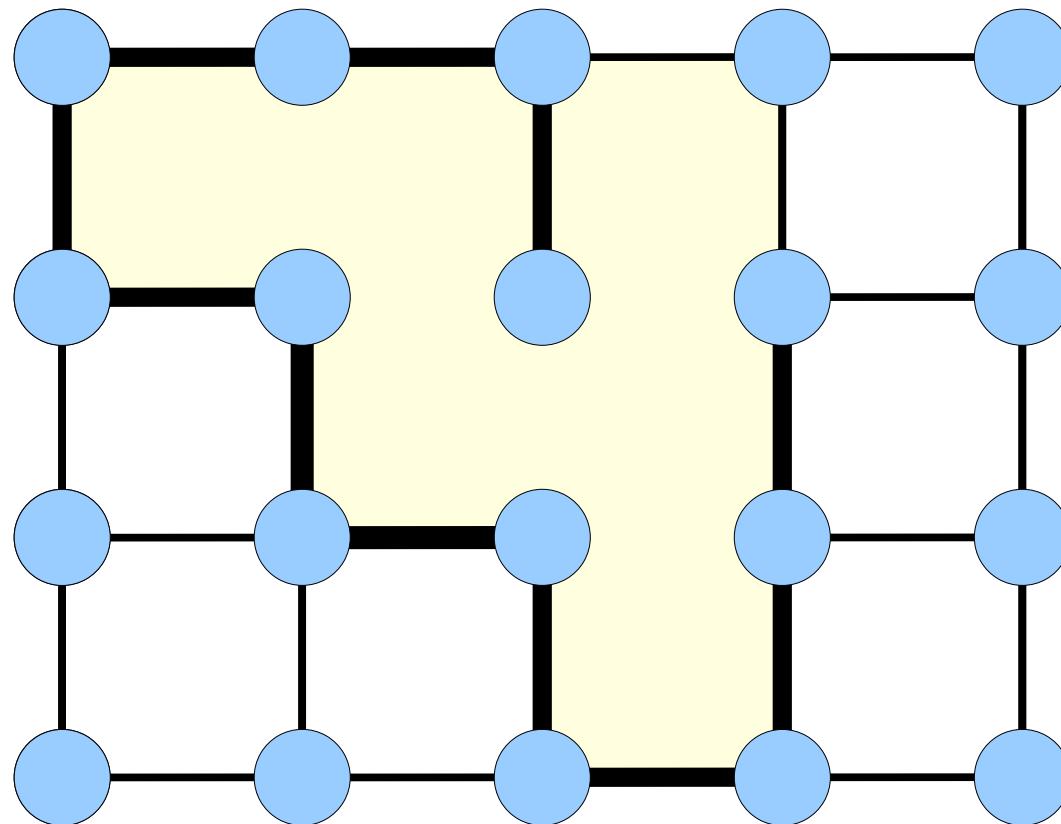
# Creating a Maze



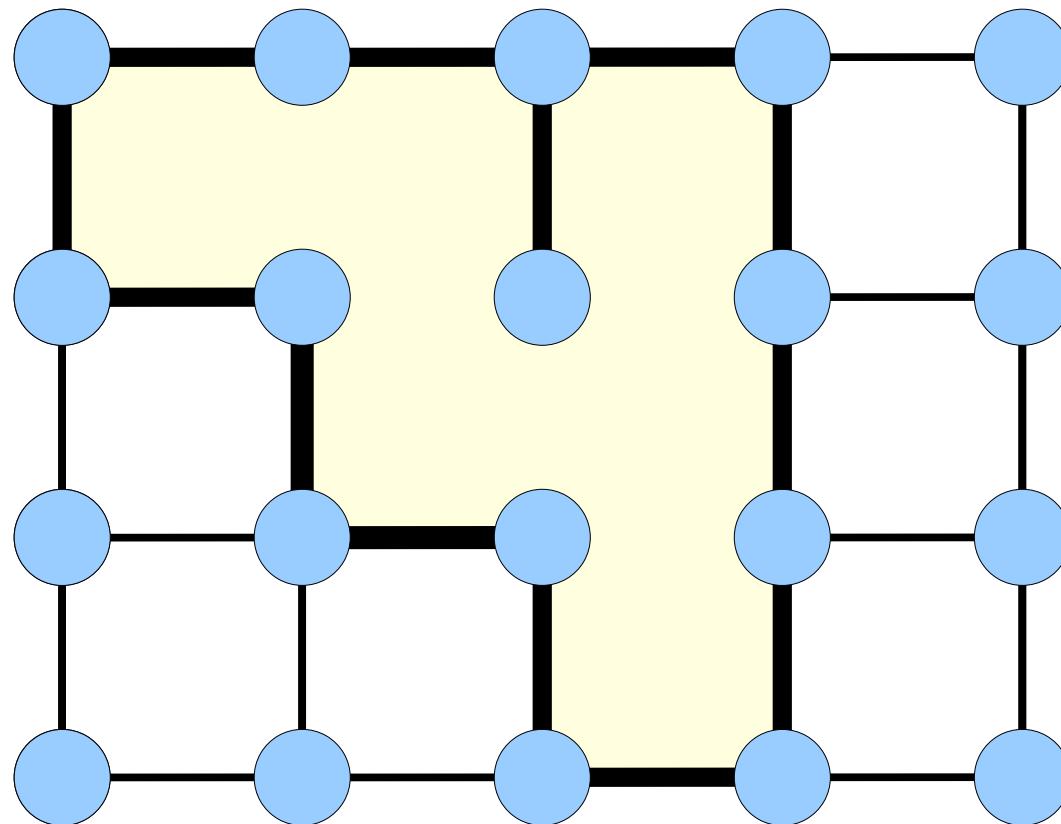
# Creating a Maze



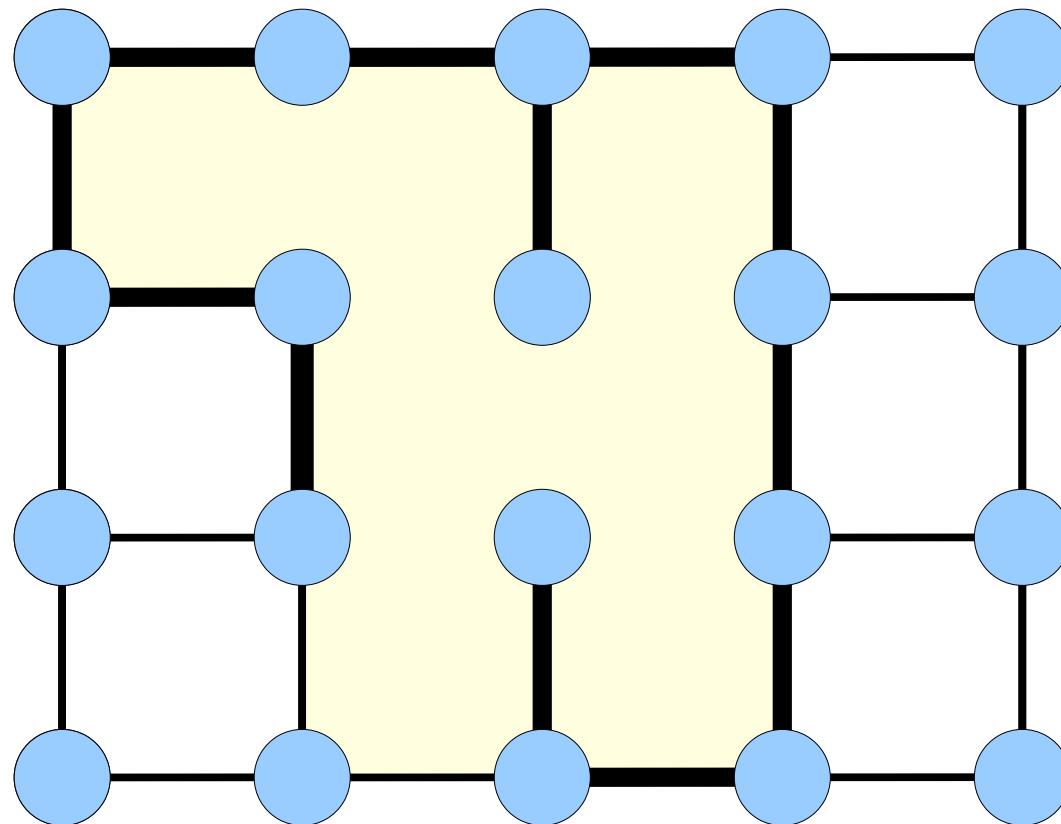
# Creating a Maze



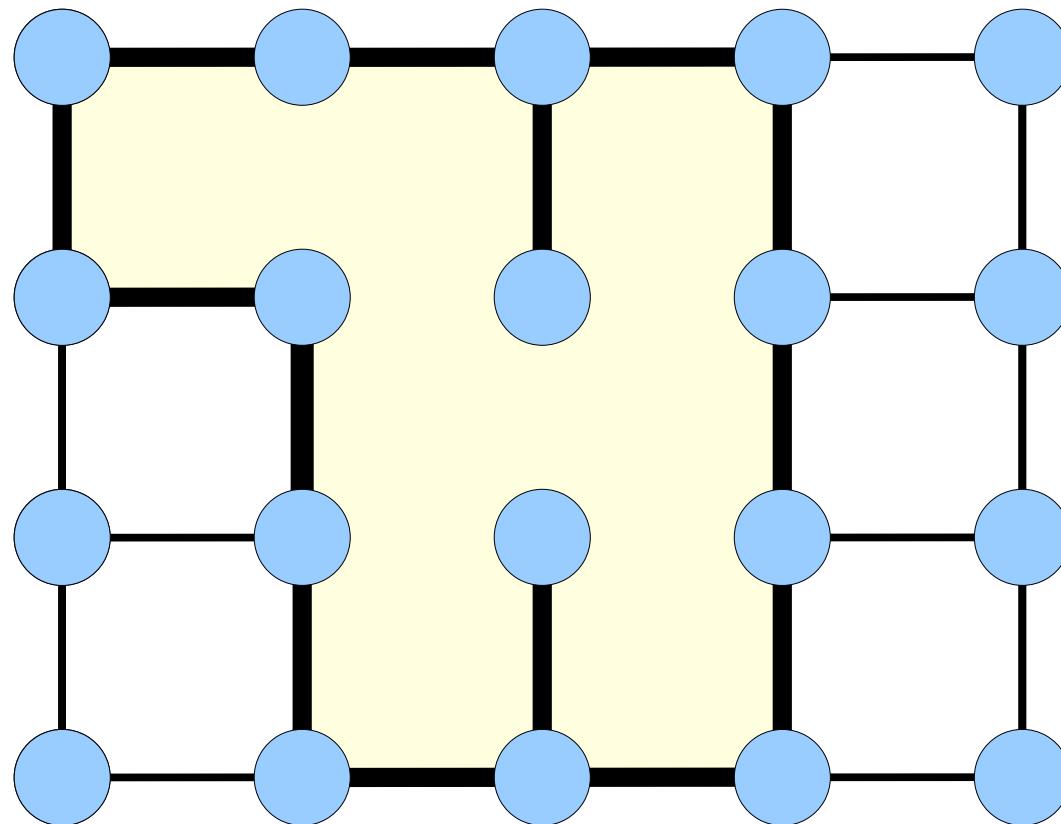
# Creating a Maze



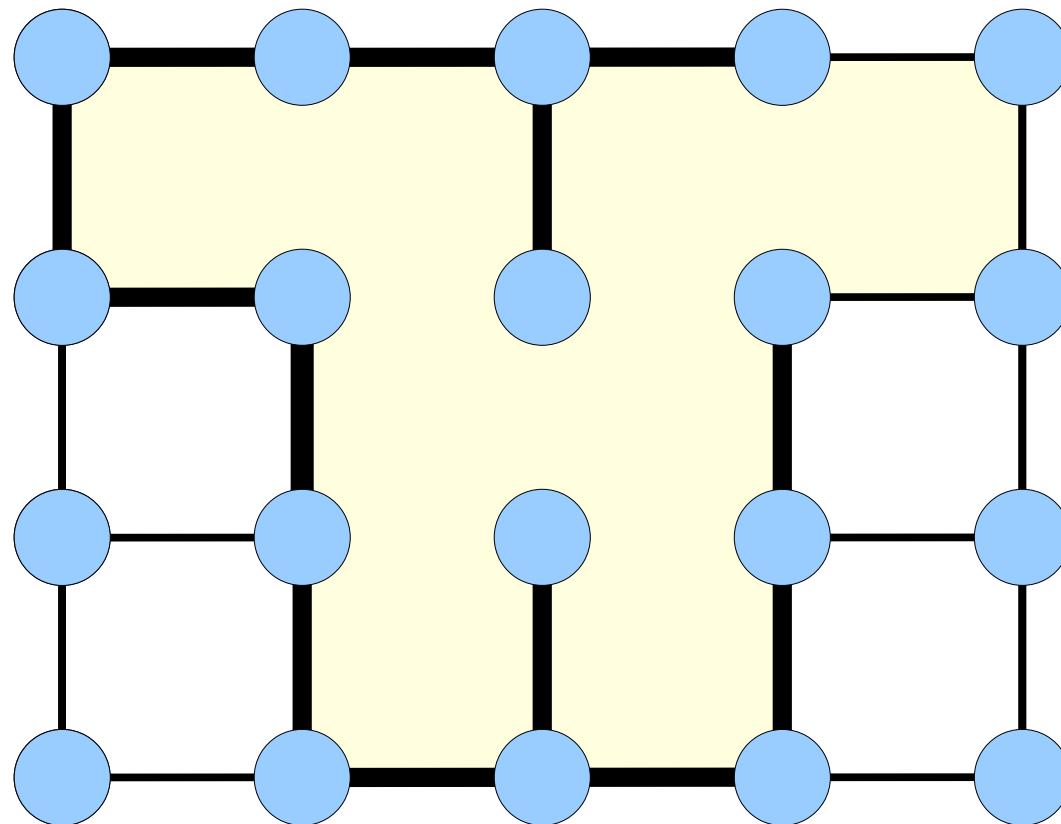
# Creating a Maze



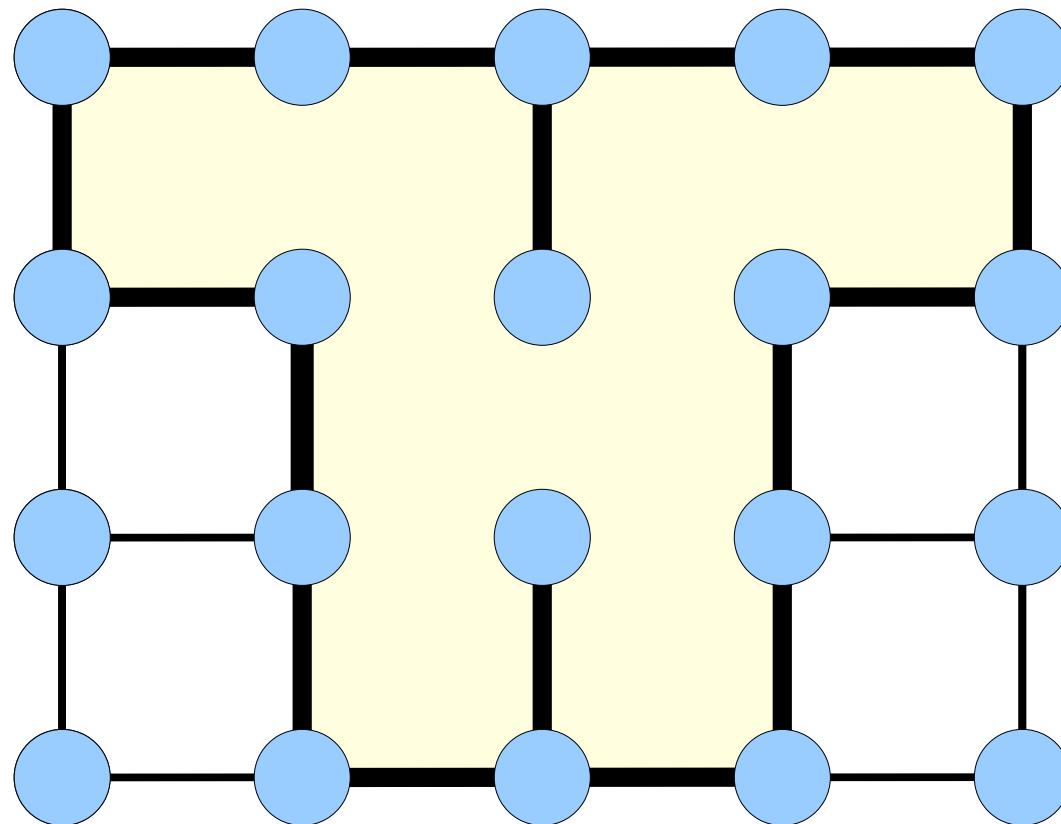
# Creating a Maze



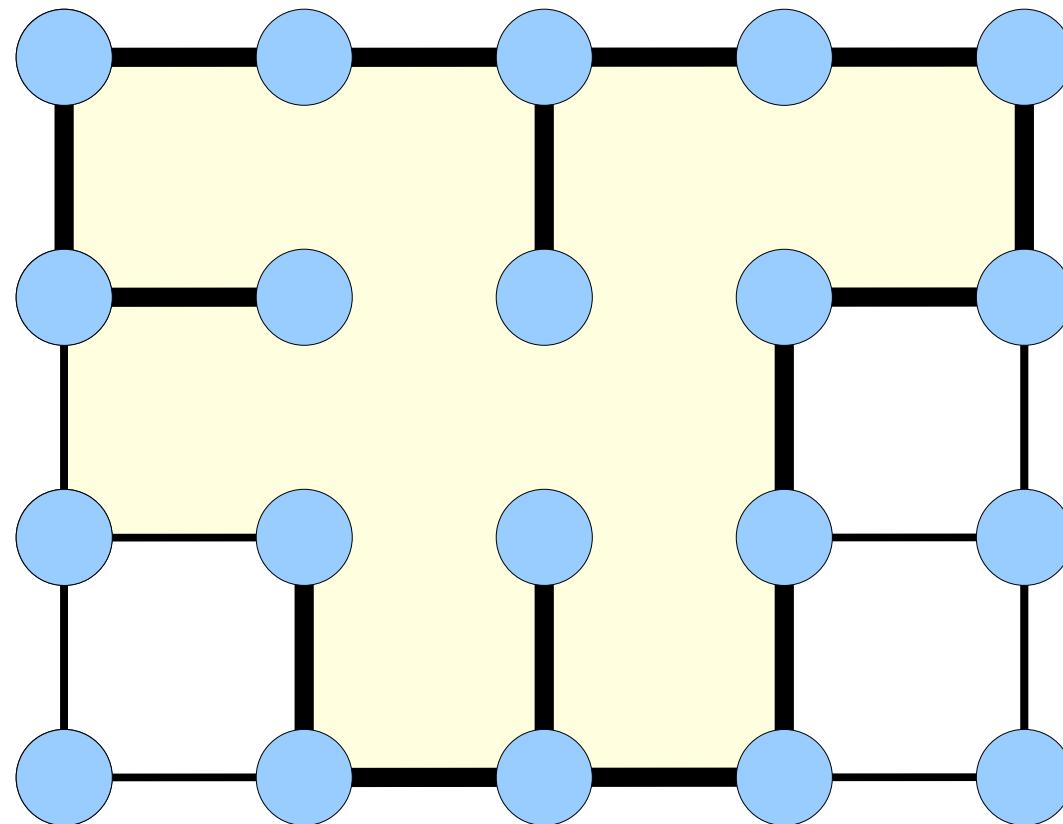
# Creating a Maze



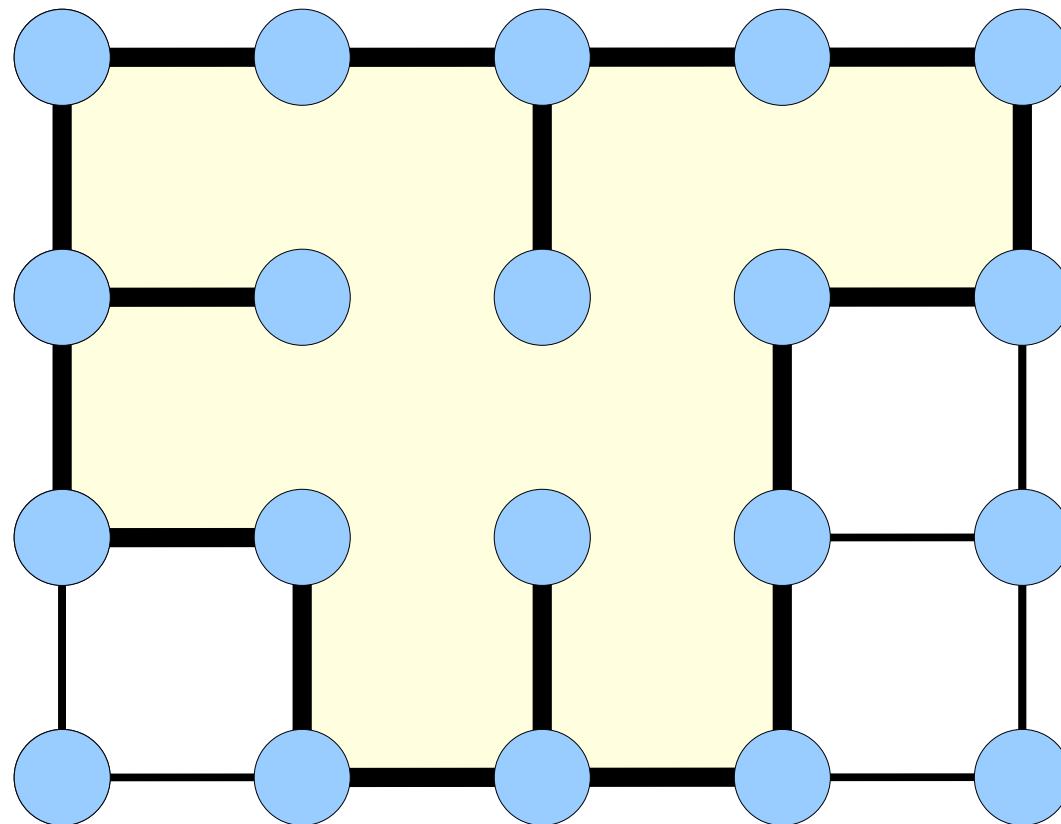
# Creating a Maze



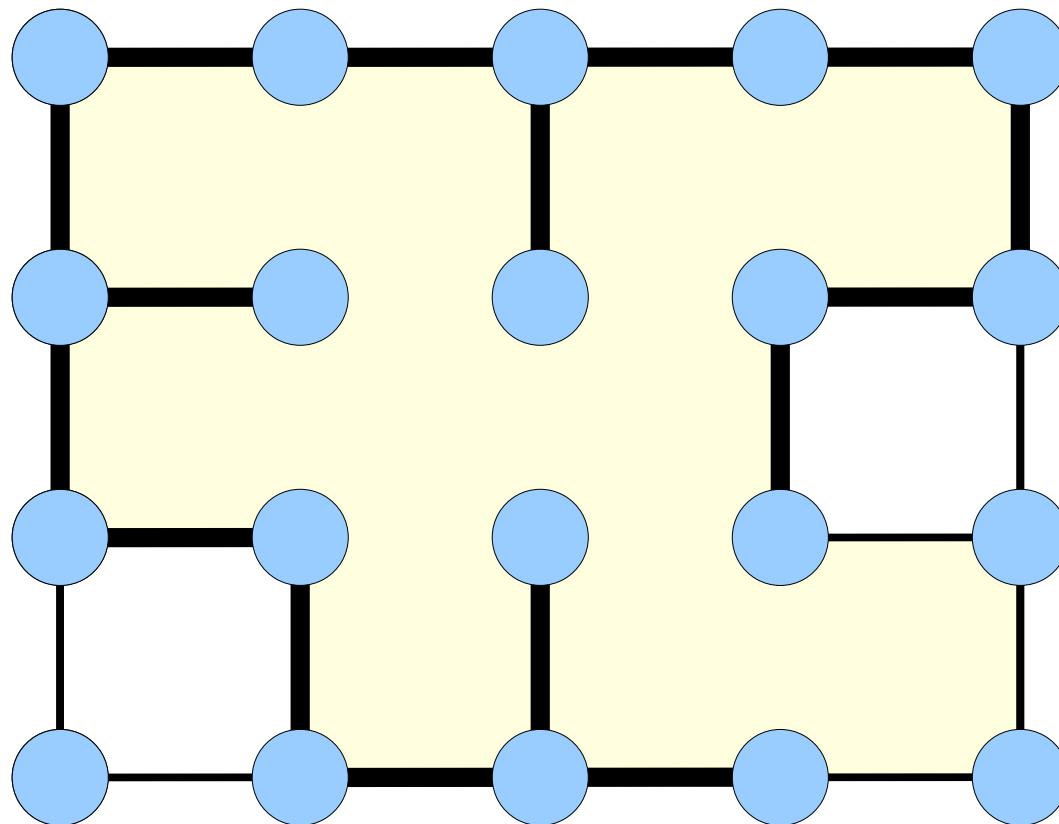
# Creating a Maze



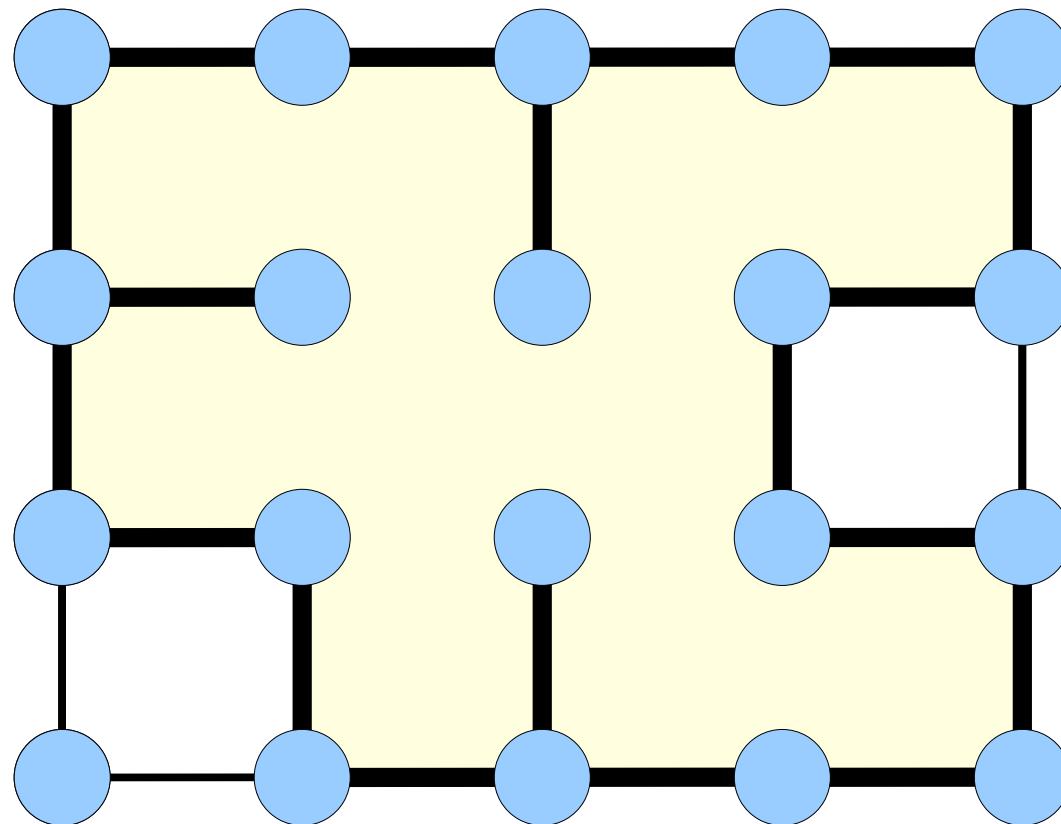
# Creating a Maze



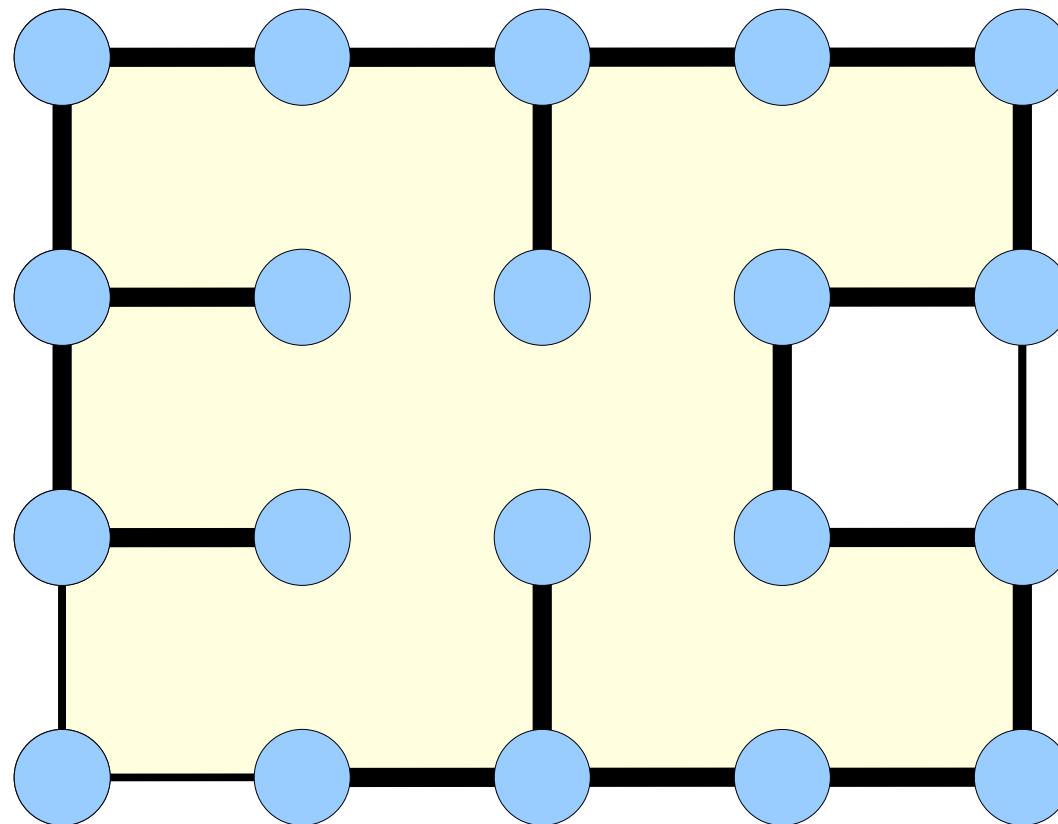
# Creating a Maze



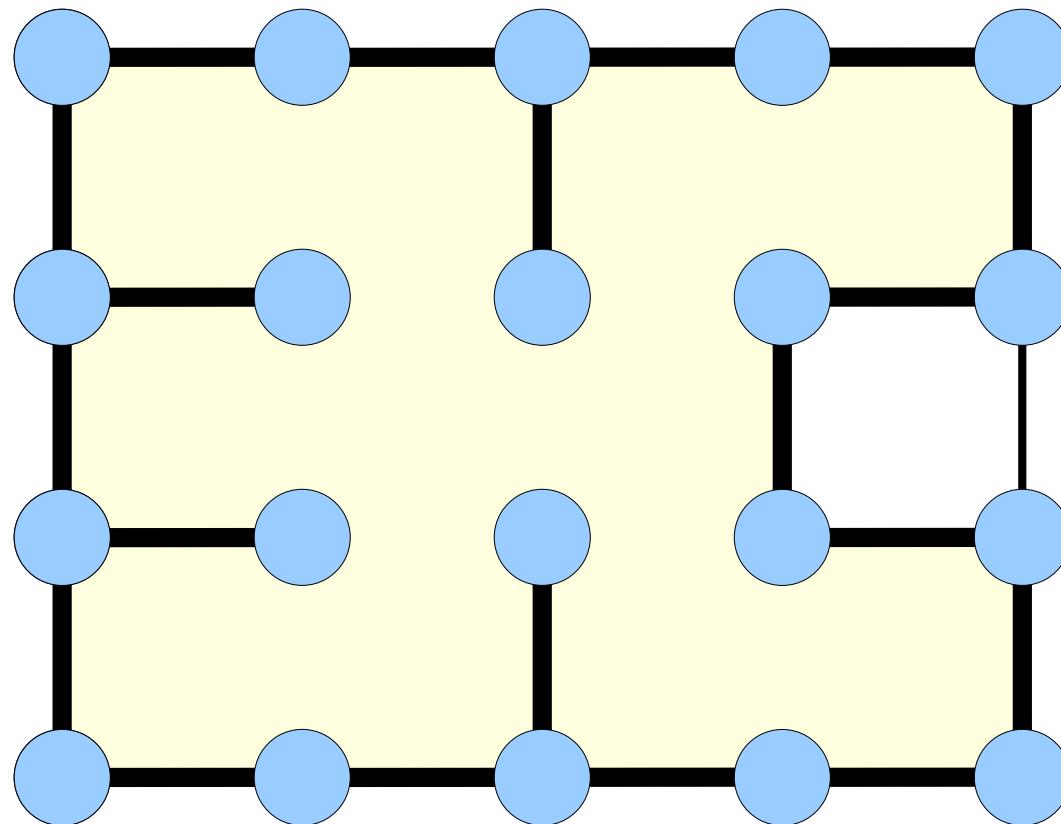
# Creating a Maze



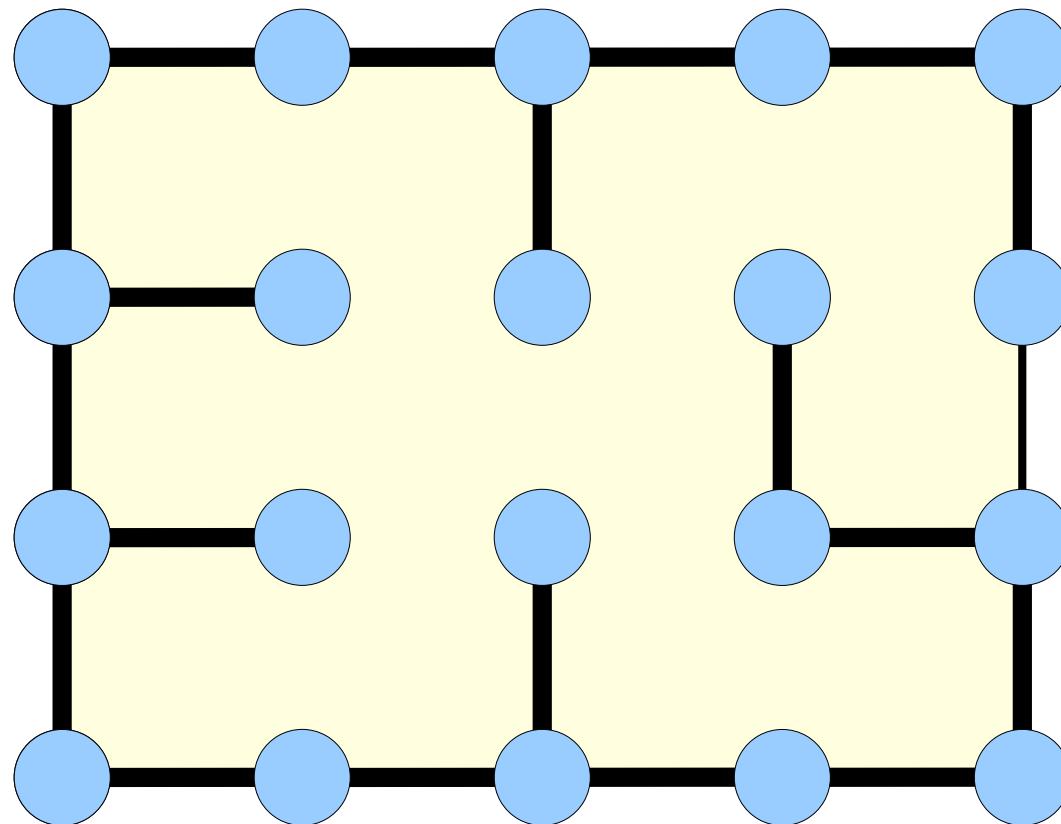
# Creating a Maze



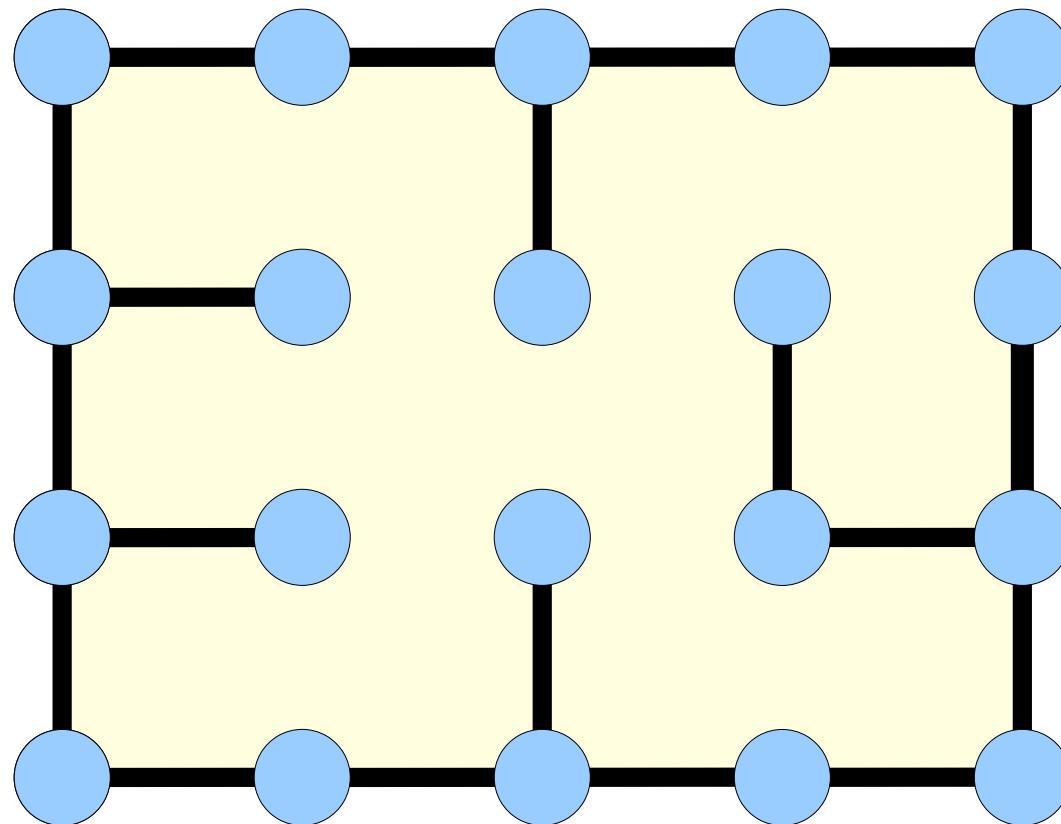
# Creating a Maze



# Creating a Maze



# Creating a Maze



# Generating Music

**<https://dice.humdrum.org/>**

# Random Bags

- A **random bag** is a container similar to a stack or queue. It supports two operations:
  - **add**, which puts an element into the random bag, and
  - **remove random**, which returns and removes a random element from the bag.
- This is exactly what we need for these examples:
  - **Bananagrams**: We **add** all the tiles to the bag. Whenever we need to draw, we call **remove random** to get another.
  - **Mazes**: We **add** the walls of the initial cell to the bag. Then we repeatedly **remove random** a wall, knock it down if it doesn't cause a loop, and add all walls of the new square into the maze.
  - **Generating Music**: We **add** all the music measures to the bag, then **remove random** enough to make a small tune.
- Unfortunately, there is no RandomBag type the way there's Stack, Queue, etc. How would we create one?

# Classes in C++

# Classes

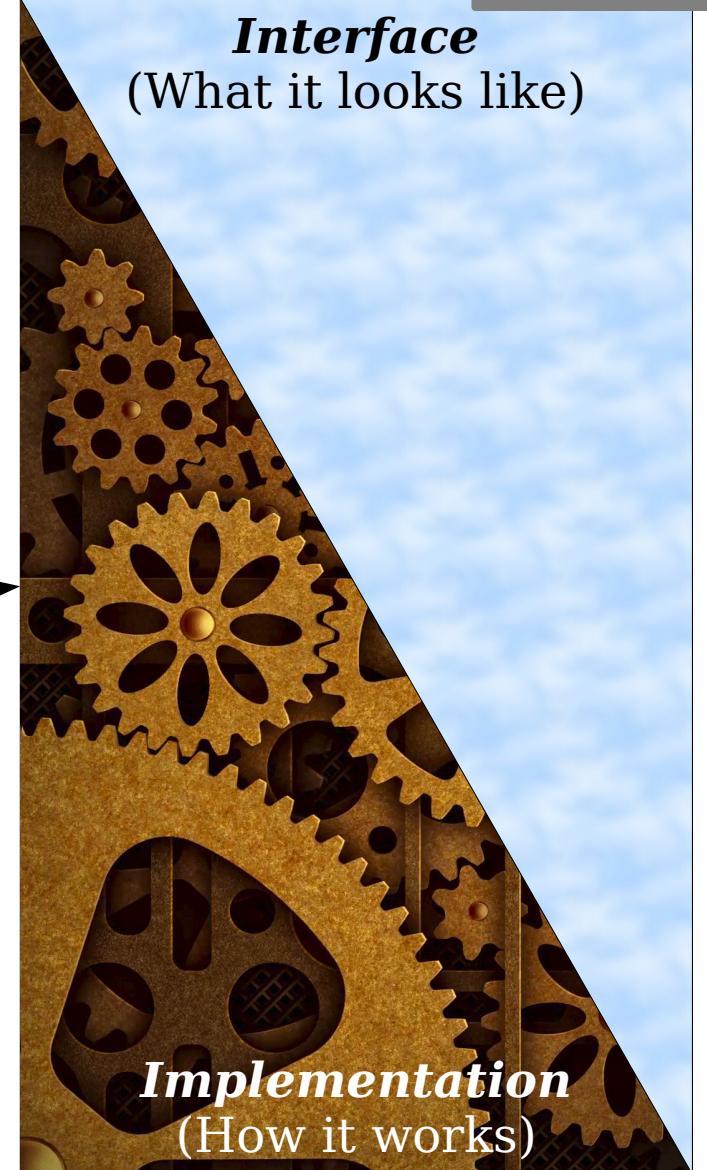
- Vector, Stack, Queue, Map, etc. are **classes** in C++.
- Classes contain
  - an **interface** specifying what operations can be performed on instances of the class.

***Interface***  
(What it looks like)

# Classes

- Vector, Stack, Queue, Map, etc. are **classes** in C++.
- Classes contain
  - an **interface** specifying what operations can be performed on instances of the class, and
  - an **implementation** specifying how those operations are to be performed.

Where  
we're going



Where  
we've  
been

# Classes

- A **class** provides a general pattern for what its **instances** look like.
  - `Vector` is a class; each variable of type `Vector` is an instance of the class.
- Each instance is a separate object, but all instances have the same common interface and set of behaviors.
- You can use a class through its interface even if you have no idea how it works behind the scenes.



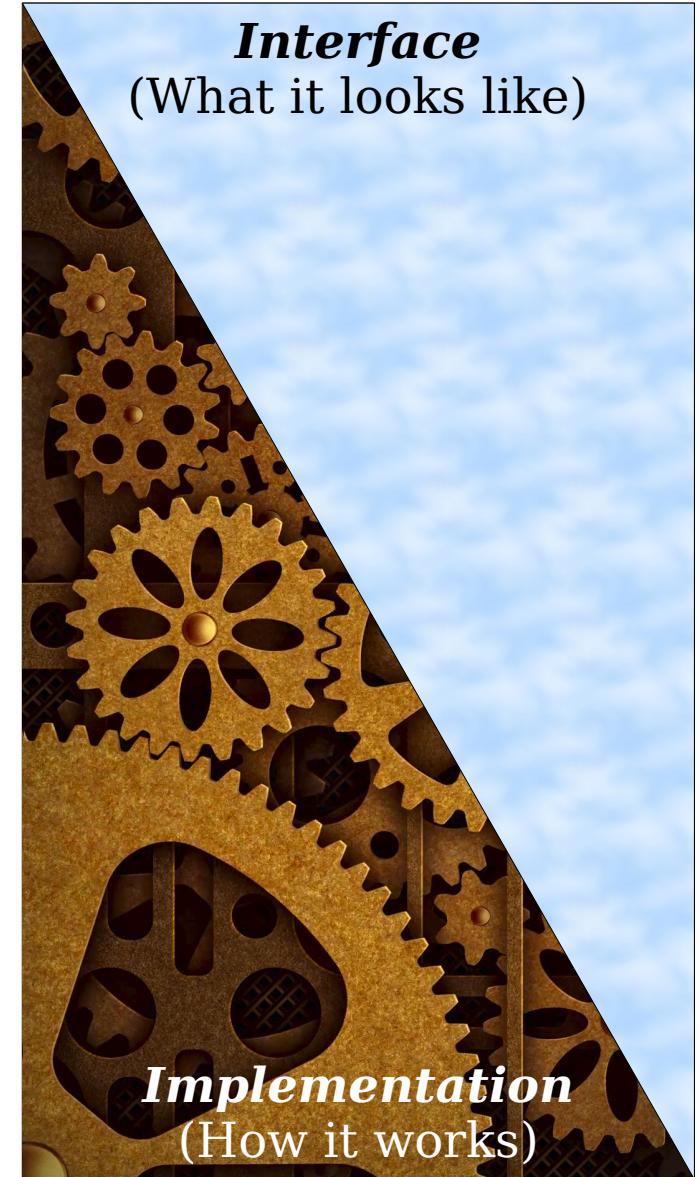
# Classes

- You can create your own classes in C++.
- This lets us expand our vocabulary of abstractions so we can solve more problems.
- Today we'll see how to make a RandomBag type we can use whenever we'd like.



# Classes

- Defining a class in C++ (usually) takes two steps:
  - Create a **header file** (ends in .h) specifying the interface and a few implementation details.
  - Create an **implementation file** (ends with .cpp) containing the class implementation.
- Clients of the class can then include the header file to use the class.



# What's in a Header?

# What's in a Header?

`#pragma once`

This is called an ***include guard***. It's used to make sure weird things don't happen if you include the same header twice.

Curious how it works?  
Come talk to me after  
class!

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
};
```

This is a ***class definition***. We're creating a new class called RandomBag. Like a struct, this defines the name of a new type that we can use in our programs.

# What's in a Header?

```
#pragma once
```

```
class RandomBag {
```

```
};
```

***Don't forget to add this semicolon!*** You'll get some Hairy Scary Compiler Errors if you leave it out.

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:  
  
private:  
};
```



# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:
```

```
private:
```

```
};
```

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:
```



```
private:
```

```
};
```

The ***public interface***

specifies what functions you can call on objects of this type.

Think things like the `Vector`'s `.add()` function or the `string`'s `.find()`.

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:
```

```
private:  
};
```



The ***public interface*** specifies what functions you can call on objects of this type.

Think things like the `Vector`'s `.add()` function or the `string`'s `.find()`.

The ***private implementation*** contains information that objects of the class type will need in order to do their job properly. This is invisible to people using the class.

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:  
    void add(int value);  
    int removeRandom();  
  
private:  
};
```

These are ***member functions*** of the RandomBag class. They're functions you can call on objects of the type RandomBag.

All member functions need to be declared in the class definition. We'll implement them in our .cpp file.

# What's in a Header?

```
#pragma once

#include "vector.h"

class RandomBag {
public:
    void add(int value);
    int removeRandom();

private:
    Vector<int> elems;
};
```

This is a ***data member*** of the class. This tells us how the class is implemented. Internally, we're going to store a `Vector<int>` holding all the elements. The only code that can access or touch this `Vector` is the `RandomBag` implementation.

# What's in a Header?

```
#pragma once

#include "vector.h"

class RandomBag {
public:
    void add(int value);
    int removeRandom();

private:
    Vector<int> elems;
};
```

```
class RandomBag {  
public:  
    void add(int value);  
    int removeRandom();  
  
private:  
    Vector<int> elems;  
};
```

```
class RandomBag {  
public:  
    void add(int value);  
    int removeRandom();  
    int size();  
    bool isEmpty();  
private:  
    Vector<int> elems;  
};
```

```
class RandomBag {  
public:  
    void add(int value);  
    int removeRandom();  
    int size() const;  
    bool isEmpty() const;  
private:  
    Vector<int> elems;  
};
```

# Your Action Items

- ***Prepare for the Midterm***
  - There's a review session this week, three practice exams up on the course website, and a huge searchable bank of practice problems to pull from. Best of luck - ***you can do this!***
- ***Read Chapter 6 of the textbook.***
  - There's a ton of goodies in there about class design that we'll talk about later on.

# Next Time

- ***Midterm on Monday - No Class***
- ***Then, When We Get Back...***
  - ***Dynamic Allocation***
    - Where does memory come from?
  - ***Constructors and Destructors***
    - Taking things out and putting them away.
  - ***Implementing the Stack***
    - Peering into our tools!