DESIGN DISCUSSION
Agenda

1. Representing the Board
2. BFS!
3. Containment and Inheritance
4. Code Incrementally!!!
The Board

- What data structure should we use to represent the board?
  - 2D array (23 x 23)

- What is the role of the SquareType enum map?
  - Saves you work! We tell you where the walls, energizers, and dots should be located
What even is an **enum**?! 

- A class that represents very simple labels and logic
  - e.g. “WALL” in the support map could be represented by 0s if the board was ints
- Why use **enums**?
  - Do you *reeeeally* want to try and remember that a 0 is a wall, a 1 is a dot, a 2 is a….etc. ?? Nah. Enums help make your code cleaner, easier to read, and easier to debug!
How to use the map of **SquareTypes**

Use it as a blueprint!

for row from 0 to 22:
  for col for 0 to 22:
    whatShouldIBuild = supportMap[row][col]
    /* insert code to build whatShouldIBuild in your own board at [row][col] */

So if the value of supportMap[row][col] == **WALL**, how would you handle this? What about if supportMap[row][col] == **ENERGIZER**?

**NOTE:** Once the board is set up, do we ever need to use the **supportMap** again?
Perks of making a **SmartSquare** class

- **SmartSquare**s know whether or not they are a wall
- Each square would then know what it contains (e.g. a **Dot**, **Energizer**, etc.)
  - Can a square contain more than one thing?
  - How to keep track of what it contains?
  - What type of objects can **SmartSquare** possibly contain?
Breadth-First Search

- When do we run BFS?
  - When Ghost has more than one valid move option
- What are the goals of BFS?
  1. Which square is closest to target
  2. Which direction gets you to that square the fastest
- What should the BFS return?
  - Direction that the Ghost should move in
Hand Simulate

method BFS():
  for each of the ghost’s valid neighboring squares:
    add the square’s location (the square’s position in the 2D array)
    to the queue and store the square’s direction in the 2D array,
    indexed by its location
  while the queue isn't empty:
    dequeue the next square location
    update the shortest distance and board coordinate
    for each valid neighbor of the current square
      if the neighbor has not been visited, add its location to the
      queue and store the current square’s direction in the 2D array
      at the neighbor’s x-y coordinates
  return the direction of the closest square
Let’s talk about BFS

- What is a “valid neighboring square”?  
  - At the start, don’t include direction that the ghost just came from  
  - Not a wall  
  - **Not already visited**  
  - Remember wrapping

- What is the “shortest distance”?  
  - Pythagorean distance  
    - In Java, you can use `Math.pow(double base, double power)`  
  - BFS can terminate if distance == 0  
  - Make sure that you store the board coordinate!
Visual Representation of BFS

TARGET

GHOST

= left  = up  = right
Help! My BFS isn’t working!

- How can we go about debugging our BFS?
  1. Hardcode the game to be in either **CHASE** or **SCATTER** mode so that you can test consistently without the ghosts switching between modes
  2. Test with Blinky (who directly targets Pacman) chasing Pacman
  3. Start with Pacman staying in one spot, then moving
  4. Test with ghosts in **SCATTER** mode going right to the corners
Containment and Inheritance

- What are some different classes we might want to have?
- What are some similarities and differences between Dots, Energizers, and Ghosts?
  - All can be collided with → Collidable?
Containment and Inheritance

- You’re going to have a lot of stuff going on in your PaneOrganizer, can we have lower-level graphical classes?
  - Heck yeah! If you feel that your PaneOrganizer is getting full, think about making a SideBar class, which would contain all of the various labels that correspond to the Game
  - How would we make sure that Game knows about SideBar, so that it can update labels accordingly?
Activity: The Ghost Pen!

When each ghost is eaten, it returns to the ghost pen. The order that the ghosts leave in is first in, first out. So what data structure should we be using to model the ghost pen?
(I) Stack
(II) Queue
(III) Array
(IV) Arraylist

It’s a queue! Queue is a first-in, first-out data structure. So the ghost entering the ghost pen first also leaves first!
So how do I actually get started?!

CODING INCREMENTALLY:
What are some possible plans for coding incrementally?
→ Discuss with a partner and come up with a plan!

START EARLY:
This is the biggest project you’ve been assigned so far--please do not wait until the last minute to begin!

Start Early... Start Today... Start Yesterday!!!
GOOD LUCK!
YOU GOT THIS!! :)