CSEN 171 - Practicum 3

The N Pieces Problem (Java)

Plan

First 30 mins: Walk through the entire practicum and implement some portions

- If you pay attention half of the practicum will be done by then

Remaining 70: Implement unfinished & remaining portions of the practicum

- Prof and I will be here to answer any questions

After class: Whatever's left over you will complete before the due date

- Refer back to these slides if you're struggling

Due Tuesday Nov 11 at 11:59pm

Setup

- Download & open Practicum3.java from Camino
- Practicum3.java in a new folder
  - Compiling this generates a bunch of .class files
- You will see a detailed description of the practicum on the top of this file

```
/**
 * PRACTICUM 3 STARTER CODE: THE N PIECES PROBLEM
 * This is basically leetcode 51 n queens with other pieces
 *
 * GOAL: Place N identical pieces on an N x N board so none attack each other.
 * This starter keeps the structure but leaves key pieces for you to implement.
 *
 * Complete the TODOs (numbered) below. Each unimplemented part currently
 * throws UnsupportedOperationException so you can compile early.
 * When you finish all TODOs, the program should work as specified.
 * TODOs:
 *   (1) PieceFactory.create [PARTIAL]
 *   (2) Knight.menaces [FULL]
 *   (3) Rook.menaces
 *   (4) Bishop.menaces
 *   (5A) Queen.place [PARTIAL]
 *   (5B) Queen.menaces [PARTIAL]
 *   (6A) Amazon.place
 *   (6B) Amazon.menaces
 *   (7) Renderer.asciiBoard [PARTIAL]
 *   (8) Renderer.coordsList [PARTIAL]
 *   (9) Solver.backtrack [PARTIAL]
 * DON'T MESS WITH CLASSES WITHOUT TODOs!!
 * Comment out the "throw new ..." lines and replace with correct code.
 *
 * TO COMPILE: javac Practicum3.java
 *   Make sure you can run javac (search it up)
 * TO RUN: echo [N] | java Practicum3 [piece] [mode] [k]
 * WHERE:
 *   [N] = board size (positive integer)
 *   [piece] = piece type (knight|rook|bishop|queen|amazon)
 *   [mode] = mode (count|one|list)
 *   [k] = integer for list mode (positive integer)
 *
 * TRY THESE WHEN UR DONE:
 *   echo 8 | java Practicum3
 *     u should get 92 (classic 8 queens problem)
 *   echo 8 | java Practicum3 queen one
 *     results in board & coords for one of 92 solutions
 *   echo 8 | java Practicum3 rook list 3
 *     results in 3 boards & coord lists
 */
```

The Task

This practicum is similar to leetcode 51 n queens but with other pieces

- GOAL: Place **N** identical pieces on an **N** x **N** board so none attack each other.
- FOR EXAMPLE: There are **92** ways to place **8** queens on an **8** x **8** board so none attack each other.

- There are 9 TODOs to implement in order for us to achieve this

The Task

- Your task is to complete the TODOs (numbered) below.
  - (1) PieceFactory.create [PARTIAL]
  - (2) Knight.menaces [FULL]
  - (3) Rook.menaces
  - (4) Bishop.menaces
  - (5A) Queen.place [PARTIAL]
  - (5B) Queen.menaces [PARTIAL]
  - (6A) Amazon.place
  - (6B) Amazon.menaces
  - (7) Renderer.asciiBoard [PARTIAL]
  - (8) Renderer.coordsList [PARTIAL]
  - (9) Solver.backtrack [PARTIAL]

Implementing TODOs

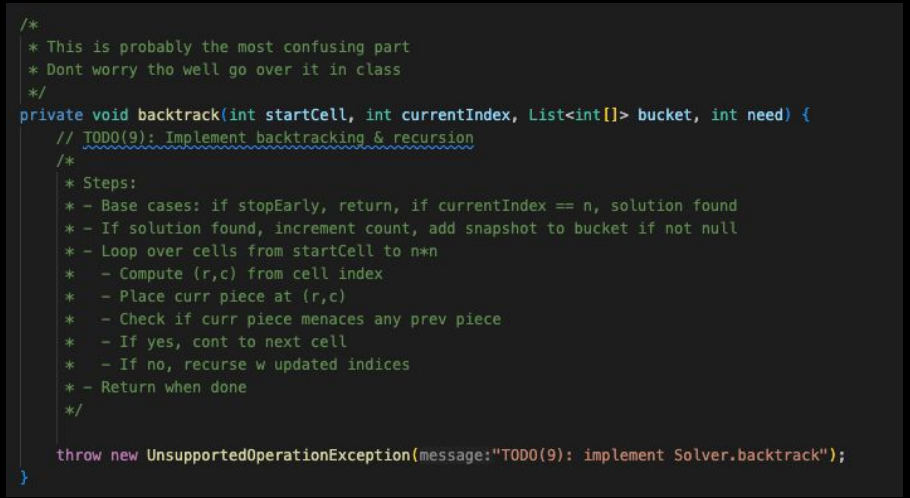
- DON'T MESS WITH CLASSES WITHOUT TODOs!!
- Comment out the "throw new ..." lines then replace with your implementation.

```
static class Knight extends Piece {
    @Override
    public boolean menaces(Piece p) {
        //TODO(2): Knights attack in an L shape: (|dr|,|dc|) = (2,1) or (1,2)
        //Replace the exception with a correct boolean expression

        throw new UnsupportedOperationException(message:"TODO(2): implement Knight.menaces");
    }
}
```

Implementing TODOs

- Each unimplemented part currently throws UnsupportedOperationException.
  - That way you can compile early and know what to implement next.



```java
/*
 * This is probably the most confusing part
 * Dont worry tho well go over it in class
 */
private void backtrack(int startCell, int currentIndex, List<int[]> bucket, int need) {
    // TODO(9): Implement backtracking & recursion
    /*
     * Steps:
     * - Base cases: if stopEarly, return, if currentIndex == n, solution found
     * - If solution found, increment count, add snapshot to bucket if not null
     * - Loop over cells from startCell to n*n
     *   - Compute (r,c) from cell index
     *   - Place curr piece at (r,c)
     *   - Check if curr piece menaces any prev piece
     *   - If yes, cont to next cell
     *   - If no, recurse w updated indices
     * - Return when done
     */

    throw new UnsupportedOperationException(message:"TODO(9): implement Solver.backtrack");
}
```

How to Compile

- TO COMPILE: javac Practicum3.java
  - See next slide if you get an error
- TO RUN: echo [N] | java Practicum3 [piece] [mode] [k]

- where…
  - [N] = board size (positive integer)
  - [piece] = piece type (knight|rook|bishop|queen|amazon)
  - [mode] = mode (count|one|list)
  - [k] = integer for list mode (positive integer)

javac (skip if installed)

- **Windows**
  - winget install EclipseAdoptium.Temurin.17.JDK
- **Mac (Silicon)**
  - brew install openjdk@17
    - See next slide if you don’t have homebrew
  - echo 'export PATH="/opt/homebrew/opt/openjdk@17/bin:$PATH"' >> ~/.zshrc
  - source ~/.zshrc
- **Mac (Intel)**
  - brew install openjdk@17
    - See next slide if you don’t have homebrew
  - echo 'export PATH="/usr/local/opt/openjdk@17/bin:$PATH"' >> ~/.zshrc
  - source ~/.zshrc

javac -version

- If it shows javac 17… or later then it’s successfully installed

Homebrew for mac (skip if windows / installed)

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

- **Mac (Silicon)**
  - echo 'export PATH="/opt/homebrew/bin:$PATH"' >> ~/.zshrc
  - source ~/.zshrc
- **Mac (Intel)**
  - echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.zshrc
  - source ~/.zshrc

brew --version

- If it shows Homebrew… then it’s successfully installed

EXAMPLE 1

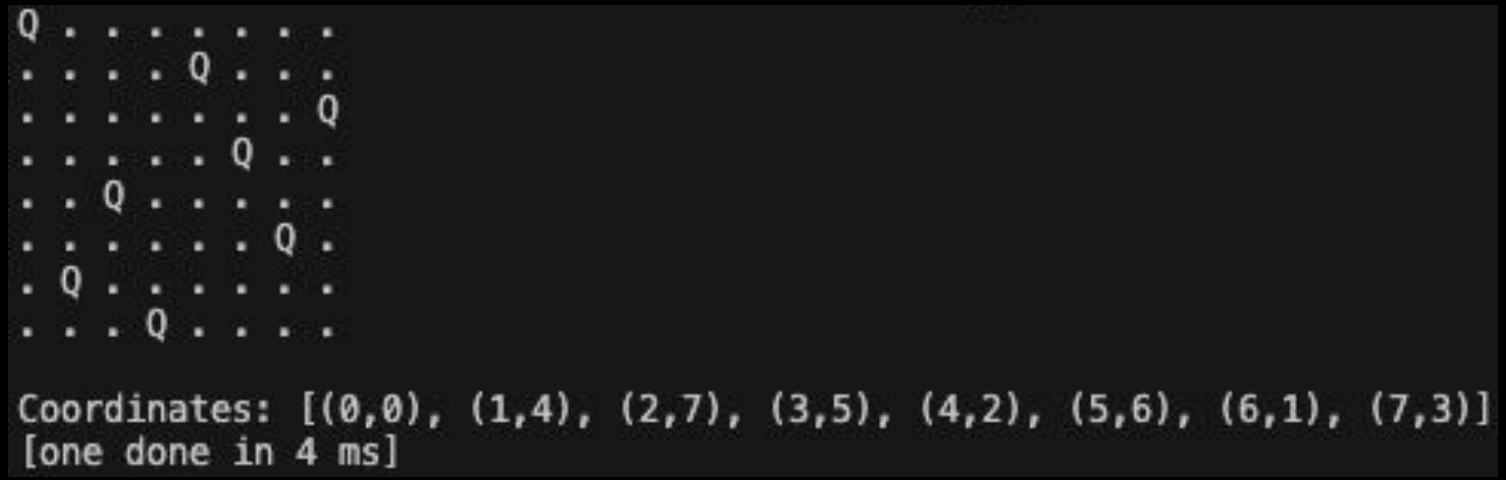
- echo 8 | java Practicum3

```
92
[count done in 31 ms]
```

RECALL: 92 possible ways to configure **8** queens on an **8x8** chessboard such that no two queens threaten each other

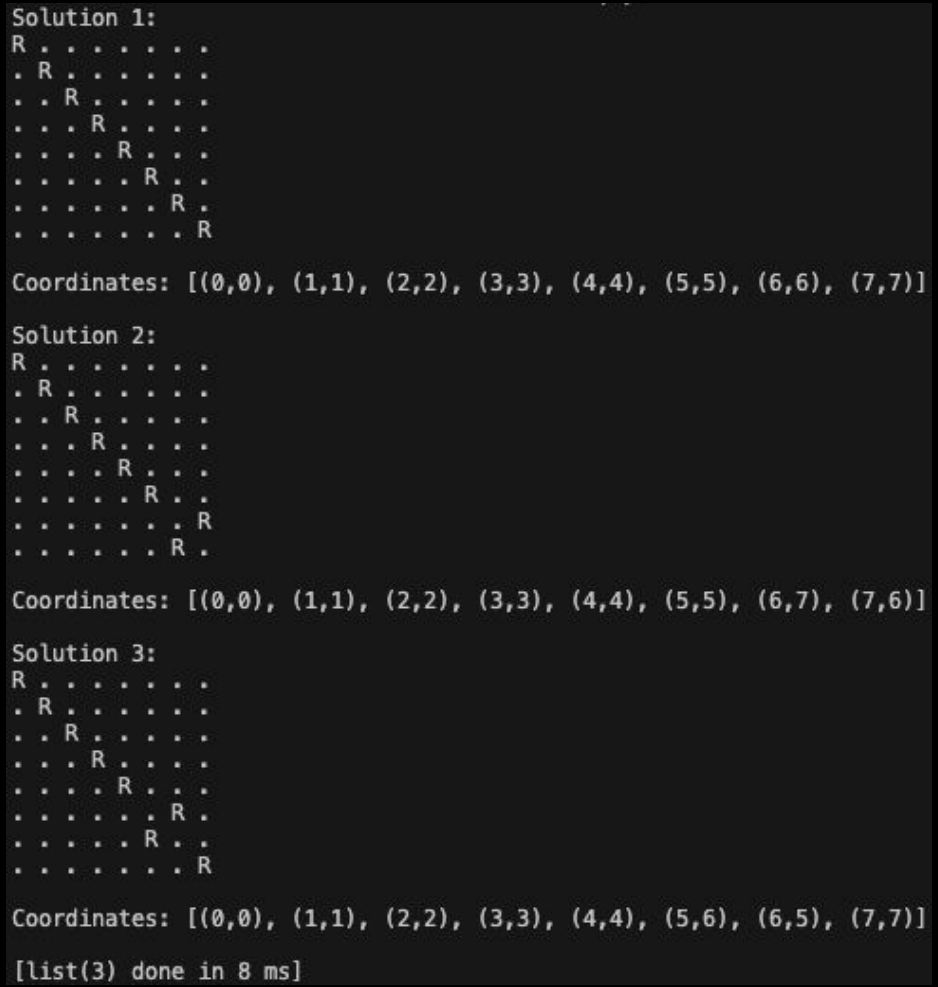
EXAMPLE 2

- echo 8 | java Practicum3 queen one



EXAMPLE 3

- echo 8 | java Practicum3 rook list 3



EXAMPLE 4 (Try this rn)

- TO COMPILE: javac Practicum3.java
- echo 8 | java Practicum3 (without implementing anything)

```
Exception in thread "main" java.lang.UnsupportedOperationException: TODO(1): implement PieceFactory.create
        at Practicum3$PieceFactory.create(Practicum3.java:233)
        at Practicum3$Solver.<init>(Practicum3.java:290)
        at Practicum3.main(Practicum3.java:64)
```

- So now let’s implement TODO(1): implement PieceFactory.create

TODO 1 - Piece Factory

This is responsible for creating chess piece objects based on the type of piece requested. This includes Knight, Rook, Bishop, Queen, and Amazon.

```
static final class PieceFactory {
    private final PieceType type;
    PieceFactory(PieceType type) { this.type = type; }

    Piece create() {
        //TODO(1): return a new instance based on type
        //up to u but i would use switch case default
        //so something like: KNIGHT -> new Knight(), ROOK -> new Rook(), etc

        throw new UnsupportedOperationException(message:"TODO(1): implement PieceFactory.create");
    }
}
```

TODO 1 - Piece Factory

Intuitively the best way to achieve this is via switch

```
switch(expression)
{
  case x:
    return;
  case y:
    return;
  default:
    return;
}
```

TODO 1 - Piece Factory

But implement it however you want

```
Piece create() {
    //TODO(1): return a new instance based on type
    //up to u but i would use switch case default
    //so something like: KNIGHT -> new Knight(), ROOK -> new Rook(), etc

    // throw new UnsupportedOperationException("TODO(1): implement PieceFactory.create");
    switch(type)
    {
    case KNIGHT:
        return new Knight();
    case y:
        return;
    //more cases
    default:
        //default case is queen
    }
}
```

TODO 2 - Knight

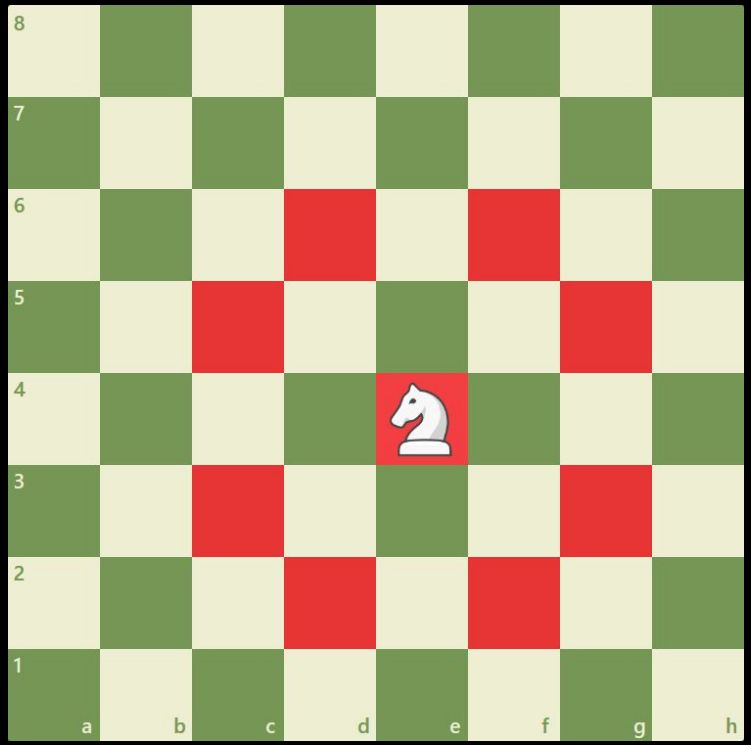
TODOs 2 to 6 are very similar

```
static class Knight extends Piece {
    @Override
    public boolean menaces(Piece p) {
        //TODO(2): Knights attack in an L shape: (|dr|,|dc|) = (2,1) or (1,2)
        //Replace the exception with a correct boolean expression

        throw new UnsupportedOperationException(message:"TODO(2): implement Knight.menaces");
    }
}
```

TODO 2 - Knight

As most of you know…



TODO 2 - Knight

Knights move in an L shape

So the difference in row and col would be either

- |dr| = 2 and |dc| = 1

OR

- |dr| = 1 and |dc| = 2

Where

dr = |this.row - p.row()|

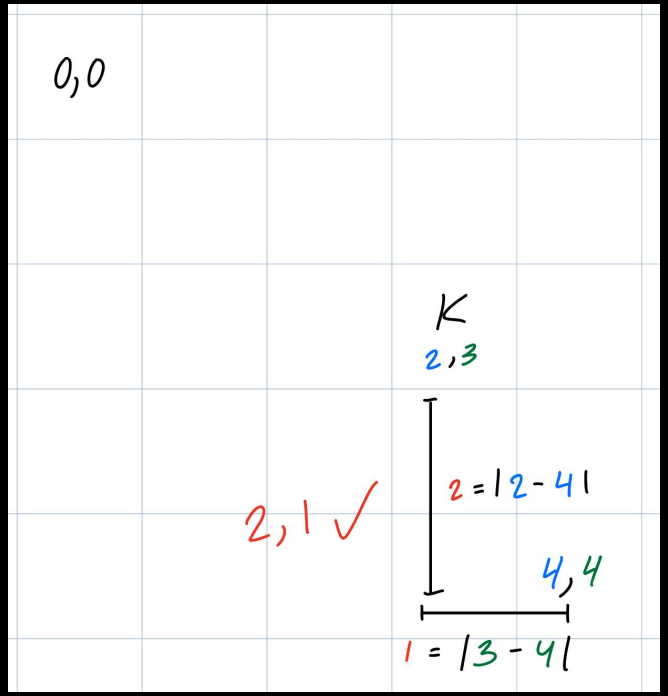
dc = |this.column - p.column()|

TODO 2 - Knight

Suppose Knight is at (2,3)

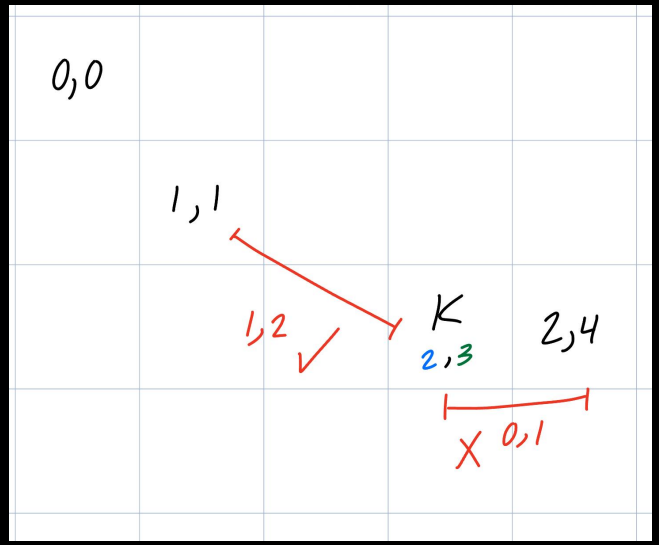
Can it attack (4,4)?

- |2-4|=2, |3-4|=1 -> Yes because 2,1



TODO 2 - Knight

Suppose Knight is at (2,3)



(1,1)?

- |2-1|=1, |3-1|=2 -> Yes because 1,2

Can it attack (2,4)?

- |2-2|=0, |3-4|=1 -> No because 0,1 (NOT 2,1 or 1,2)

Now try implementing this yourself

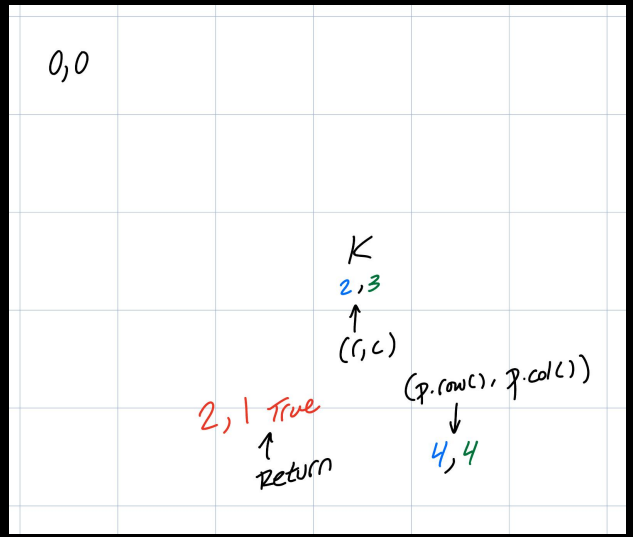
Given:

r = knight's row

c = knight's column

p.row() = row in question

p.column() = column in question



Return: whether or not a knight in position (r,c) menaces a piece in position (p.row(), p.column())

TODO 2 - Knight

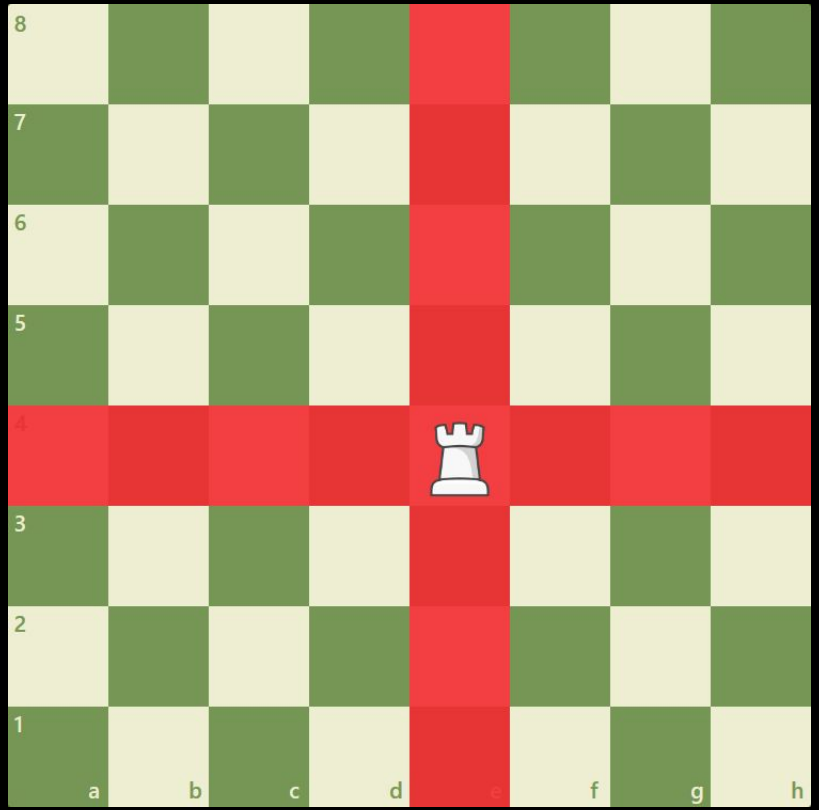
So now this should make a lot more sense

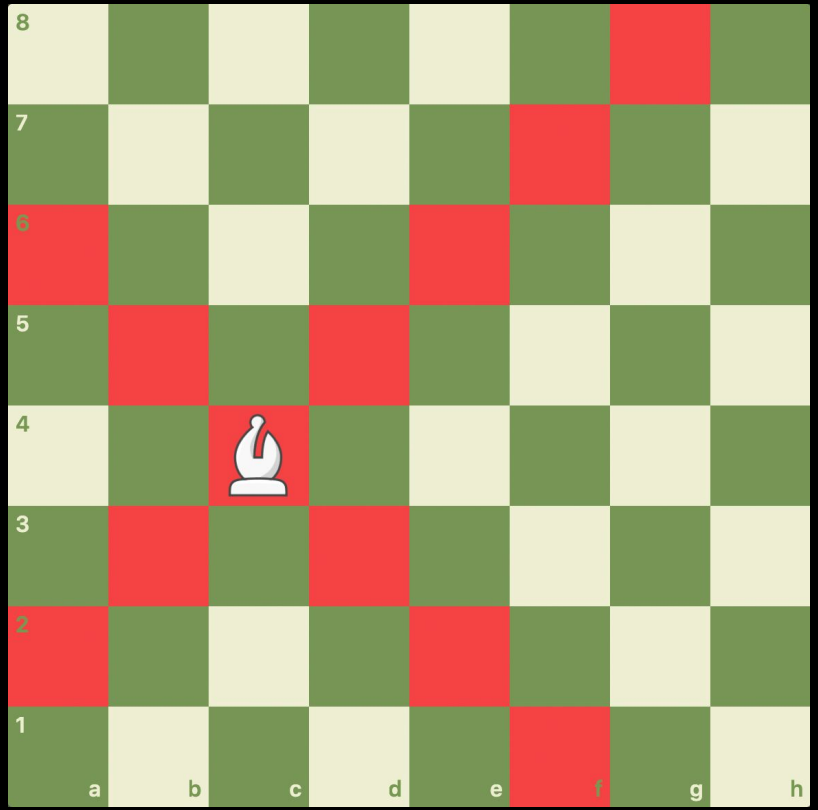
```
static class Knight extends Piece {
    @Override
    public boolean menaces(Piece p) {
        //TODO(2): Knights attack in an L shape: (|dr|,|dc|) = (2,1) or (1,2)
        //Replace the exception with a correct boolean expression

        // throw new UnsupportedOperationException("TODO(2): implement Knight.menaces");
        int dr = Math.abs(r - p.row());
        int dc = Math.abs(c - p.column());
        return (dr == 2 && dc == 1) || (dr == 1 && dc == 2);
    }
}
```

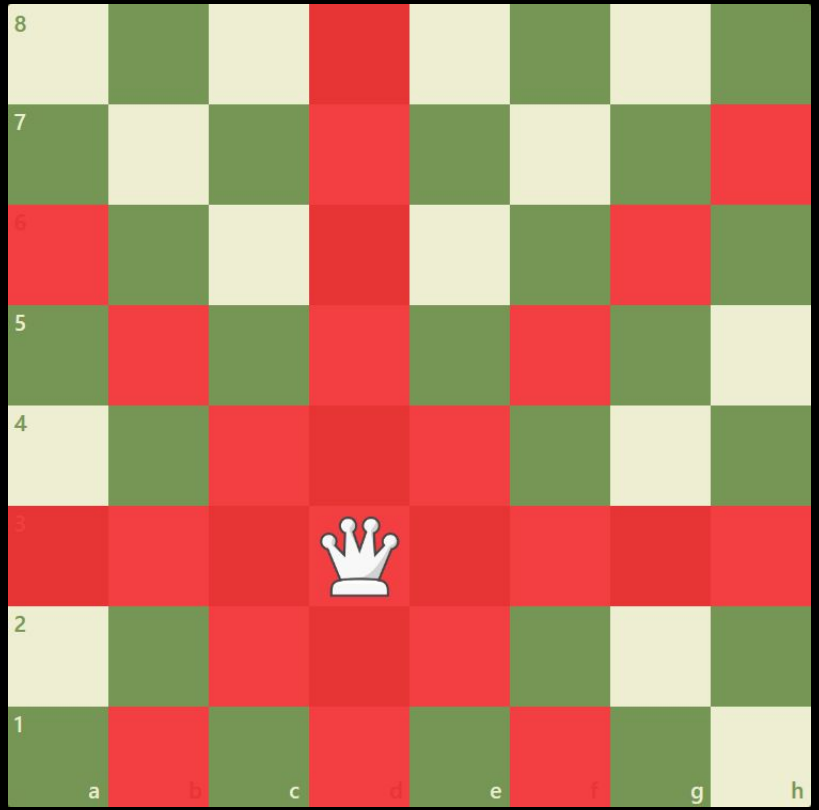
Now implementing other pieces should be fairly intuitive

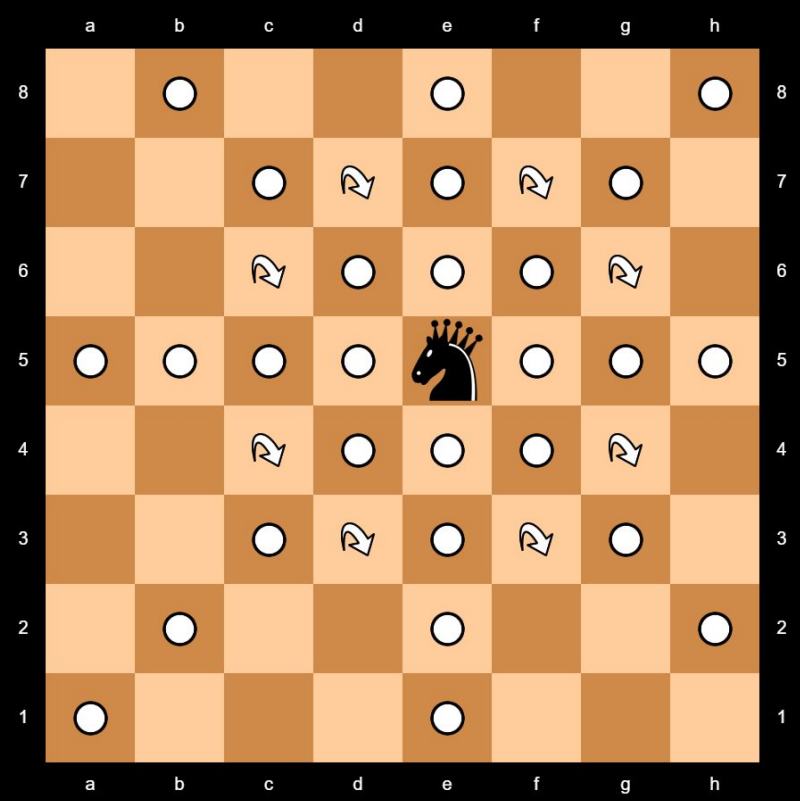
Rook & Bishop





Queen (Rook + Bishop) & Amazon (Queen + Knight)





TODO 5A - Queen.place

Queen class EXTENDS rook and bishop and is both

- a piece by inheritance via super
- a composition of two helper objects

Therefore Queen.place is three fold (and also three lines of code)

- queen inherited position (super place)
- rook helper position (r place)
- bishop helper position (b place)

TODO 5B - Queen.menaces

Since you already implemented rook and bishop

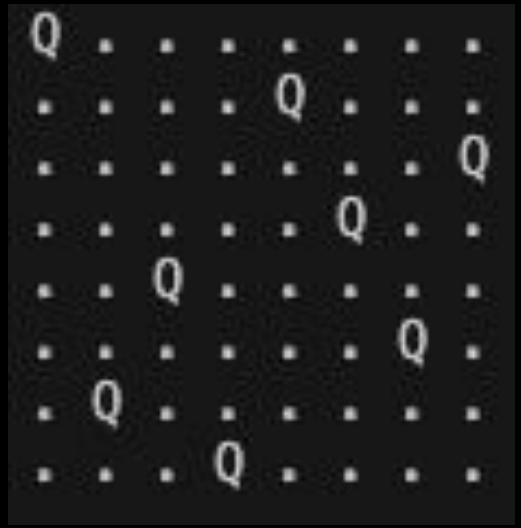
- Given: piece p

- Return: true if rook or bishop menaces p

Now implementing amazon should be fairly intuitive

TODO 7 - Print Board

- Given: n, coords, and glyph
- EXAMPLE: 8, [(0,0), (1,4), (2,7), (3,5), (4,2), (5,6), (6,1), (7,3)], Q
- Result:



TODO 7 - Print Board

- Make n x n grid
- Fill every cell with ‘.’ via Arrays.fill();

- Place pieces from given coords
- Treat coords as (r,c) with i row i+1 col
  - for (int i = 0; i < coords.length; i += 2)
    - int r = coords[i], c = coords[i+1]
    - if (in bounds)
      - set grid[r][c] to glyph

TODO 7 - Print Board

Spacing

- stringbuilder()
    - row
        - col
            - stringbuilder append grid coords
            - if (c + 1 < n) stringbuilder append space
        - sb.append('\n');
- return sb.toString();

TODO 8 - Print Coords

```
[(0,0), (1,4), (2,7), (3,5), (4,2), (5,6), (6,1), (7,3)]
```

0,0,1,4,2,7,3,5….

[for each pair:

- if (i > 0) then + “, ”
- ‘(‘ + coord i + ‘,’ coord i+1 + ‘)’

]

return

Recall

- GOAL: Place **N** identical pieces on an **N** x **N** board so none attack each other.

- FOR EXAMPLE: There are **x** ways to place **4** queens on an **4** x **4** board so none attack each other.

- We want to find **x** given …

Finally backtrack

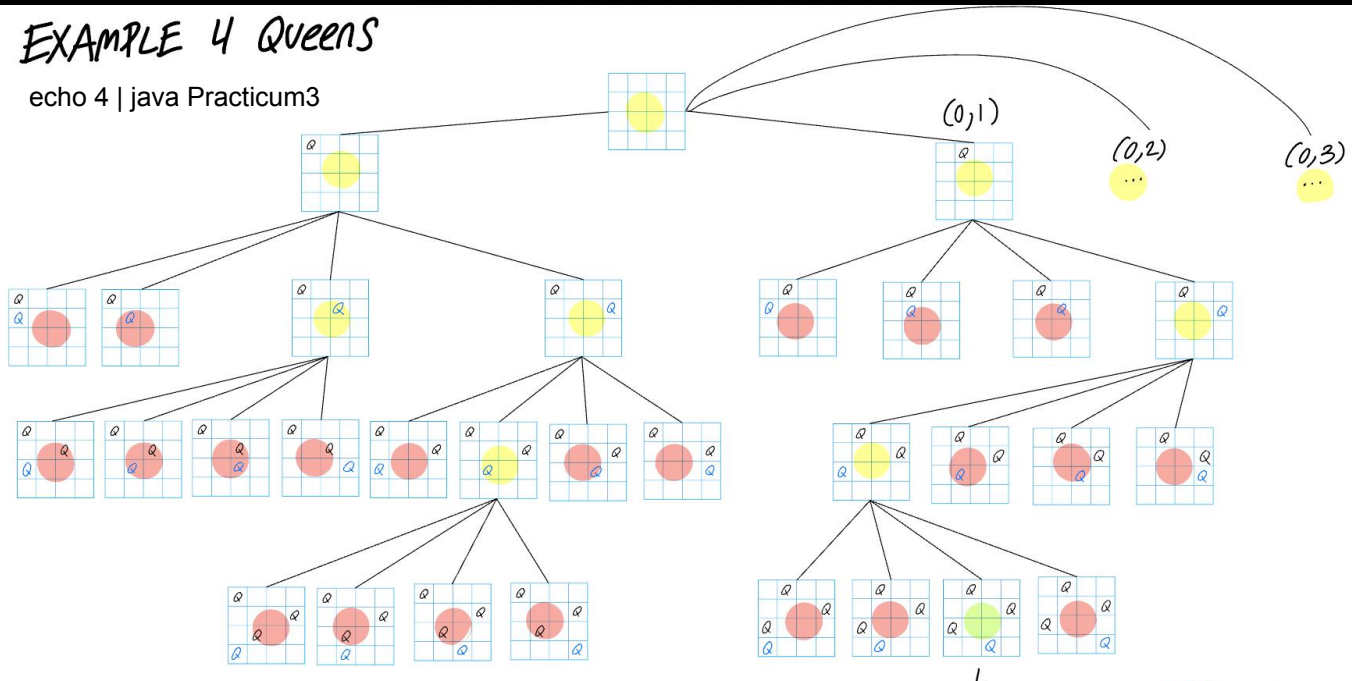
We want to find **x** given…

- startCell (Recurse with startCell + 1)
- currentIndex (Pieces already placed)
- Bucket[] (Stores valid board(s))

- Need (How many solutions we need before stopping early)
  - Recall: This depends on [mode] = mode (count|one|list)
    - Example: echo 8 | java Practicum3 rook list 3
    - Then need = 3

EXAMPLE 4 Queens

echo 4 | java Practicum3



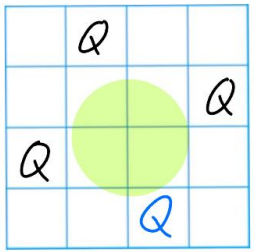
= MENACE → SKIP

= Possible → backtrack cell+1

= Valid board → buck

Save to bucket

SOLUTION #1



All N pieces placed = Valid

Expected backtracking results

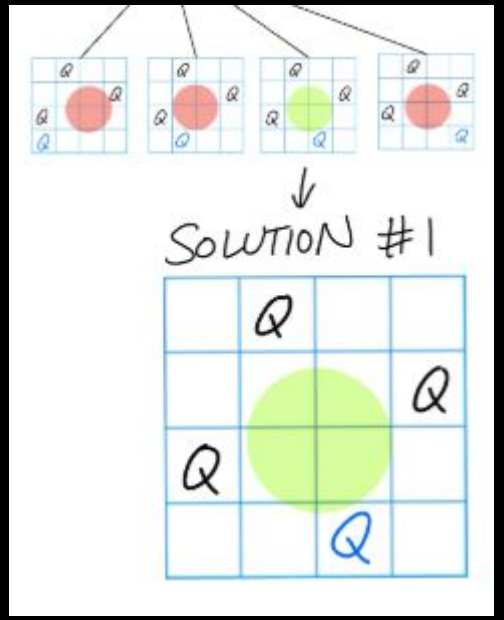
echo 4 | java Practicum3 queen list 2

```
Solution 1:
. Q . .
. . . Q
Q . . .
. . Q .

Coordinates: [(0,1), (1,3), (2,0), (3,2)]

Solution 2:
. . Q .
Q . . .
. . . Q
. Q . .

Coordinates: [(0,2), (1,0), (2,3), (3,1)]
```



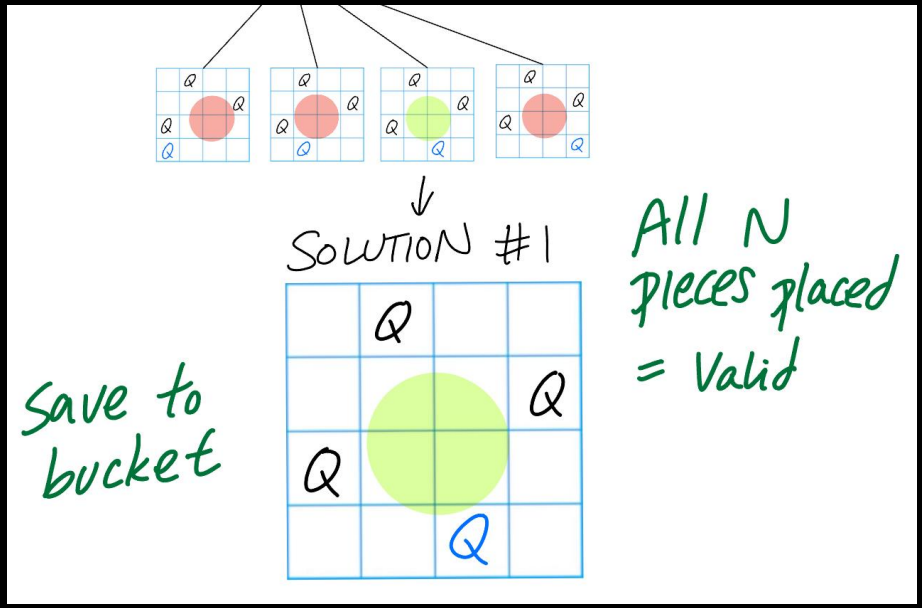
Backtrack structure

Base cases:

- if stopEarly, return

[GREEN] if valid board completed

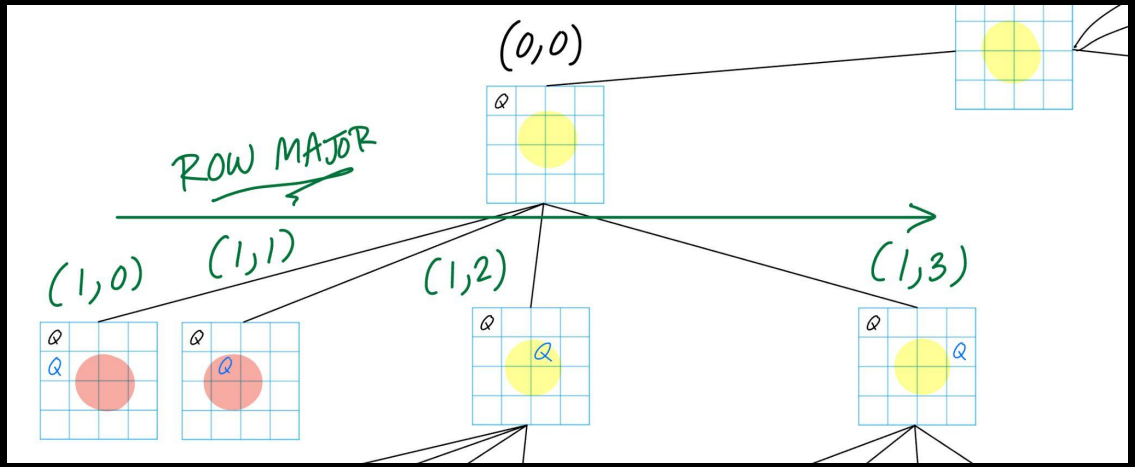
- Count++
- If bucket not empty
  - Save coords via bucket.add(**snapshot()**);
  - If bucket size >= need -> stopEarly = true
- Return



Backtrack structure

Next cells:

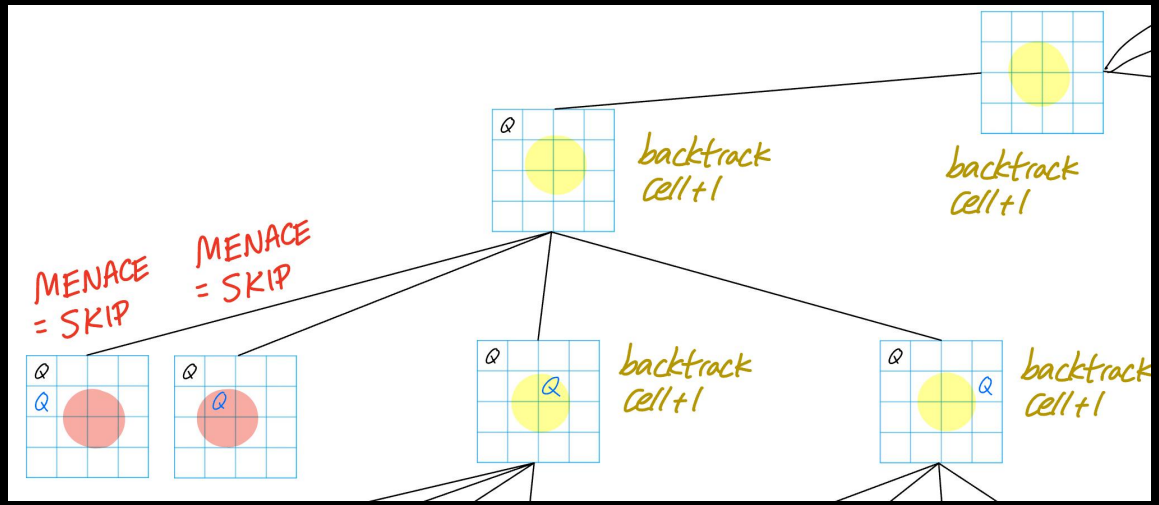
- Iterate from start cell to end of cell (as long as !stopEarly)
  - map index via **ROW MAJOR** where row = cell / n and col = cell % n
  - Place curr index at r, c

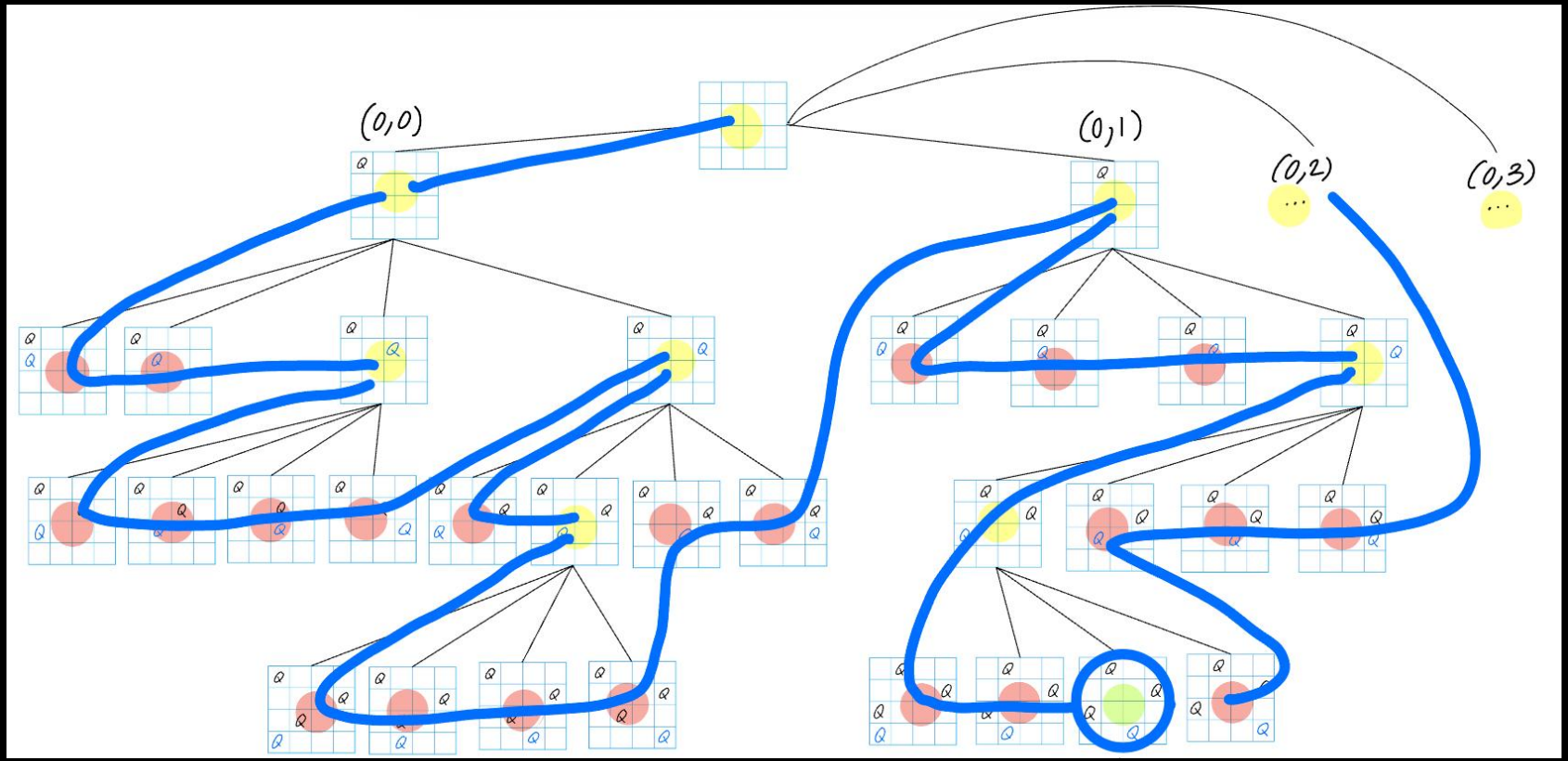


Backtrack structure

Next cells (cont):

- Consider invalid: compare piece we just placed with pieces already placed
  - **[RED]** If any piece **menaces** current piece then skip cell
- **[YELLOW]** Recursive call: **backtrack(cell + 1, currentIndex + 1, bucket, need)**




(0,0)
(0,1)
(0,2)
...
(0,3)
...

Grading

- Practicum attendance in lieu of report (20 points)
- Upload completed Practicum3.java on gradescope (80 points)

- As usual, the gradescope score is mainly for feedback, not your final score.
  - Test your code on gradescope for as many times as you want to before the due date.
  - Make changes according to the autograder results.

How I’m grading this

- (1) If the autograder shows 80/80 then you will receive full credit.
- (2) If there are some cases that fail
  - … and if there's no work shown the grade will stay as is.
  - … and if there's work shown I'll give half credit for that case.
- (3) If your program exceeds runtime (likely from backtracking) and no score is shown on gradescope
  - I will find the section that causes it then replace it with the solution
  - You will get half credit for the cases related to that section, then the rest will be graded the same way as (2).

Getting help

Again, if you’re having trouble with the practicum at any point before the due date I’m more than happy to help.

- Office Hours: Monday Nov 10, 3 - 4 PM @ Heafey 135
- Email: dchui2@scu.edu