Problem Solving Session

- The remainder of today's class will comprise the ***problem solving session*** (***PSS***).
- Your instructor will divide you into ***teams of 3 or 4 students***.
- Each team will ***work together*** to solve the following problems over the course of ***20-30 minutes***.
  - You may work on paper, a white board, or digitally as determined by your instructor.
  - You will submit your solution by pushing it to GitHub before the end of class.
- Your instructor will go over the solution before the end of class.
- If there is any time remaining, you will begin work on your homework assignment.

Class participation is a significant part of your grade (20%). This includes in class activities and the problem solving session.

Your Course Assistants will grade your participation by verifying that you pushed your solutions before the end of the class period each day.

Problem Solving Team Members

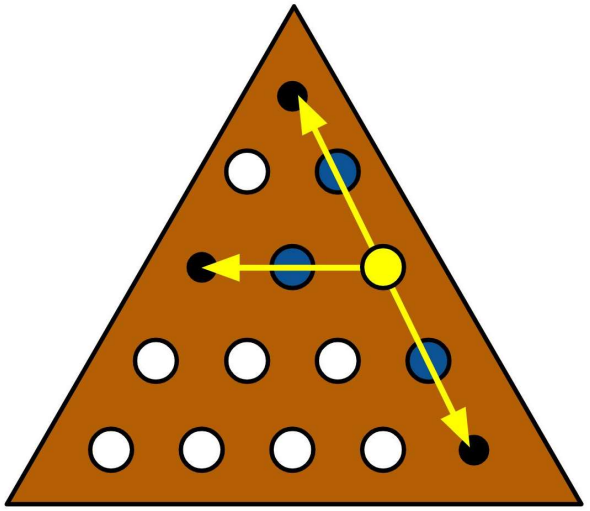
Record the name of each of your problem solving team members here.

Do not forget to ***add every team member's name***! Your instructor (or course assistant) may or may not use this to determine whether or not you participated in the problem solving session.

Nicholas Milonni
Michael Scalzetti
Vishwesh Venkatramani

The Peg Game Part 3

- This is the ***third*** part of a ***three*** part project.
  - This part is due on ***Monday April 5th, 2021***.
- In this part of the project, you will primarily be focused on creating a ***triangular peg game*** that will be played on board in the shape of an isosceles triangle.
  - Moves can be made ***up***, ***down***, or ***horizontally***.
  - Vertical moves are made at an angle.
- This means that, if you carefully implemented parts 1 and 2 of the project using only the `PegGame` interface, you should be able to play a triangular game without modifying your ***command line interface*** or your ***backtracking configuration***.



A triangular board includes a specific number of ***rows***. Each row has one additional column than the previous row. The bottom row has columns equal to the height of the board.

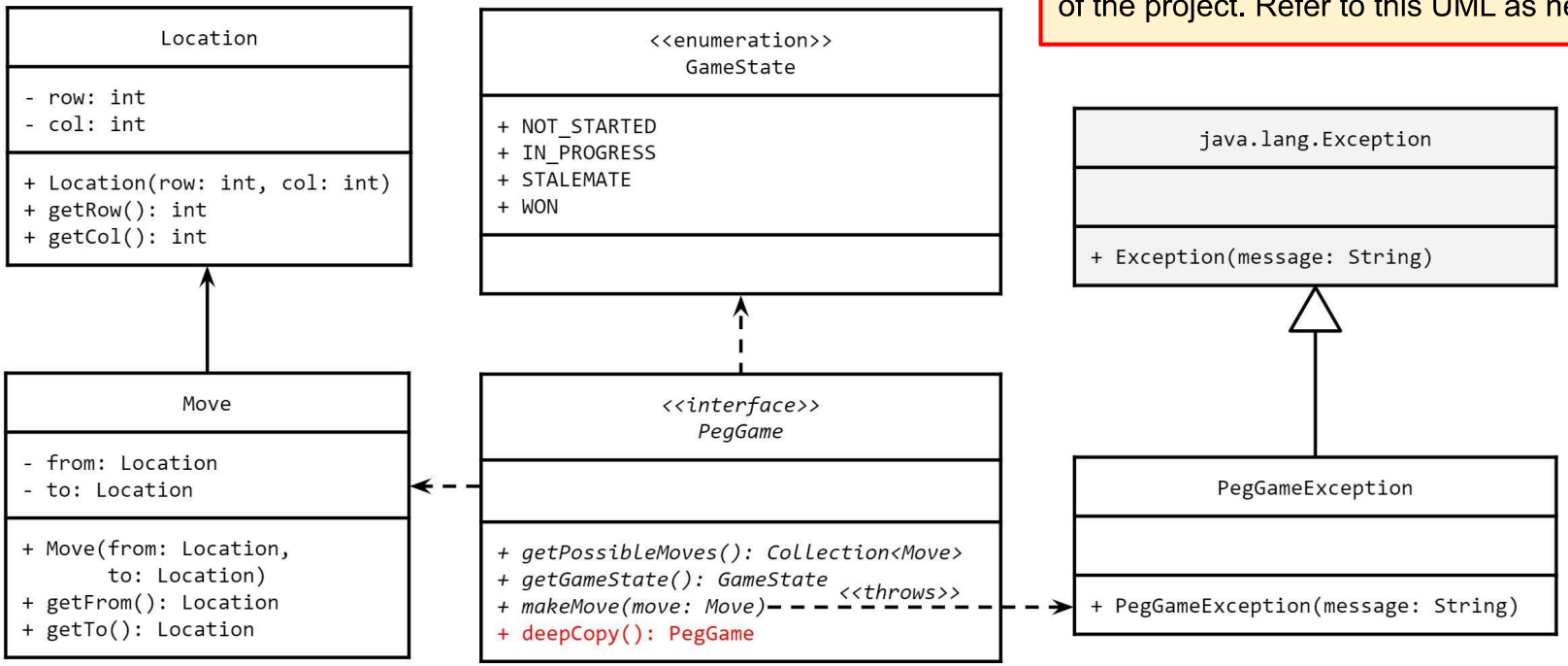
- ***Pair programming*** is a technique during which two developers collaborate to solve a software problem by writing code together.
- One developer takes on the role of ***the driver***.
  - Shares their screen.
  - Is actively writing code.
- The other developer(s) takes on the role of ***the navigator***.
  - Watches while the driver codes.
  - Takes notes.
  - Asks questions.
  - Points out potential errors.
  - Makes suggestions for improvements.
- The driver and navigator regularly ***switch roles***, e.g. every ***10-20 minutes***.
  - Set a timer!
  - ***Push your code!***
- For the rest of today's problem solving session, you and your team will practice pair programming with ***one*** team member acting as the driver and the ***remaining*** team members acting as the navigators.

Pair Programming

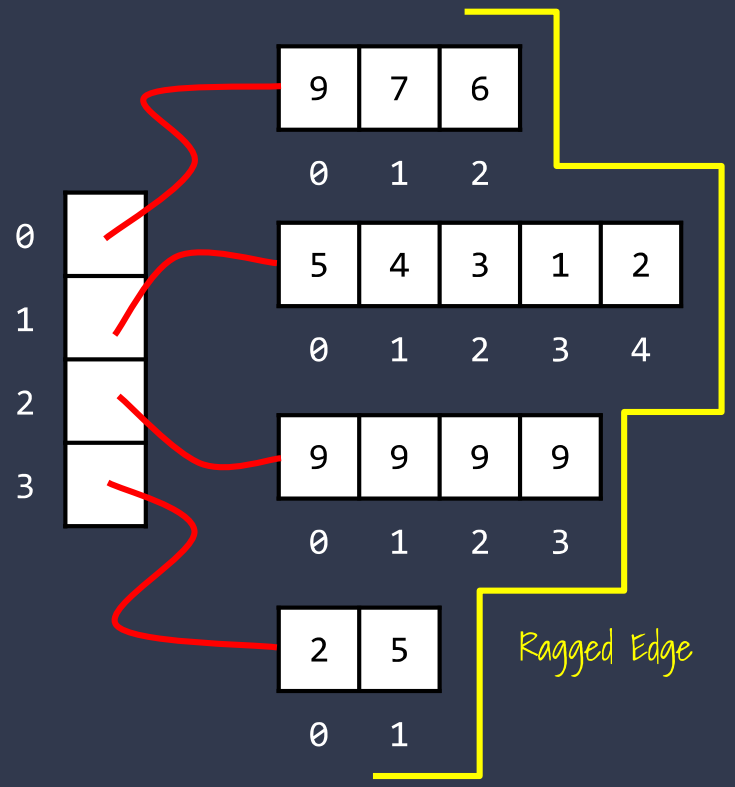
When you are the driver, you should be using Zoom or Discord to ***share your screen***. Be sure to ***push your code*** and ***switch roles*** every 10-20 minutes!

A Partial Design

Your base design should not change in this part of the project. Refer to this UML as needed.



Ragged Arrays



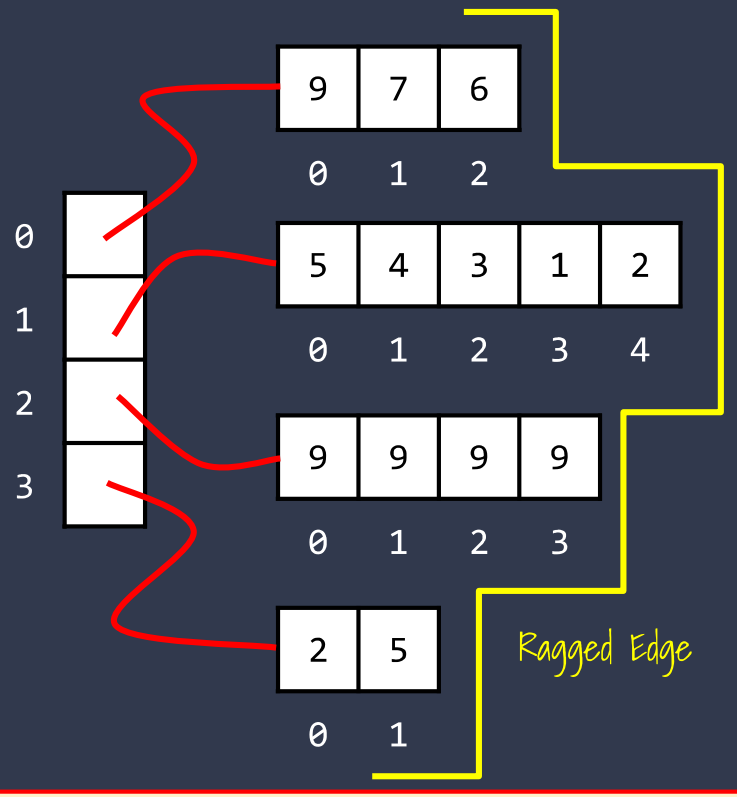
Each index in the first dimension of a 2D array is a reference to another array.

- You should recall that a ***two dimensional array*** is really an ***array of arrays***.
  - Each element in the ***first dimension*** is a reference to ***another array***.
- While 2D arrays are convenient for creating tables, where every ***row*** has the same number of ***columns***, this is not required!
  - Each row may have a variable number of columns.
- When stacked on top of each other, the rows create a ***ragged*** edge.
- Such an array can be created by specifying only the size of the ***first dimension***.
  - `int[][] x = new int[5][];`
- Each row can then be initialized separately.
  - `x[0] = new int[3];`
  - `x[2] = new int[4];`
- Care must be taken when accessing individual elements because the ***length*** of each row is potentially different.
  - `int rowLength = x[2].length;`

```
// write the code to declare a 2D array
// matching the one pictured to the right.
// there are any number of ways to do this.
int[][] arr = new int[4][];

 Arr[0] = {9,7,6};//new int[3];
Arr[1] = {5,4,3,1,2};//new int[5]
Arr[2] = {9,9,9,9};//new int[4];
Arr[3] = {2,5};//new int[2];
```

Practice



You are ***not*** required to use a 2D array (or a ragged array) in your implementation, but a little practice never hurt.

Locations & Moves

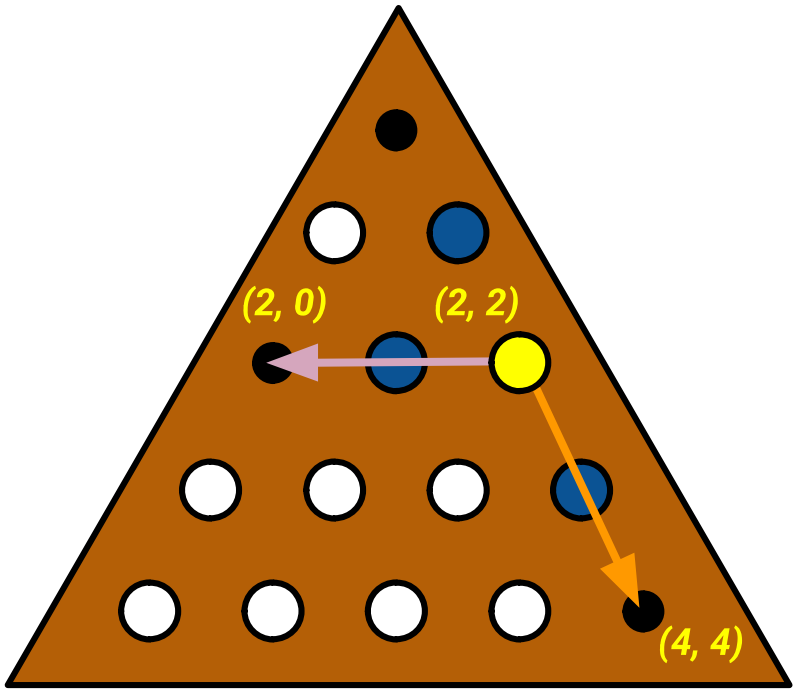
We will continue to use a scheme that is similar to a rectangular board when addressing locations on a triangular board.

As with a rectangular board, rows are numbered from ***top-to-bottom*** beginning with ***0***.

Columns are numbered ***left-to-right*** with the first space in each row being in column ***0***.

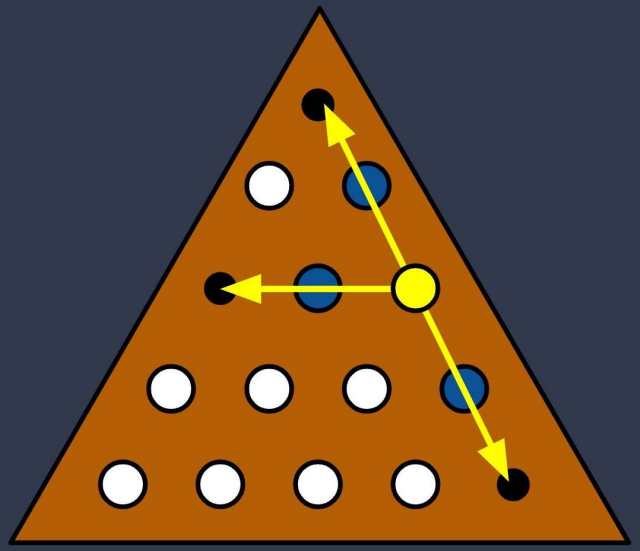
Therefore the yellow peg in the diagram to the right is at location ***(2, 2)***.

The move shown in ***orange*** is from location ***(2, 2)*** to location ***(4, 4)***.



Thinking Ahead

Consider your existing implementation of a rectangular board. What changes will (or should) you make to accommodate the new board shape?



What are some of the strengths of your current design?

Simple, which makes it easy to understand and to debug. It's easy to check if things are out of bounds.

What do you think could be improved in this new implementation?

It doesn't work with any shape, which isn't epic 😔

What parts of your current implementation might you be able to reuse?

Deep copy, isValid with slight change, isGoal with slight change, get successor with small changes, basically ever

What data structure(s) will you use to represent a triangular board?

We will use a Ragged Array to represent the triangular board.

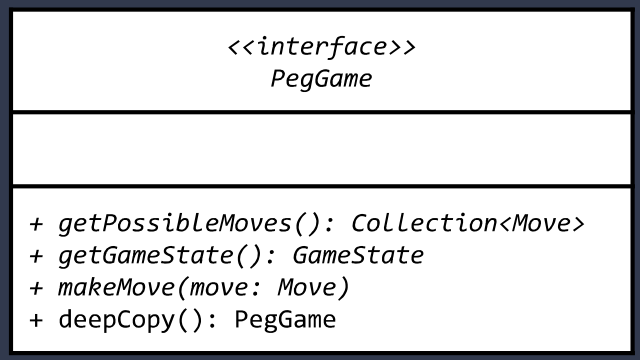
What changes will be needed in your existing command line and backtracking configuration to work with a new kind of game?

We need to change the logic for checking if pegs are in bounds.

TrangleBoard
Boolean[][] board; GameState state;
+getPossibleMoves(): Collection<Move> +getGameState(): GameState +makeMove(move: Move) +deepCopy(): PegGame +get(loc: Location): boolean +set(loc: Location, value: boolean):

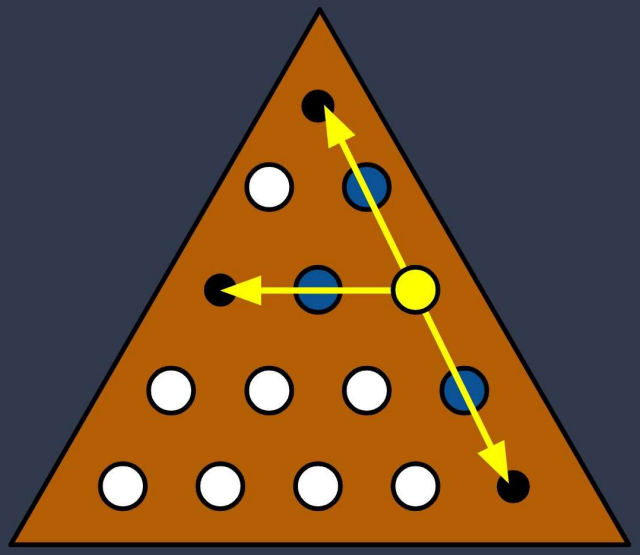
UP Front Design

You will need to create an entirely new class to implement a triangular peg game board. This new class will in the very least implement the `PegGame` interface.



Use the table to the left to create a UML class diagram for your new class. Try to include as much detail as you can. Feel free to use your existing rectangular peg game class for inspiration.

Initial Implementation



At this point you will only need fields and a constructor to continue with the remaining activities.

- Work with your team to begin implementing your ***triangular peg game*** class.
  - Begin be creating the class and implementing the `PegGame` interface.
  - Stub out all of the methods for now.
  - Declare any fields and a constructor.
- Keep in mind that you will need to read board configurations from files as you have done previously.
  - What fields will you need to pass into the constructor?
- Again, use your rectangular board game implementation for inspiration as needed.
  - What changes or improvements could or should you make?

- In part 1 of the project you wrote a class that could be used to create a rectangular peg game by reading a board configuration from a text file.
- Now you will write a similar class that can create a triangular peg game using a similar file format.
- This class should read the board configuration from a file like that depicted to the right.
    - The first line in the file will specify the number of rows.
    - The point of the triangle will always be at the top.
    - The characters in the file form a right triangle shape, but the triangle displayed to players will be an isosceles triangle.
- At least one major difference for this new class is that you will create an instance of your new triangular peg game.
- Begin implementing your class now.

Reading Boards

The same file format will be used for triangular boards, but the number of columns per row will not be the same on every line.

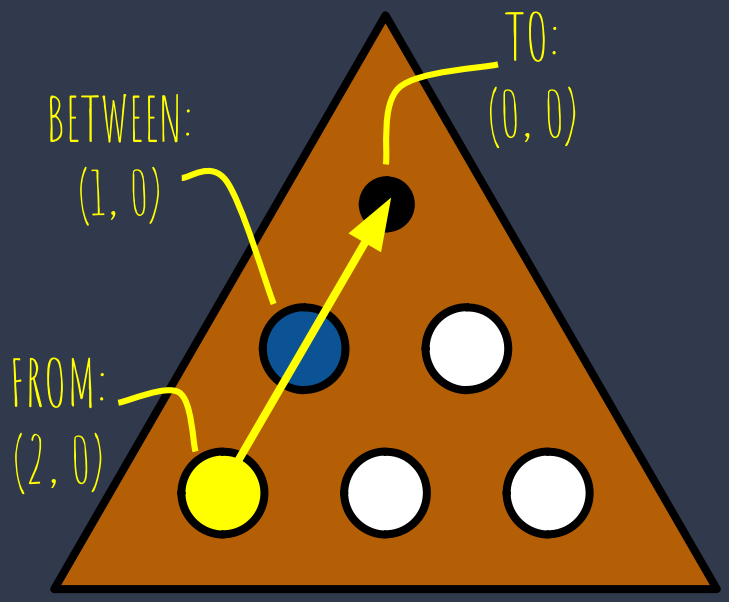
As before, the first line will specify the number of rows on the board. A `'.'` indicates an empty space and a `'o'` indicates a peg.

```
5
.
oo
ooo
oooo
ooooo
```

```
    .
   o o
  o o o
 o o o o
o o o o o
```

The left example is a text file containing a board layout. The right is an example of how the board would be displayed as an isosceles triangle.

Man in the Middle



A move in the Peg Game ***requires*** that one peg "jump" over another to remove it from the board.

- The Move class comprises two locations: the location ***from*** which a peg is being moved, and the location ***to*** which it is moving.
- When a valid move is executed, the peg ***in between*** those two locations is removed from the board.
  - However this third location is ***not*** part of the move class (nor should it be).
- Not all Peg Games are played on rectangular boards, so "in between" means different things for different shapes.
  - Assuming that a move is being made on a ***triangular board***, how will you determine the location that is between the two end points of the move?
- Work with your team to try and come up with a strategy for determining the location of the "in between" peg given ***only*** the to/from locations.
  - A move may be horizontal or vertical.
  - Try experimenting with jumps to and from different locations on a board. Can you find a ***pattern*** for the "in between" location relative to the to/from locations?

Date/Time	Location	Area of Focus
4/03@8:00PM	Discord	Make a new board, shaped like a triangle
4/04@8:00PM	Discord	Debugging, and other small changes

Meeting Times

This part of the project is due ***Monday April 5th, 2021*** at the start of class.

You should plan to ***work together*** with your team as much as you can, even if that means setting up a remote meeting using Zoom or Discord.

Use the table to the left to plan ***at least 1 out of class meeting*** over the course of this part of the project. The meeting should be at least ***1 hour***.

An example has been provided. You should delete it.

If You Made it This Far...

Your team is off to a good start, but you are ***not quite*** finished with ***Part 3*** of the Project yet. Remember that this part of the project is due on ***Monday April 5th, 2021***.

You still need to fully implement the ***triangular peg game*** and verify that it works with your existing ***command line interface*** and ***backtracking configuration***.

If you have time remaining in class, you should begin reading the full project description, which you will find on MyCourses.