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

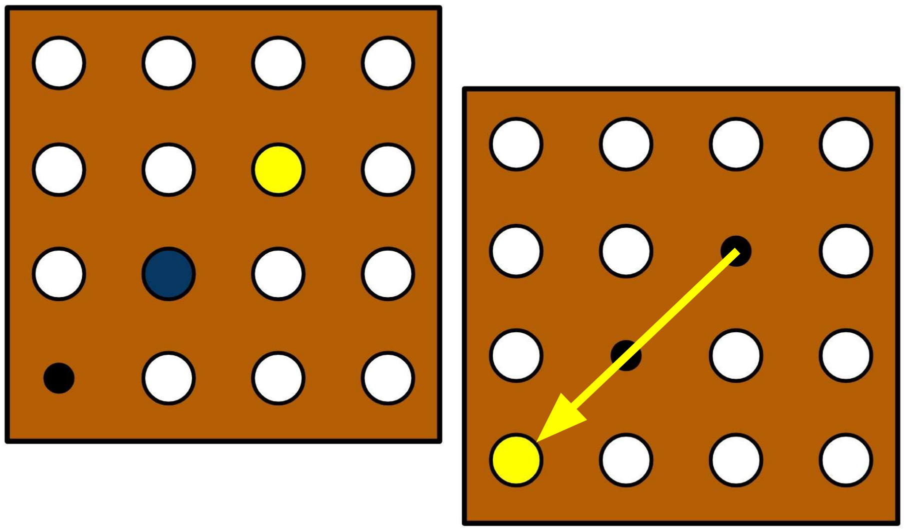
Record the name of each of your problem solving team members here.

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.

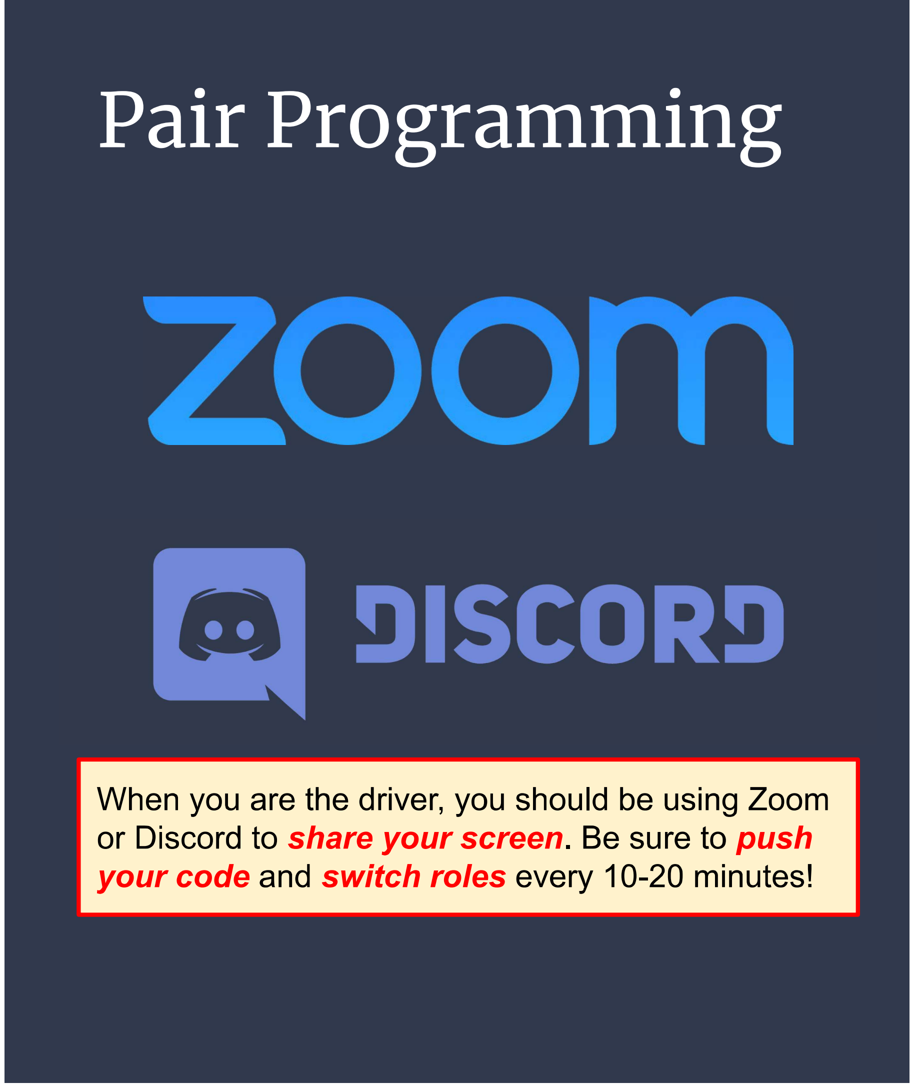
Nicholas Milonni
Michael Scalzetti
Vishwesh Venkatramani

The Peg Game Part 2

- This is the ***second*** part of a ***three*** part project.
  - This part is due on ***Monday March 29th, 2021***.
- In this part of the project, you will primarily be focused on creating a ***backtracking configuration*** that will attempt to find a series of moves that will win a Peg Game.
  - ***Zero or more*** moves may have already been made.
  - A winning solution ***may not exist***.
  - If a solution does exist, you will need to provide the ***list of winning moves***.
- Your configuration will be used to implement two new features in your command line interface.
  - An improved ***hint*** command that will only suggest moves that are part of a winning solution.
  - A ***solve*** command that will automatically solve a Peg Game that is in progress (if possible).



Rather than suggest ***any*** possible move, your improved help function will ***only*** suggest moves that are part of a winning solution to the current game.

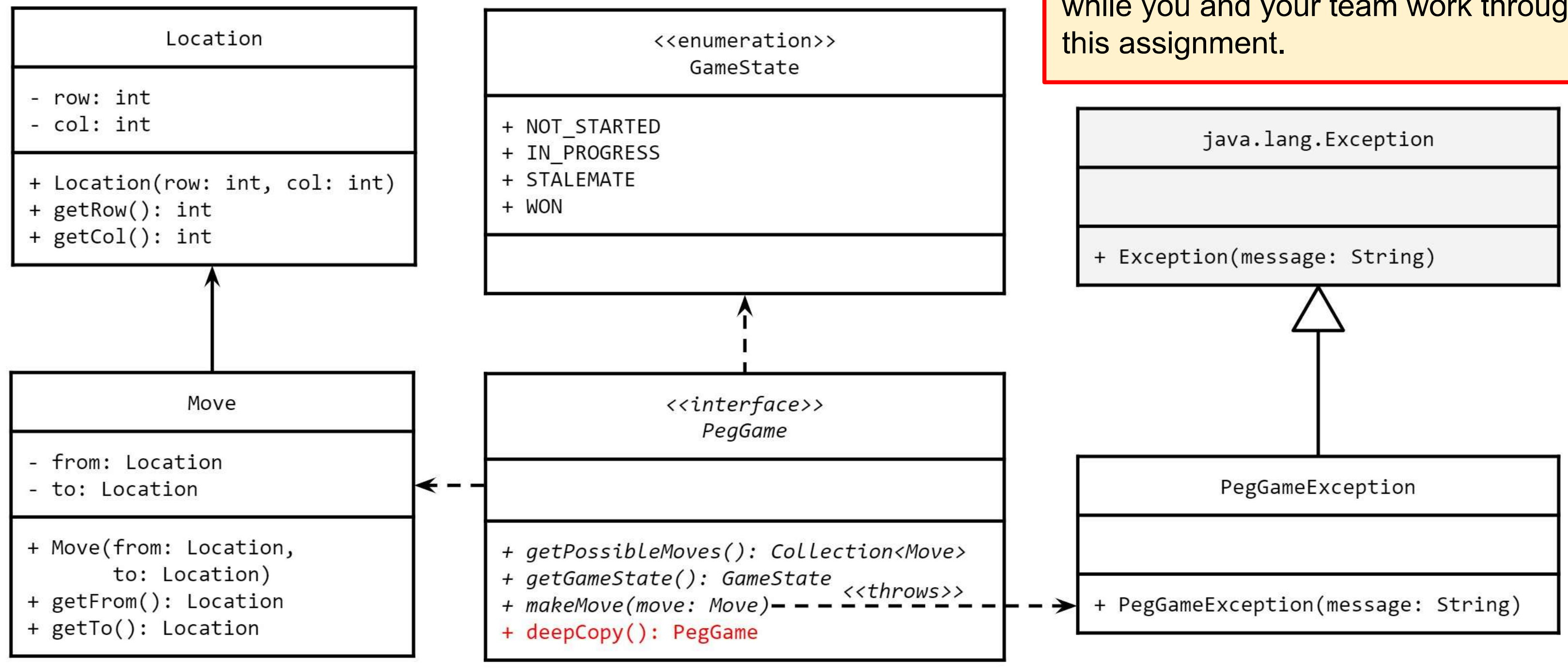


Pair Programming

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!

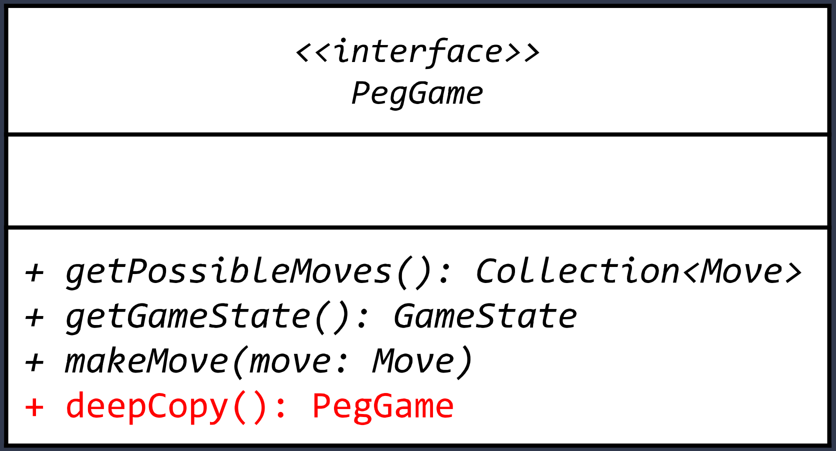
- ***Pair programming*** is a technique during which two developers collaborate to solve a software problem by writing code together.
- One developer takes on the role of ***the driver***.
    - Shares their screen.
    - Is actively writing code.
- The other developer(s) takes on the role of ***the 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.

A Partial Design



Refer to this full UML class diagram as needed while you and your team work through the rest of this assignment.

Deep Copy



```
<<interface>>
PegGame

+ getPossibleMoves(): Collection<Move>
+ getGameState(): GameState
+ makeMove(move: Move)
+ deepCopy(): PegGame
```

Default methods are ideal for adding ***new*** methods to ***existing*** interfaces that have already been implemented.

- Your ***backtracking configuration*** will almost certainly need to create a ***deep copies*** of your `PegGame` implementation as it creates successors.
- In order to guarantee that ***any*** current or future implementation of `PegGame` can create a deep copy of itself, we will need to add a method to the interface.
  - This will guarantee that the method exists in any class that implements the interface!
  - In order to avoid breaking existing code, you will add a ***default implementation*** that should throw an `UnsupportedOperationException` .
- Next, you and your team should begin working on implementing the `deepCopy()` method in your rectangular `PegGame` implementation.
  - When called on an instance of the class, it should return a ***deep copy of itself***.
  - You may need to consider creating an additional constructor!
  - Don't forget that you will need to make deep copies of ***data structures*** like lists and arrays!

Backtracking

In this part of the project you will be creating a ***backtracking configuration*** that will attempt to solve a Peg Game. You will also need to provide the list of moves needed to win.

Examine your code and think about how you will implement your configuration and answer the questions to the left. Be as detailed as possible.

Remember:

- A ***configuration*** is at least a partial attempt at a solution.
- A ***successor*** is a new configuration that includes one additional choice.
- A configuration is ***invalid*** if it is impossible to find a solution from this point.
- A configuration is the ***goal*** if it is a valid solution to the problem.

What state will your ***configuration*** need to keep track of as it attempts to find a solution?

`Board, location (to, from), possible moves`

How will you make ***successor*** configurations?

We already have a getpossible moves that returns a set so that can be used to make a new board then we can do pruning.

How will you determine if a configuration is ***invalid***?

Check if there is atleast one move left to make or if we have the goal, if empty vals >= 2 then we have a stalemate.

How will you determine if the configuration is ***the goal***?

We will need to get successor for each possible move, prune the illegal moves, then if we have anything remaining that will be the goal and only one peg remains

A Configuration

You may want to consider adding a main method to your configuration to manually test it. Use the following example to guide you.

```java
Configuration config = new MyConfig(game);
Backtracker backtracker = new Backtracker(true);
Configuration sol = backtracker.solve(config);
if(sol == null) {
   System.out.println("No solution.");
} else {
   System.out.println(sol);
}
```

You will of course need to create an instance of your Peg Game implementation class to use as well.

- If you have not already done so, download the provided files from MyCourses and add them to your project.
  - Make sure to push them to the repository so that all team members have access.
  - The `Backtracker` and `Configuration` classes are among the files that have been provided.
  - You have also been provided with some output examples that you can use to guide the implementation of the new features in the ***command-line interface***.
- Using the information that your team brainstormed in the previous activity, begin implementing a ***backtracking configuration*** to solve ***any*** Peg Game.
  - ***Do not*** change the provided classes.
  - The configuration should ***not*** use any functionality that is specific to your rectangular implementation.
  - Consider overriding the `toString()` method in your configuration for use when debug mode is enabled in the backtracker.

Date/Time	Location	Area of Focus
3/20@1:00PM	Discord	solve
3/27@8:00PM	Discord	
3/28@8:00PM	Discord	

Meeting Times

This part of the project is due ***Monday March 29th, 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 2*** of the Project yet. Remember that this part of the project is due on ***Monday March 29th, 2021***.

You still need to implement new features in your ***command-line interface*** including an ***improved hint*** feature and a feature that automatically solves the game after any number of moves have been made (if it can be solved).

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