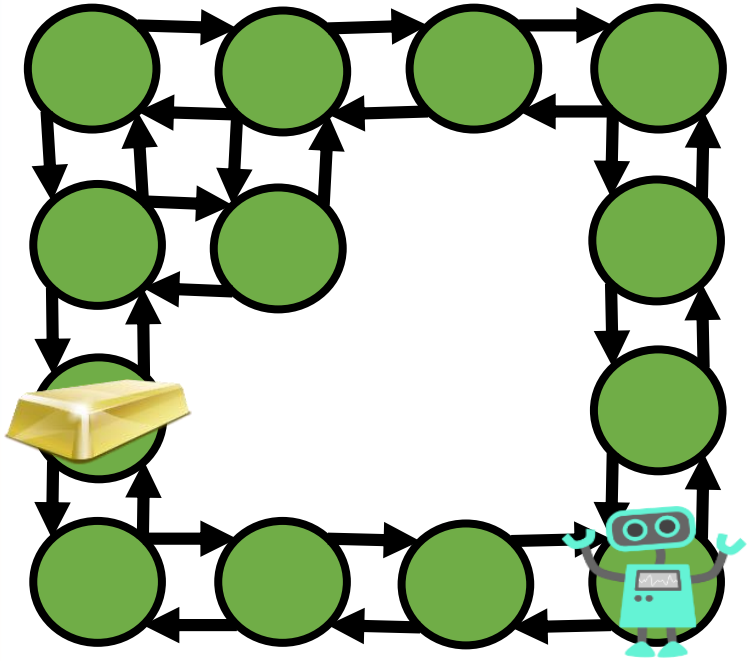
Class Design

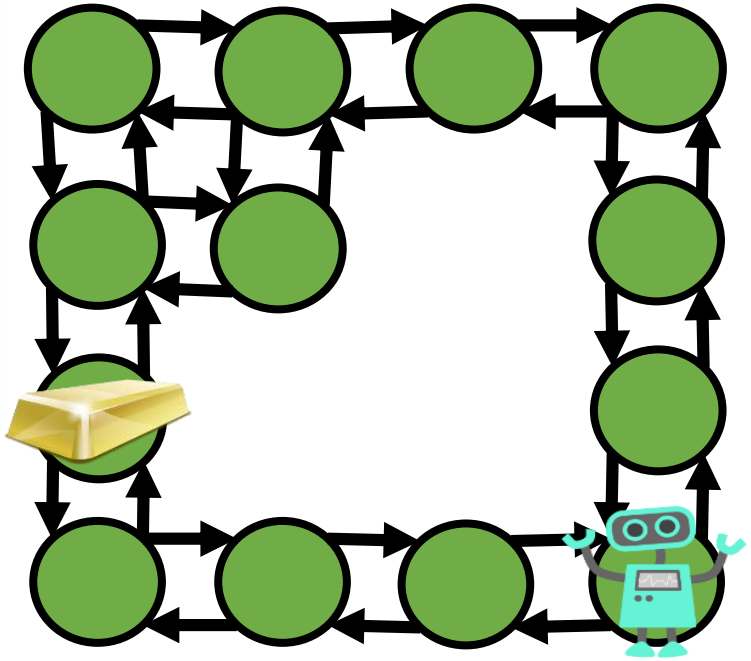
Part 1: From Graphs to Java classes

By the end of this video you will be able to…

- Design classes to represent a grid-based graph



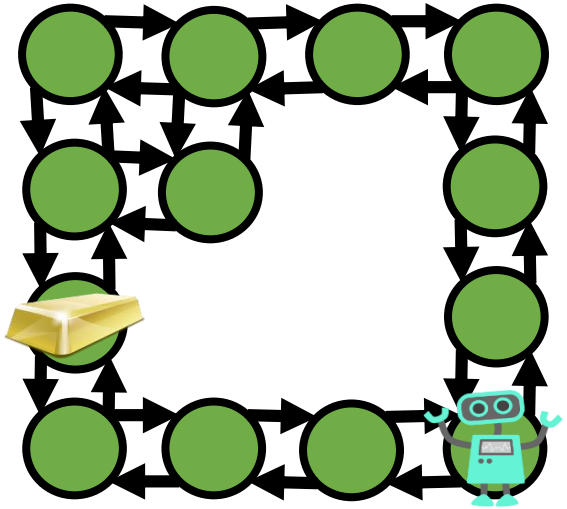
Goal: Design classes to support path finding through a maze



Goal: Design classes to support path finding through a maze

Questions to consider:

- What do I want to do with the graph?
- What is the ratio of edges to nodes? (Adj. list or matrix)?
- How do I need to access to nodes/edges?
- What properties (if any) do nodes and edges need to store?



```
----
--*-
-**-
----
```

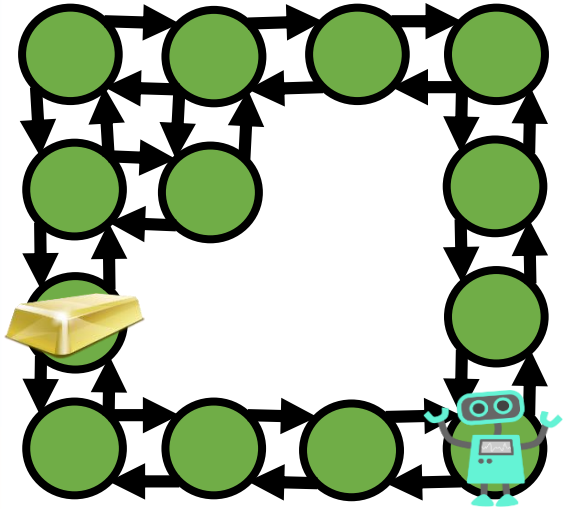
```
oooo
o-*o
G**o
---S
```

DFS

```
----
--*-
G**-
oooS
```

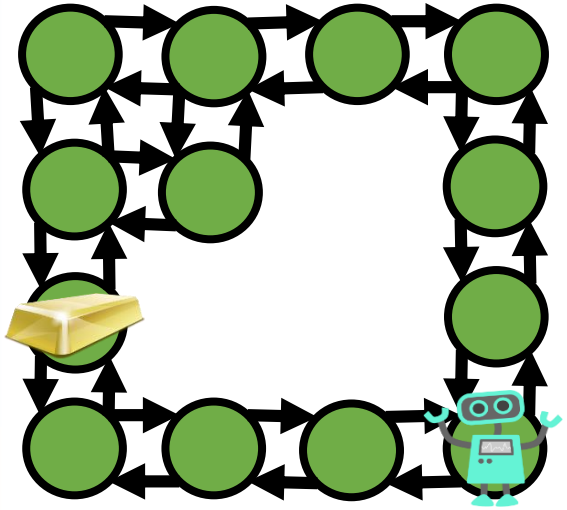
BFS

What do I want to do with the graph?



A class to represent the graph

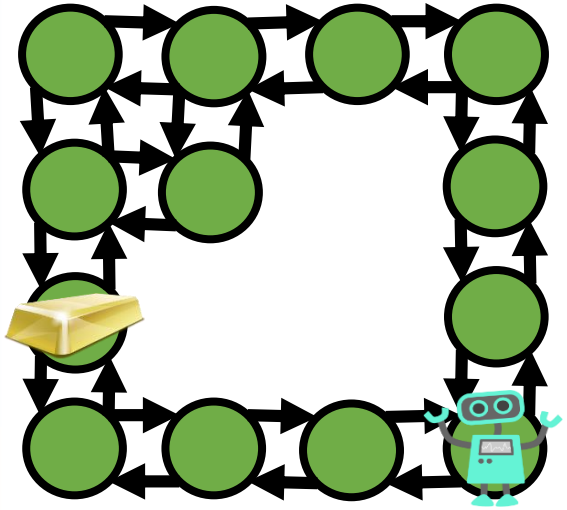
Maze
???
List bfs(start, goal) List dfs(start, goal) printMaze()



Adjacency List or Adjacency Matrix?
Which is better for this graph?

Maze
???

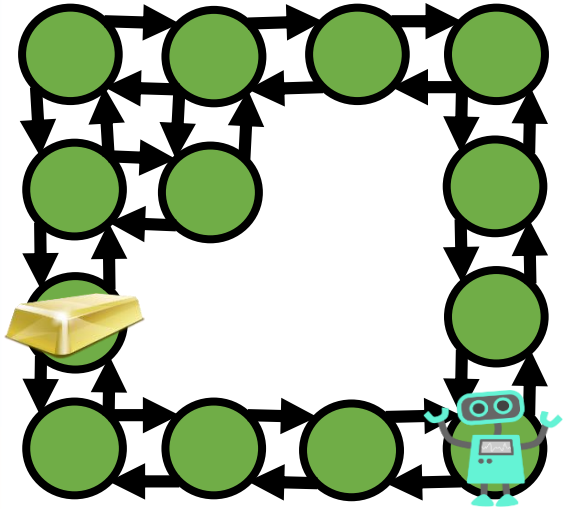
`List bfs(start, goal)`
`List dfs(start, goal)`
`printMaze()`



Adjacency List or Adjacency Matrix?
Which is better for this graph?

Maze
???
List bfs(start, goal) List dfs(start, goal) printMaze()

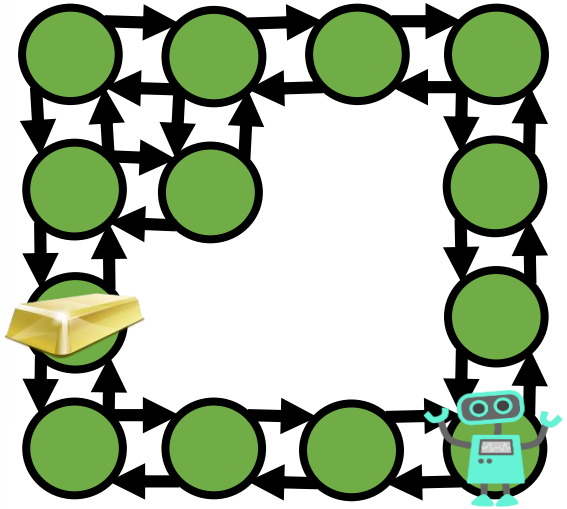
<IVQ HERE>



Do I need a class for Nodes? Edges? (What information will they store?)

Maze
Adj list representation (but how?)

`List bfs(start, goal)`
`List dfs(start, goal)`
`printMaze()`

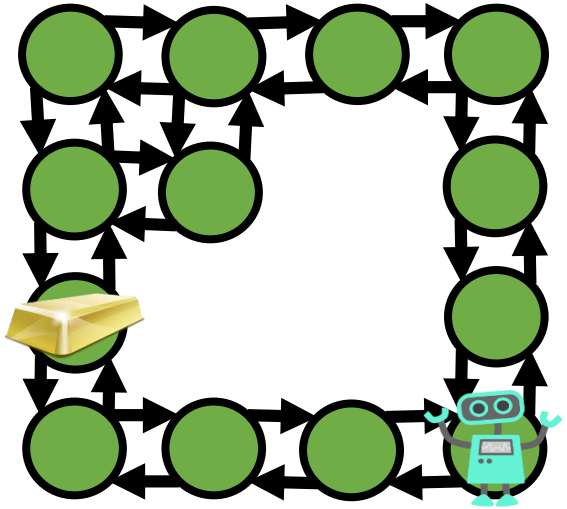


Do I need a class for Nodes? Edges? (What information will they store?)

Maze
Adj list representation (but how?)

`List bfs(start, goal)`
`List dfs(start, goal)`
`printMaze()`

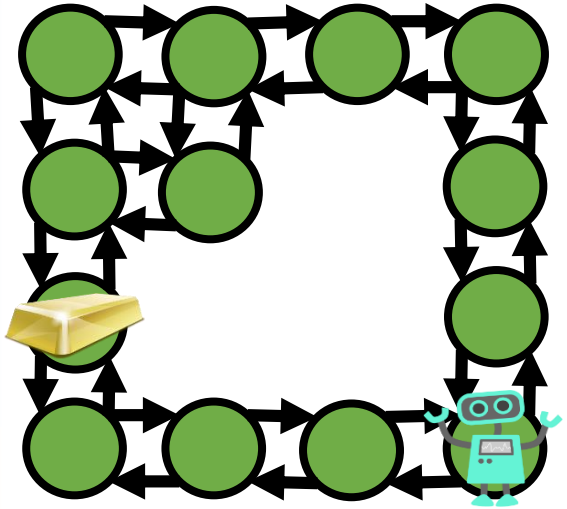
MazeNode
`int row, column` `char dispChar`
`getters and setters`



How do I store the Nodes in the graph?

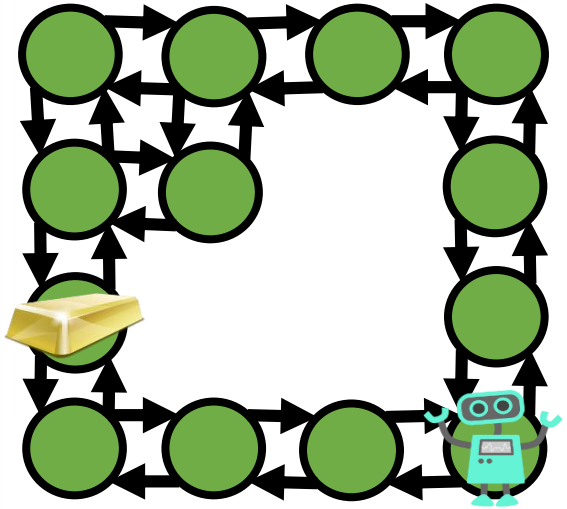
Maze
HashMap<MazeNode, List<MazeNode>> nodes
List bfs(start, goal) List dfs(start, goal) printMaze()

MazeNode
int row, column char dispChar
getters and setters



How do I store the Nodes in the graph?

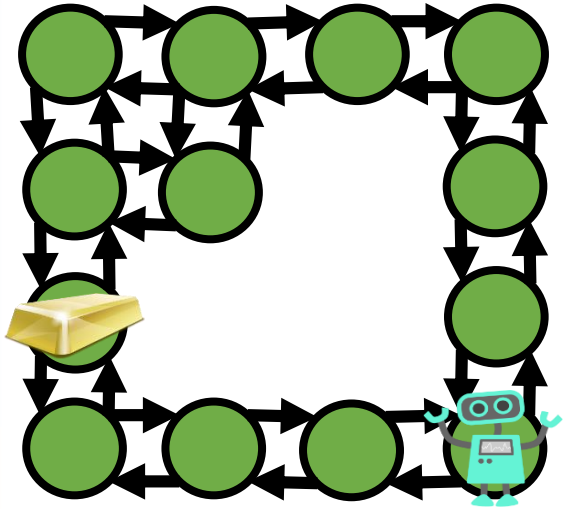
Maze
HashMap<MazeNode, List<MazeNode>> nodes
List bfs(start, goal) List dfs(start, goal) printMaze()



How do I store the Nodes in the graph?

Maze
HashMap<MazeNode, List<MazeNode>> nodes
List bfs(startRow, startCol, goalRow, goalCol)
…

Need quick access to Nodes by (row, col)

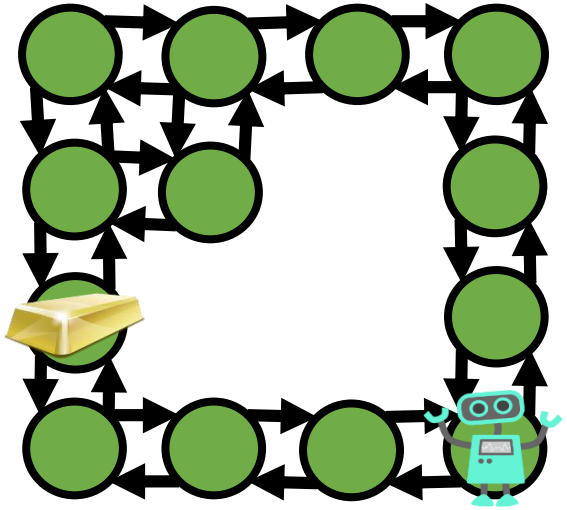


How do I store the Nodes in the graph?

Maze
`MazeNode[][] nodes`

`List bfs(start, goal)`
`List dfs(start, goal)`
`printMaze()`

Make common operations fast (but don't go overboard!)



Where to store the edges?

Maze

```
MazeNode[][] nodes
```

here?

```
List bfs(start, goal)
List dfs(start, goal)
printMaze()
```

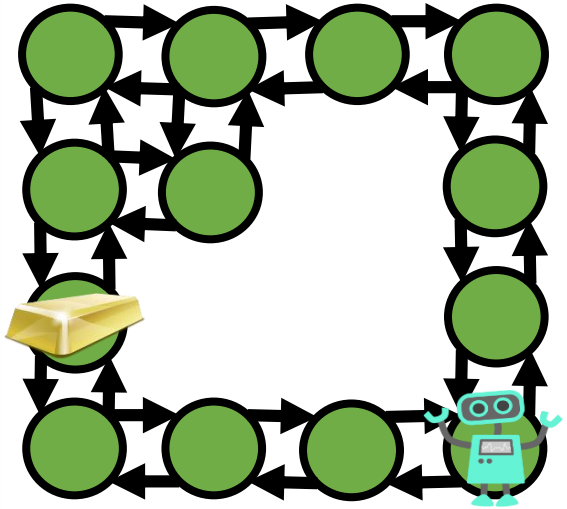
MazeNode

```
int row, column
char dispChar
```

here?

```
getters and
setters
```

It's up to you!



Where to store the edges?

Maze

```
MazeNode[][] nodes
```

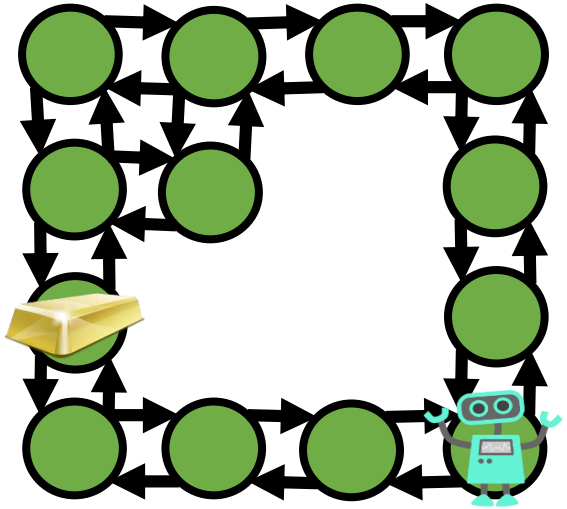
```
List bfs(start, goal)
List dfs(start, goal)
printMaze()
```

MazeNode

```
int row, column
char dispChar
List neighbors
```

here!

```
getters and
  setters
addNeighbor
getNeighbors
```



Check out our code!

Maze

```
MazeNode[][] nodes
```

```
List bfs(start, goal)
List dfs(start, goal)
printMaze()
```

MazeNode

```
int row, column
char dispChar
List neighbors
```

here!

```
getters and
  setters
addNeighbor
getNeighbors
```

What to look for in a good design

- Objects that make sense, whose data and methods go together
- Interfaces are clean; private data (or data structures) are not exposed
- It's easy and fast to do the operations you want to do
- Methods are short and easy to read and understand