Buggy Robot

SER 2016
(Problem also available on Kattis)

For spoilers, see:
https://github.fit.edu/ballard2014/BuggyRobot

Problem Statement

- You Control a robot on a grid
- Goal: reach the goal coordinate
- Some squares are blocked off by obstacles
- You are given a sequence of commands (U,D,L,R) which are not guaranteed to bring your robot to the goal
- What's the minimum number of changes to this sequence which brings the robot to the goal?

Input:

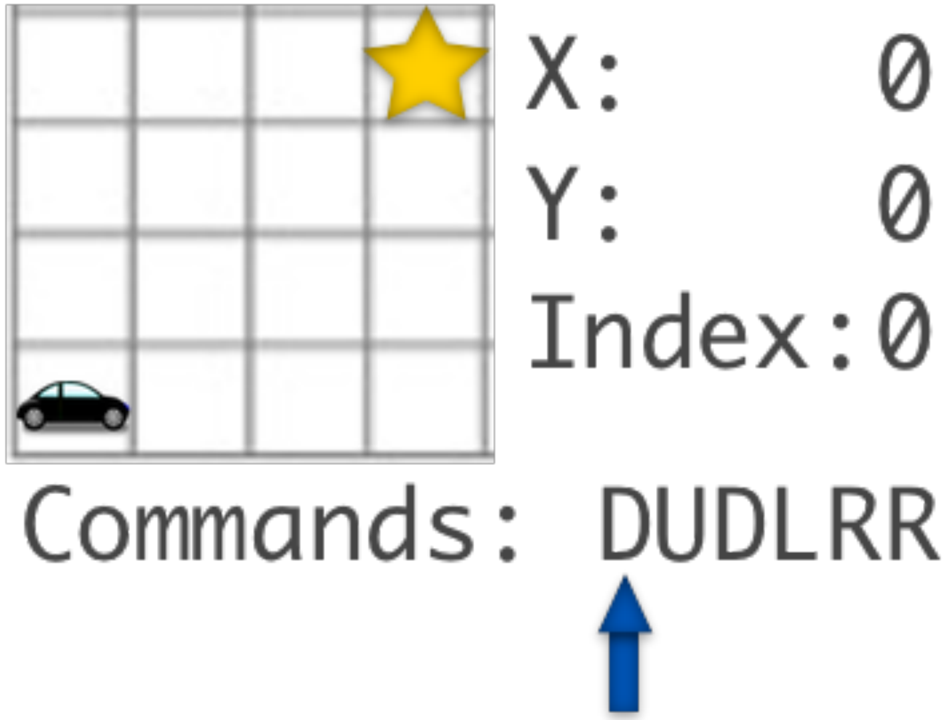
- A rectangular map containing obstacles
- A start coordinate
- A goal coordinate
- A string of commands

- Constraints:
    - Map is <= 50x50
    - Command length <= 50

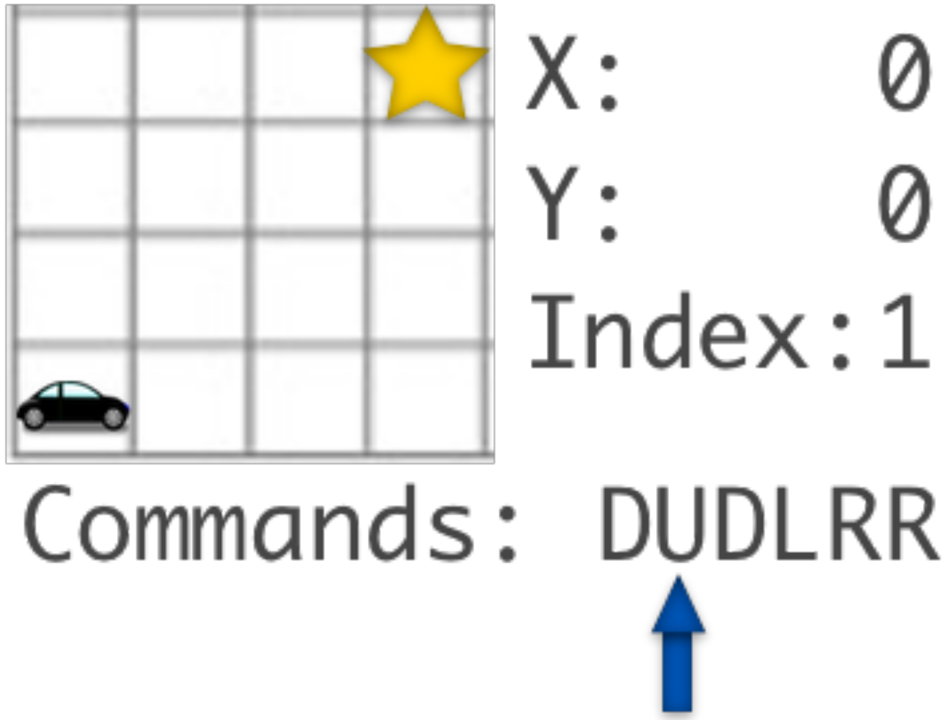
Rules

- A single change consists of:
  - Inserting a command (U,D,L,R)
  - Deleting a command
- The robot will not obey commands asking it to:
  - Run into an obstacle
  - Run off the map’s boundary

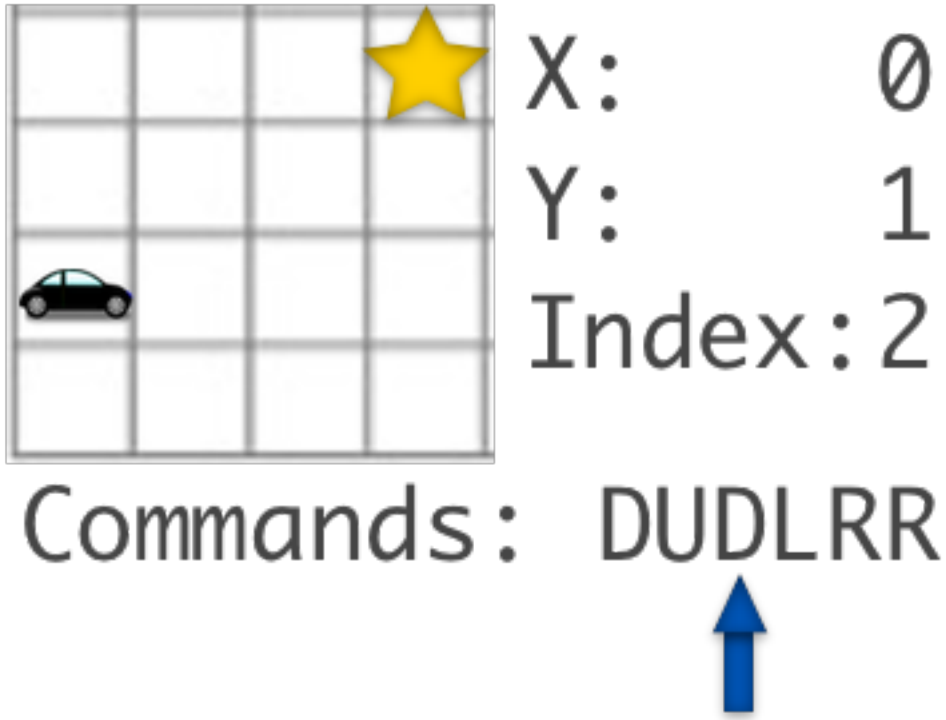
Executing Commands



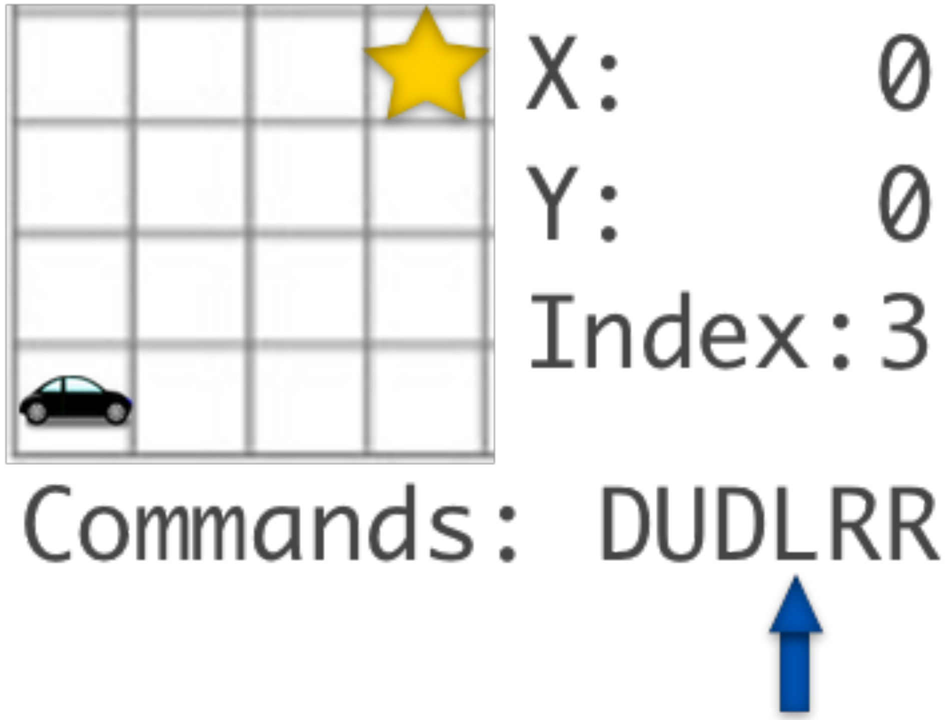
X: 0
Y: 0
Index: 0
Commands: DUDLRR



X: 0
Y: 0
Index: 1
Commands: DUDLRR

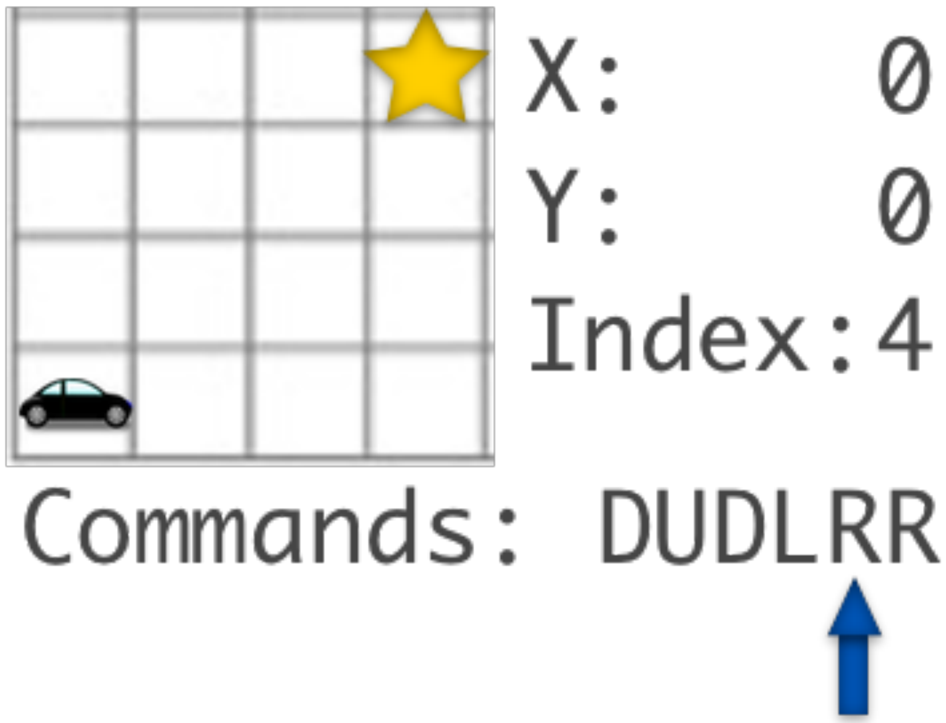


X: 0
Y: 1
Index: 2
Commands: DUDLRR

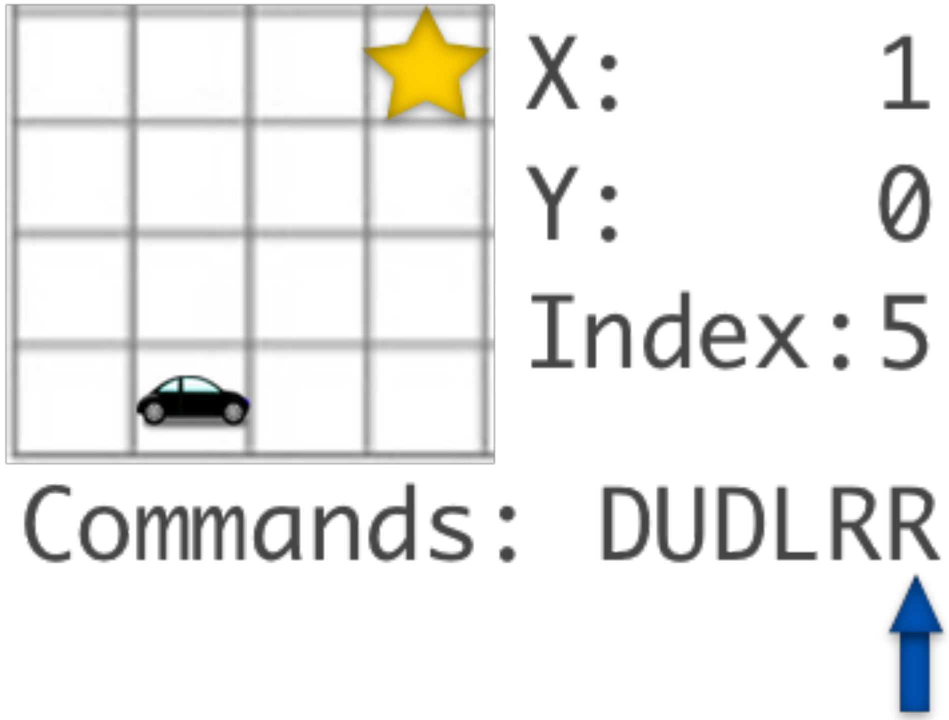


X: 0
Y: 0
Index: 3
Commands: DUDLRR

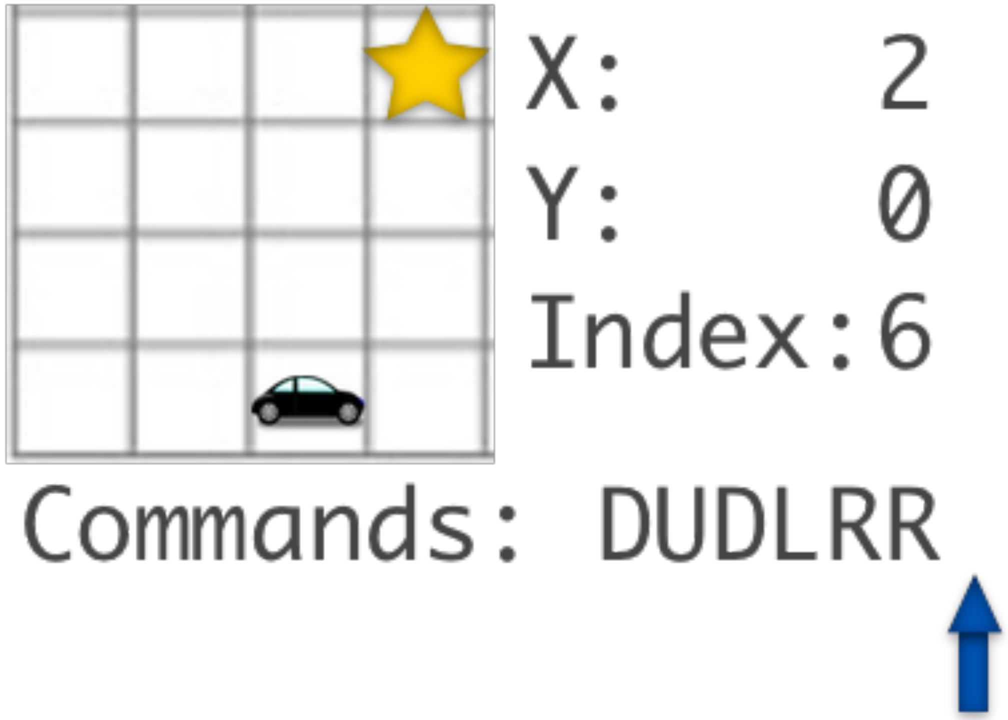
Executing Commands



```
X:     0
Y:     0
Index:4
Commands: DUDLRR
```



```
X:     1
Y:     0
Index:5
Commands: DUDLRR
```



```
X:     2
Y:     0
Index:6
Commands: DUDLRR
```

Done!

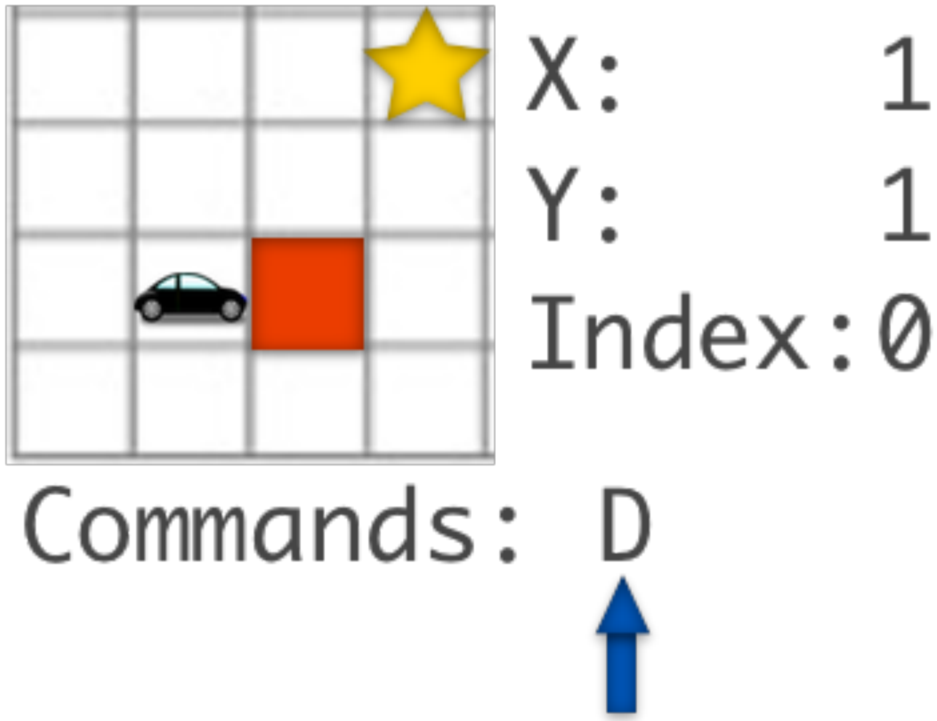
Observations

- The previous example has a special property:
  1. There's no benefit to appending commands anywhere but the end of the string

- Does this always hold?

Counter Example

Starting Point —>

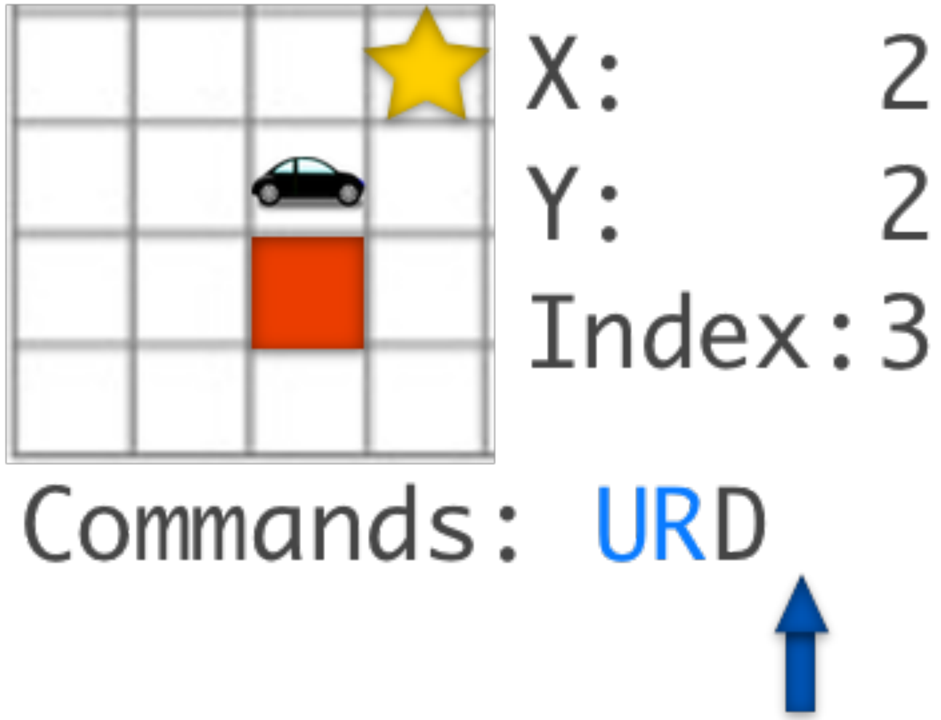


X: 1

Y: 1

Index:0

Commands: D

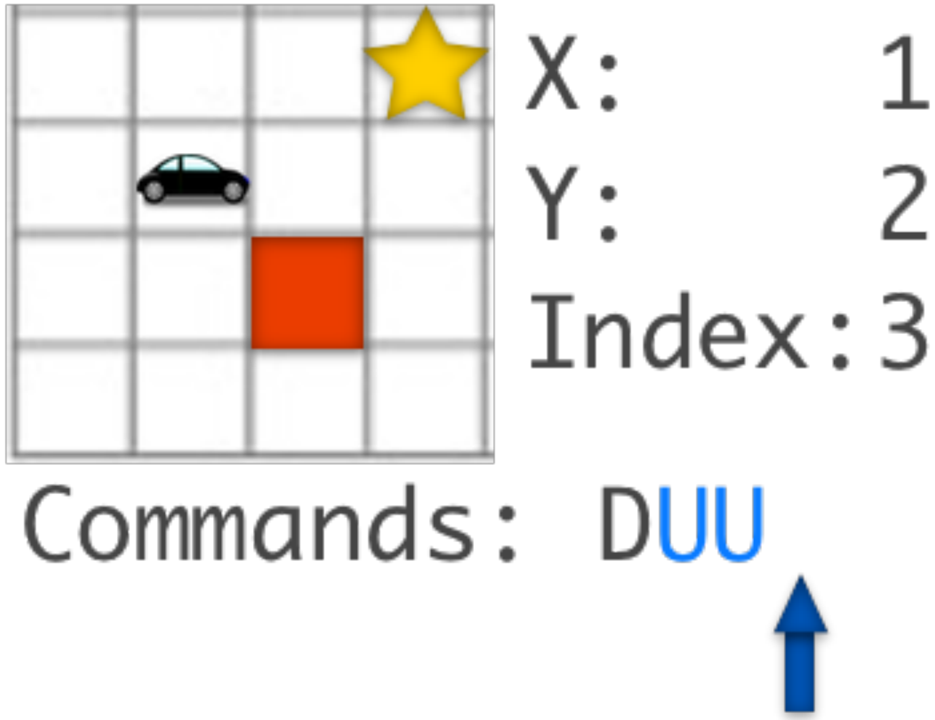


X: 2

Y: 2

Index:3

Commands: URD



X: 1

Y: 2

Index:3

Commands: DUU

More Thoughts

- For each command we can add, we have to consider what would happen if we add it anywhere in the current string
  - Large search space

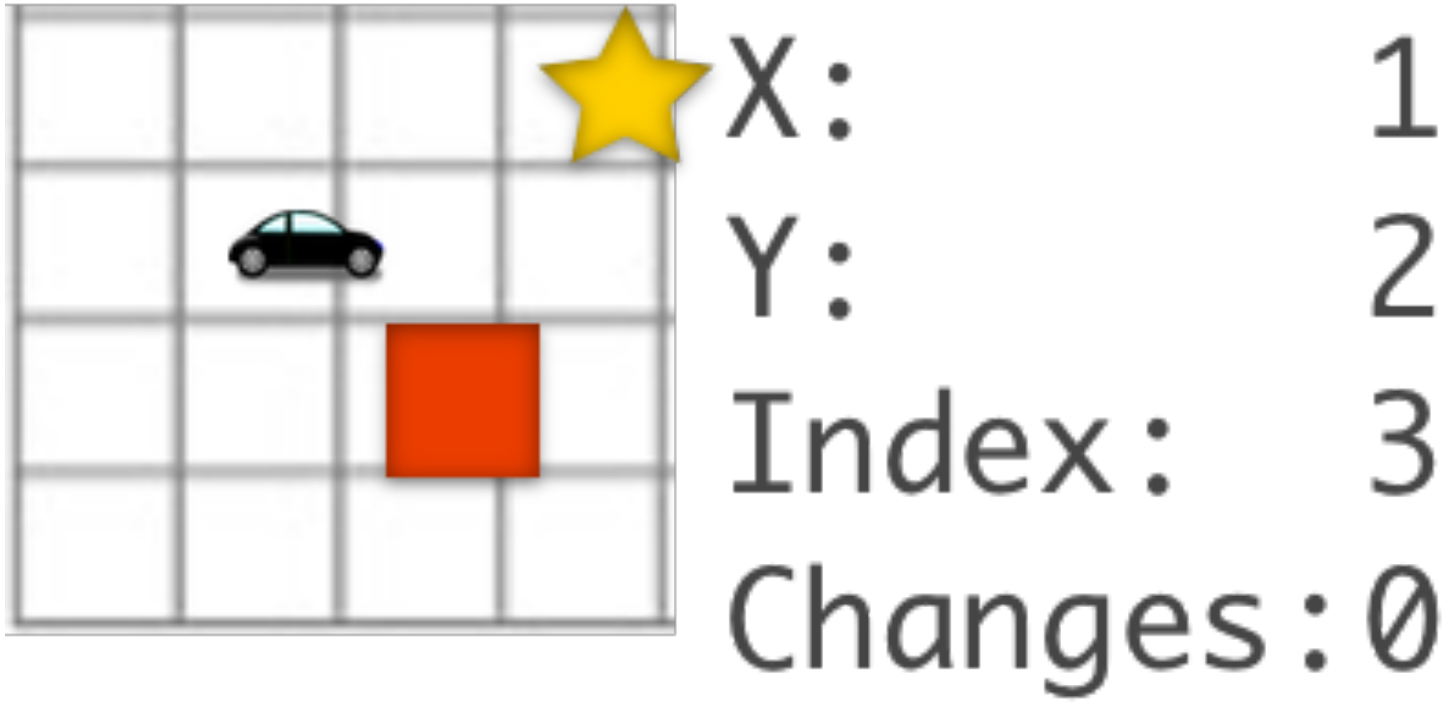
- This isn’t a textbook pathfinding problem, but we can borrow some ideas from A*, Dijkstra, etc.

More Thoughts

- Cost Function: The fewer changes we make to the command sequence, the better
- Secondary Cost Function: Manhattan Distance
- Greedy Strategy: Consider ‘Universes’ in the order of increasing changes (priority queue?)
- Terminate upon reaching the goal

A Universe

- It will be useful to define 'Universe' objects, which store the robot's state as we follow different possible sequences of altering the command string
- Example:



X: 1

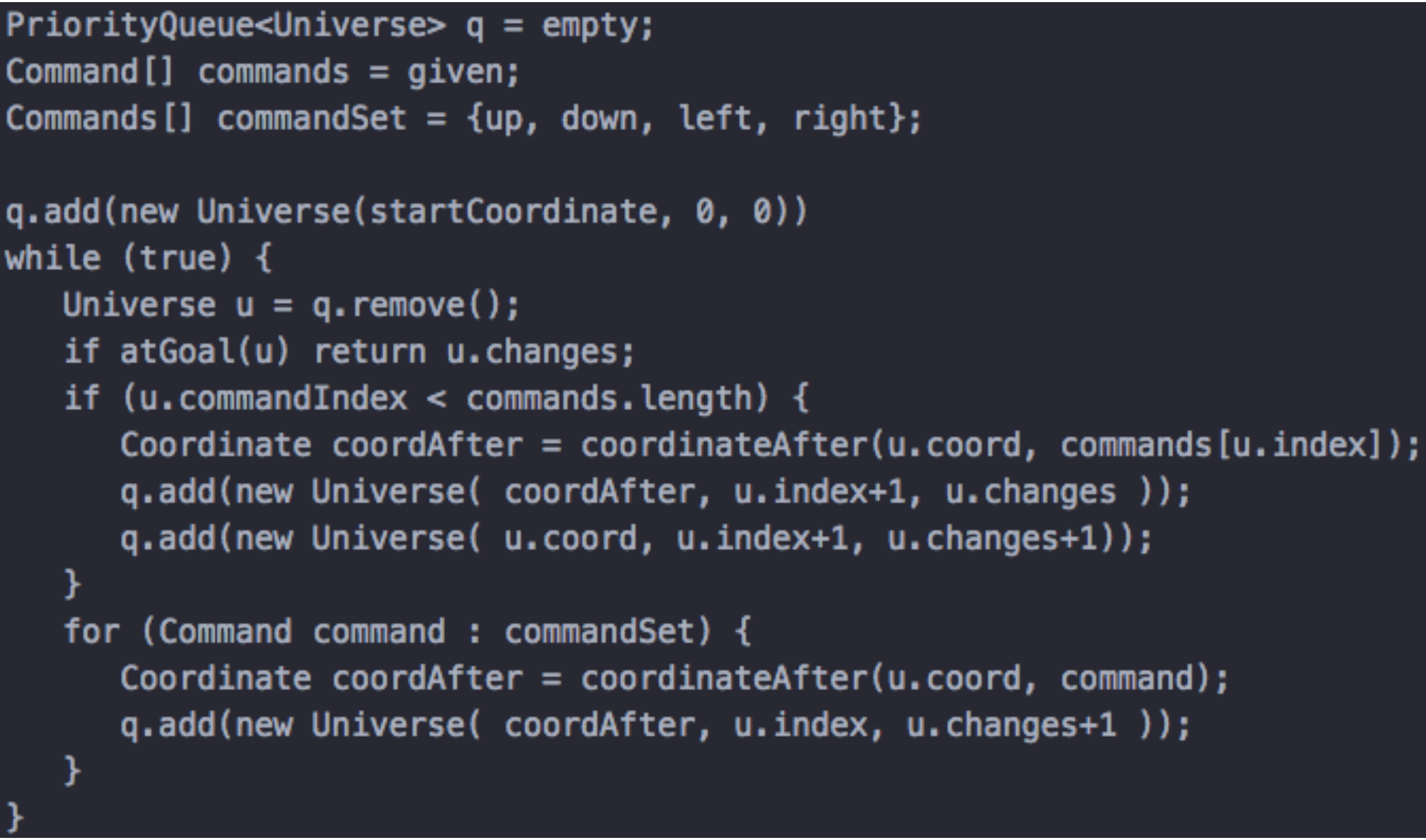
Y: 2

Index: 3

Changes: 0

Brute Force

Note: priority = # of changes



```
PriorityQueue<Universe> q = empty;
Command[] commands = given;
Commands[] commandSet = {up, down, left, right};

q.add(new Universe(startCoordinate, 0, 0))
while (true) {
   Universe u = q.remove();
   if atGoal(u) return u.changes;
   if (u.commandIndex < commands.length) {
      Coordinate coordAfter = coordinateAfter(u.coord, commands[u.index]);
      q.add(new Universe( coordAfter, u.index+1, u.changes ));
      q.add(new Universe( u.coord, u.index+1, u.changes+1));
   }
   for (Command command : commandSet) {
      Coordinate coordAfter = coordinateAfter(u.coord, command);
      q.add(new Universe( coordAfter, u.index, u.changes+1 ));
   }
}
```

Analysis

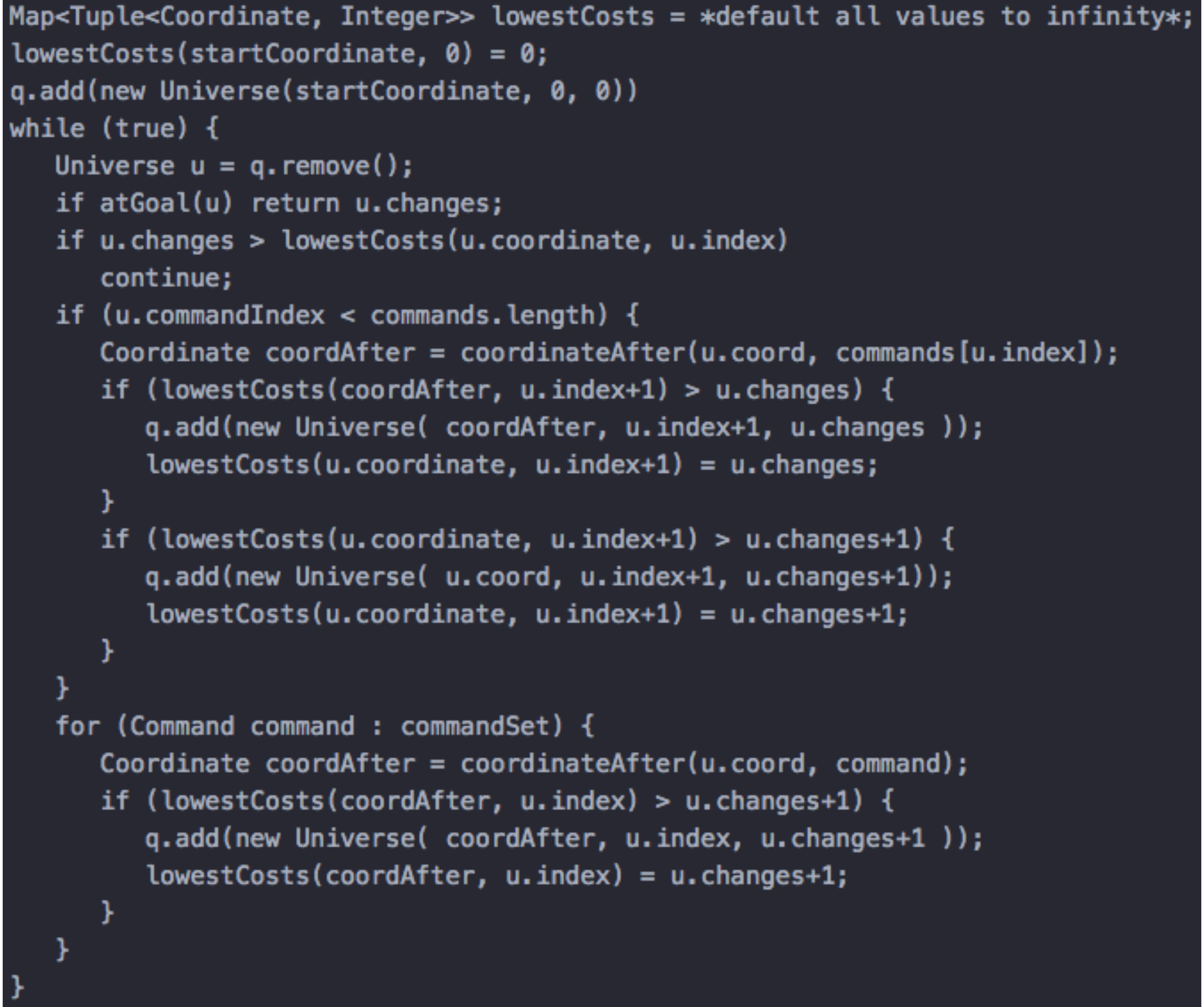
- Complexity: L * 5^n
    - L: length of command string
    - n: Solution (min # of changes to reach goal)

- This solution introduces a lot of redundancy, doubling the remaining problem size with every redundant Universe creation

Eliminating Redundancy

- LPA*: Lifelong Planning Algorithm
  - behavior like A*
  - records the cost of the best method for reaching any given point in map

- We can avoid redundancy by tracking the lowest cost of any (x position, y position, command index) tuple



```
Map<Tuple<Coordinate, Integer>> lowestCosts = *default all values to infinity*;
lowestCosts(startCoordinate, 0) = 0;
q.add(new Universe(startCoordinate, 0, 0))
while (true) {
   Universe u = q.remove();
   if atGoal(u) return u.changes;
   if u.changes > lowestCosts(u.coordinate, u.index)
      continue;
   if (u.commandIndex < commands.length) {
      Coordinate coordAfter = coordinateAfter(u.coord, commands[u.index]);
      if (lowestCosts(coordAfter, u.index+1) > u.changes) {
         q.add(new Universe( coordAfter, u.index+1, u.changes ));
         lowestCosts(u.coordinate, u.index+1) = u.changes;
      }
      if (lowestCosts(u.coordinate, u.index+1) > u.changes+1) {
         q.add(new Universe( u.coord, u.index+1, u.changes+1));
         lowestCosts(u.coordinate, u.index+1) = u.changes+1;
      }
   }
   for (Command command : commandSet) {
      Coordinate coordAfter = coordinateAfter(u.coord, command);
      if (lowestCosts(coordAfter, u.index) > u.changes+1) {
         q.add(new Universe( coordAfter, u.index, u.changes+1 ));
         lowestCosts(coordAfter, u.index) = u.changes+1;
      }
   }
}
```

Analysis

- Complexity (time *and* space): L * W * H
  - L: length of command string
  - W: Width of map
  - H: Height of map

Questions, Comments?