Updates

• Homework:
  – Homeworks #31, 34 (note: out of order) due Sunday 4/8 @ 11pm
  – Please take a look at recitation exercises before recitation

• Project:
  – Project 8 due tonight @ 11pm
  – Project 9 out, due 4/13 @ 11pm
  – Project 10 is online (under construction), due 4/27 @ 11pm
  – Lecture on 4/11 is optional and will have some hints for Project 10

• Exam #2 is coming up 5/2
• Be sure to fill out question sheet to receive attendance points today!
Today

- Project 9
- Robotic lawn mower exercise

https://www.cartoonstock.com/directory/m/mowing_the_lawn.asp
Project 9

• Fix compiler warnings
• Fix any failing unit tests
• Peer review
• Monitoring/invariants
Compiler Warning Flags

• Same as in Project 8
Compilation Notes

• Remember: Makefiles only rebuild source files if they have changed (or if they haven’t successfully compiled before)
• If you don’t change any files, warnings won’t appear on second compile
• Our solution: keep the –Werror flag  
  – Turns all warnings into errors  
  – Compilation won’t succeed until all warnings are removed
• If you disable the –Werror flag, remember to re-enable it and check build for warnings
• Turn in Project 9 code with all warning flags, especially –Werror, enabled  
  – Make sure you compile it with –Werror before turning it in
Unit Tests

• In Project 8, you wrote unit tests to achieve 100% transition, 100% branch, and sufficient data coverage

• For Project 9, you must maintain this coverage and have your tests pass
Unit Test Peer Review

- Groups assigned on canvas
- Same deal with the scribes and leaders and etiquette as previous projects
- Review testing coverage and traceability
  - Print out state chart (or have it on its own screen at all times during review)
  - For each transition: check that the test exists and has a commented annotation in code, and that the test is correct
  - Check code style, but don’t focus all your time on it
Runtime monitors

- In general: some sort of external entity that plugs into a system’s interface to monitor system output and state
- For ece642rtle: a separate ROS node that listens to messages sent by the turtle system
- You are given an interface for handling message interrupts
- By paying attention to the sequence of messages, can infer state and detect improper behavior
Logging Monitor

- Example provided to you to show you how message interrupt handling works
- Just prints every message it sees
- Doesn’t record state or check invariants
Invariants

- A condition that is always true during proper operation
- A monitor can check for invariant violations by keeping track of the state of the system
Step Monitor

- Invariant: turtle shall not move more than one square at a time
- Example provided to you to show how monitor can log state to check invariants
- Uses ROS_WARN to indicate an invariant violation

```c
/* Returns absolute value of x
 * WARNING: unsafe for edge-case values
 */
inline int abs(int x) {
    return x < 0 ? -1*x : x;
}
/*
 * Whenever the turtle moves, compare the current location
 * to the previous location and throw an invariant violation
 * if the locations differ by more than 1 in Manhattan Distance.
 */
void poseInterrupt(int x, int y) {
    // Print pose info
    // Last conditional makes sure that if suppress_double_visits is true, that the same pose isn't printed twice
    if (!suppress_double_visits || !moved) {
        (last_pose.x != x || last_pose.y != y)) {
            ROS_INFO("'Pose' was sent. Data: x = %d, y=%d", x, y);
        }
    }
    // Check that the turtle has moved before and that the Manhattan distance between the positions does not exceed 1
    if (moved && (abs(last_pose.x - x) + abs(last_pose.y - y) > 1) {
        ROS_WARN("VIOLATION: Difference between last coordinate (%d,%d) and current coordinate (%d,%d) is more than 1 square!", last_pose.x, last_pose.y, x, y);
    }
    // store last Pose in memory
    last_pose.x = x;
    last_pose.y = y;
    // Update this flag the first time the turtle moves
    if (!moved) {
        moved = true;
    }
```
Turn monitor

• Invariant: turtle shall not turn more than 90 degrees at a time
• You must write this invariant
• Project page gives some hints
• Run it with your turtle
  – If invariant is violated: is it because of your turtle code, or a bug in the monitor?
  – You do not have to fix invariant violations for Project 9
Project 9 summary

• Fix warnings
• Fix failing unit tests
• Peer review
• Invariant monitors
• Lots of choices to manage your time:
  – Unit tests compile independent of ROS project, so you can fix warnings first OR fix unit tests first
  – Write monitor at any time
• Make sure what you turn in compiles successfully and that the documentation is up-to-date
Looking ahead to Project 10

• Solve several new mazes (acceptance tests)
• Update documentation/traceability/unit tests
• Write more invariants
• Peer reviews
Project Questions?
Robotic Lawn Mower

- Essentially a Roomba with knives
- Sketch a safety case for it, one person at a time
  - See back of participation sheet for details
  - Ideas to get you started: behavior when picked up, battery temperature