18-642 Recitation #4

September 20, 2019

Updates

- Homework:
  - Last week homeworks graded on canvas
  - Next week homeworks due Wednesday night
- Homework grading
  - Points are mostly for effort
  - Read comments on canvas, even if you got full points
  - Full points does not mean you got the right answer
    - We'll try to cover some common issues in recitation
    - If you're not sure – ask!

Updates

- Projects:
  - Project 3 graded on canvas
  - Project 4 due tonight
    - Remember that code must comply with every Project 3 Checklist item
    - Issues from peer reviews should all be fixed
  - Project 5 released, due in a week

Updates

- Exam #1 in less than a month
  In class on Thursday Oct 10, 2019

Today

- Project 5
- HW discussion
  - HW #8-4 – vehicle safety questions
  - (Find your handed in answers to these homeworks NOW if you don't remember what you said in them)

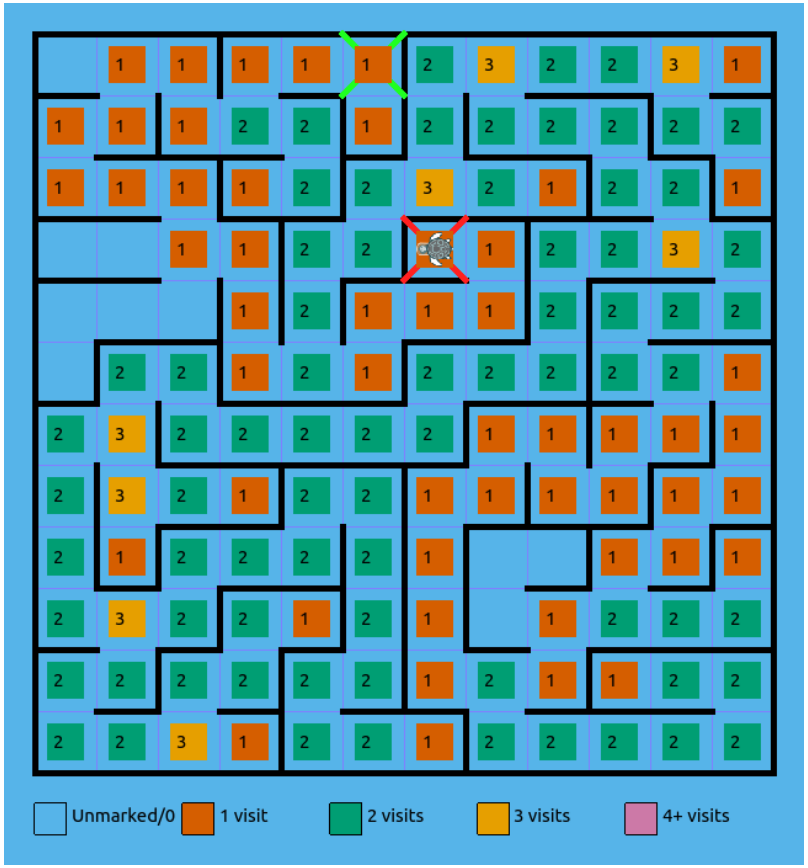
Project #4 Questions?

- Were TA meetings to get reviews on track helpful?

Project 5

- Keep track of how many times the turtle has visited a cell
- Split code into turtle and maze components
- Must build and solve m1.maze
- Try running on new maze: m2.maze
  - You do **not** have to solve this maze

Cell Visit Counts



- You are given a new parameter for displayTurtle:

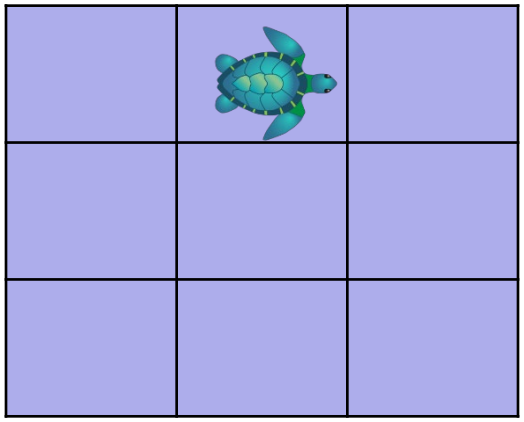
  displayTurtle(int nw_or, int visits);
- Call this function before you return in your main routine
- Function is for debugging/visualization only – accuracy depends on how well you are keeping track of data locally

student_maze and student_turtle

- student_maze.cpp
  - Does not decide how turtle moves
  - Translates moves from student_turtle into absolute coordinates
  - Think of this like building a simulator for the turtle
- student_turtle.cpp
  - Does not know its absolute coordinates
  - **cannot ask student_maze where it is or absolute orientation**
    - **cannot call bump(x1,y1,x2,y2)**
    - **student_maze should tell student_turtle if a move resulted in a bump**
  - Decides how to move based on current state and whether it has bumped
  - This is how a real robot (or human dropped into a maze) would behave

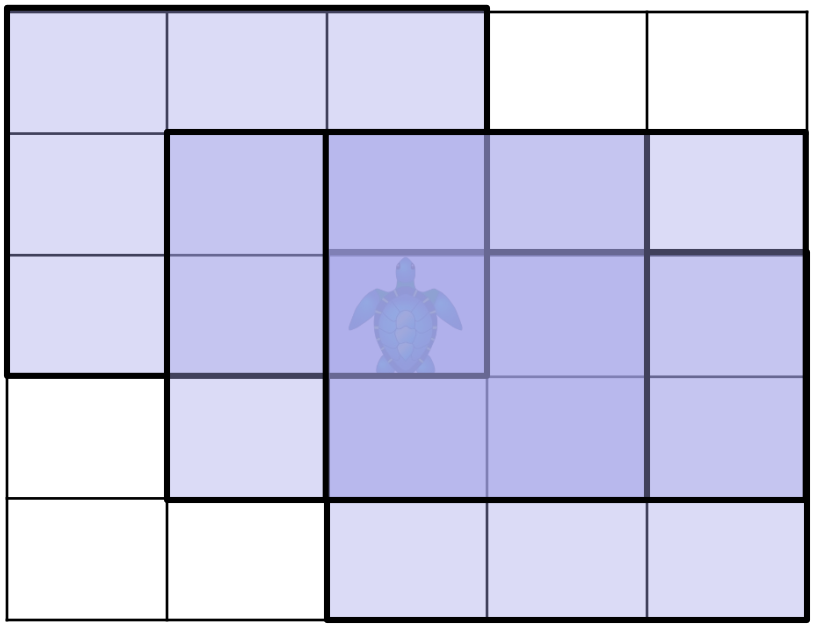
Example with 3x3 Maze

- Turtle is dropped into maze
  - Doesn't know where or which cardinal direction it's facing
  - Will draw its own map of the maze (2D array) – like a human carrying graph paper around



Ground Truth

Drawing its own maze

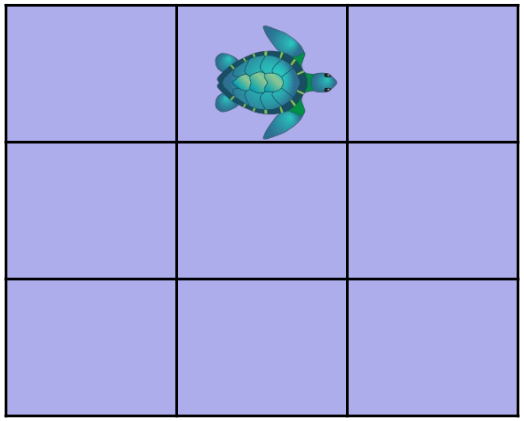


Turtle's local model

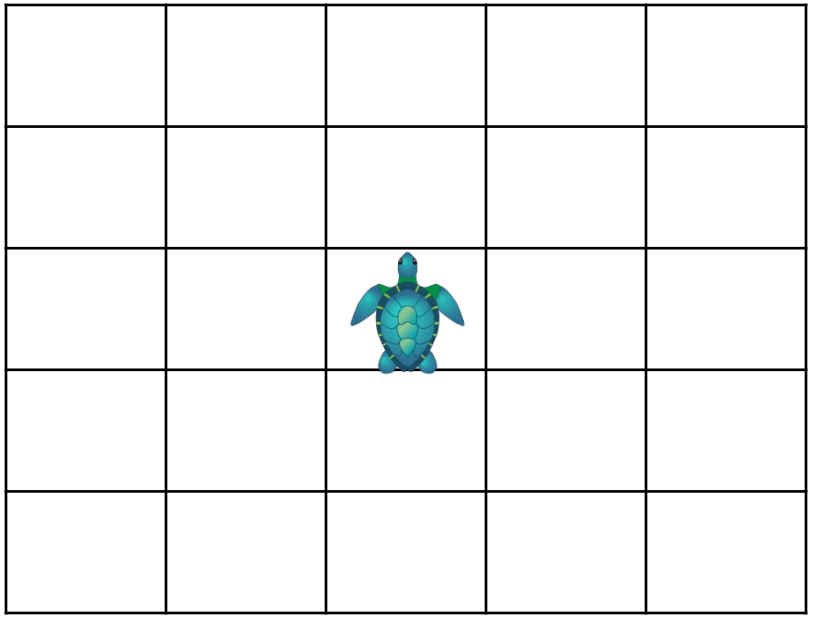
Real maze could be any of these (in rotation) or more!

- Turtle doesn't know where it is: could be anywhere in a 3x3 maze
- Knows it can move at most two squares in any direction from start point because 3x3 maze
- Therefore needs only a 5x5 array to keep track of any moves it makes (assuming it starts in the center of its array)
- Doesn't know which direction it faces
  - Can assume "local north"

Translating Actions: Start

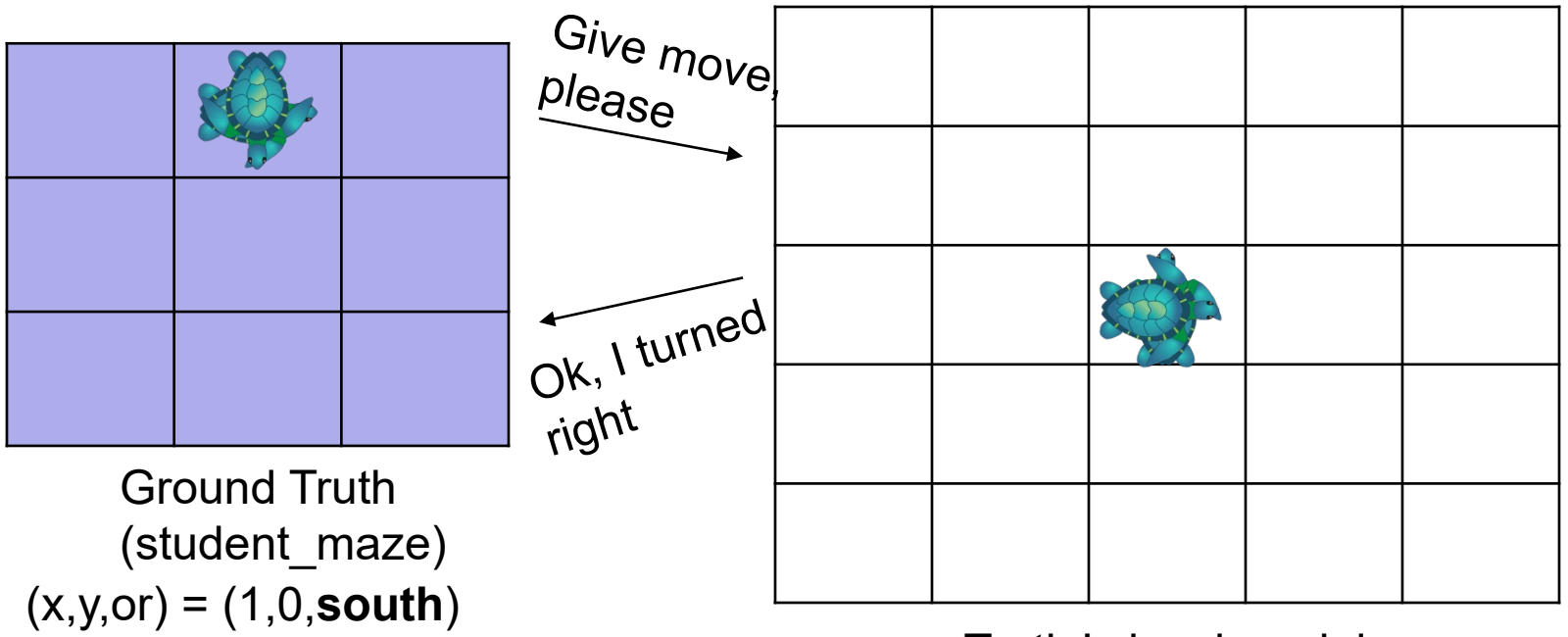


Ground Truth
(student_maze)
(x,y,orient) = (1,0,east)



Turtle's local model
(student_turtle)
(x',y',orient') = (2,2,north)

Translating Actions: turn right



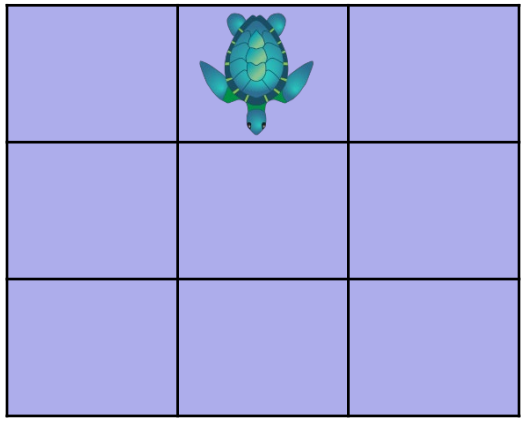
Ground Truth
(student_maze)
(x,y,or) = (1,0,**south**)

Turtle's local model
(student_turtle)

(x',y',or') = (2,2,**east**)

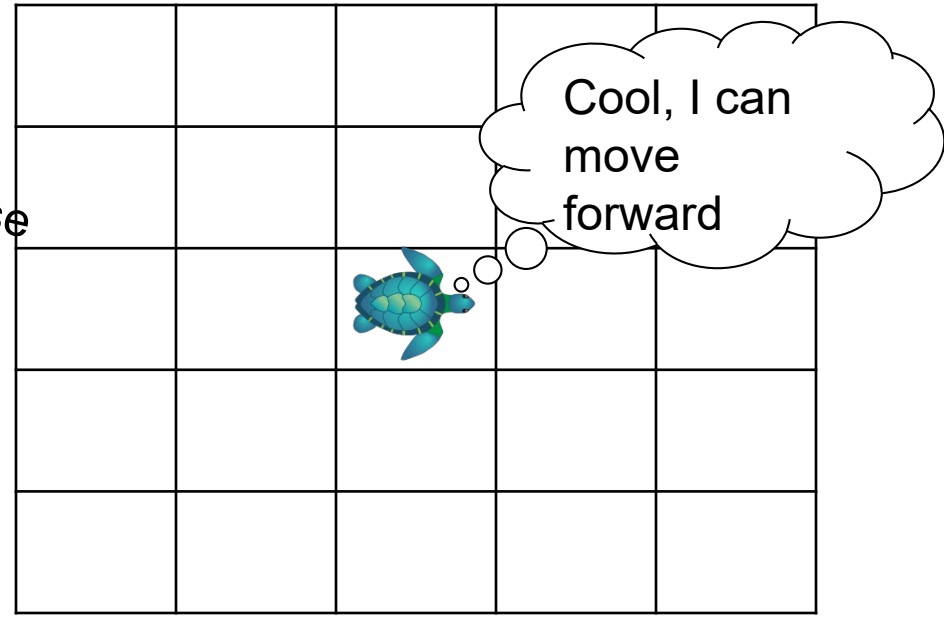
- Student_turtle decides to turn right
- Updates its local orientation *or'*
- Tells student_maze it turned right, which updates absolute orientation *or*

Translating Actions: check bump



Ground Truth
(student_maze)
(x,y,or) = (1,0,south)

Bump = false

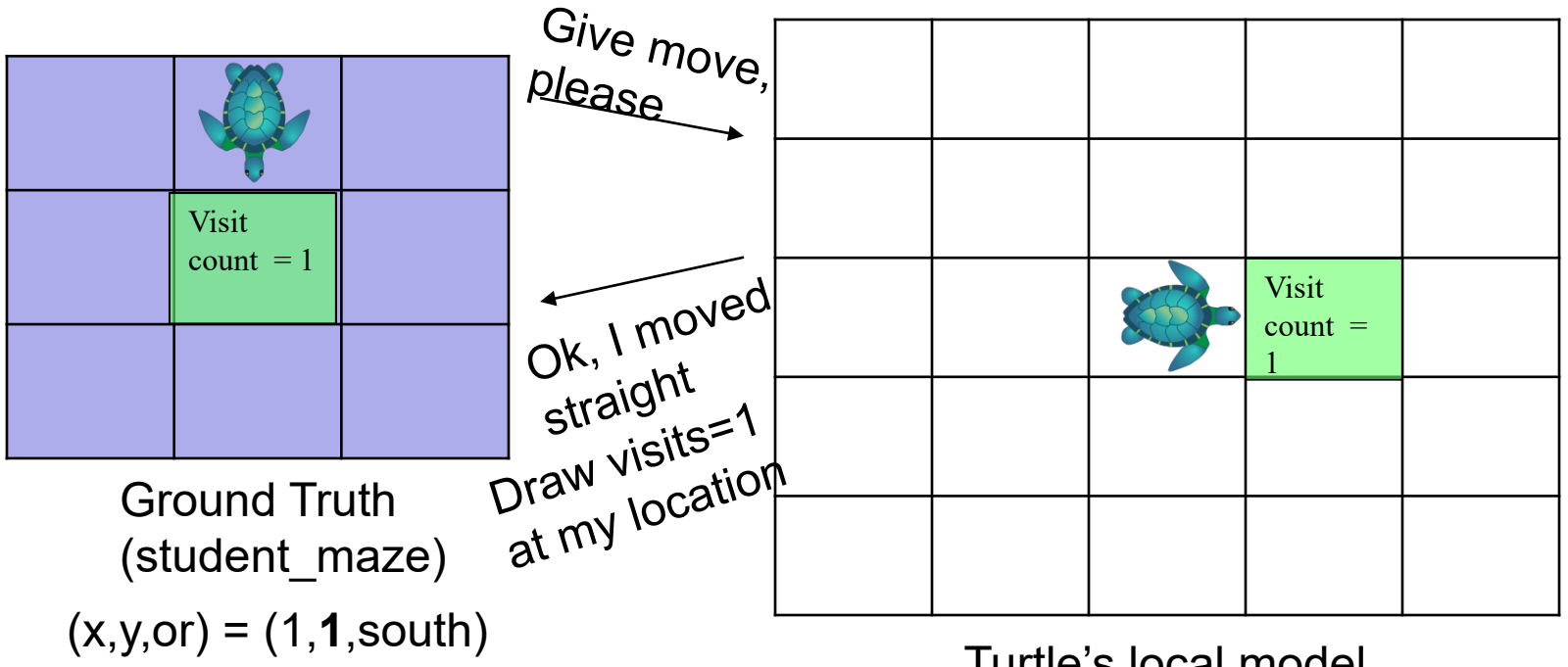


Turtle's local model
(student_turtle)

(x',y',or') = (2,2,east)

- Student_turtle wants to check for bump
- Student_maze sends whether there's a wall in front of turtle based on *x,y,or*

Translating Actions: Move Straight



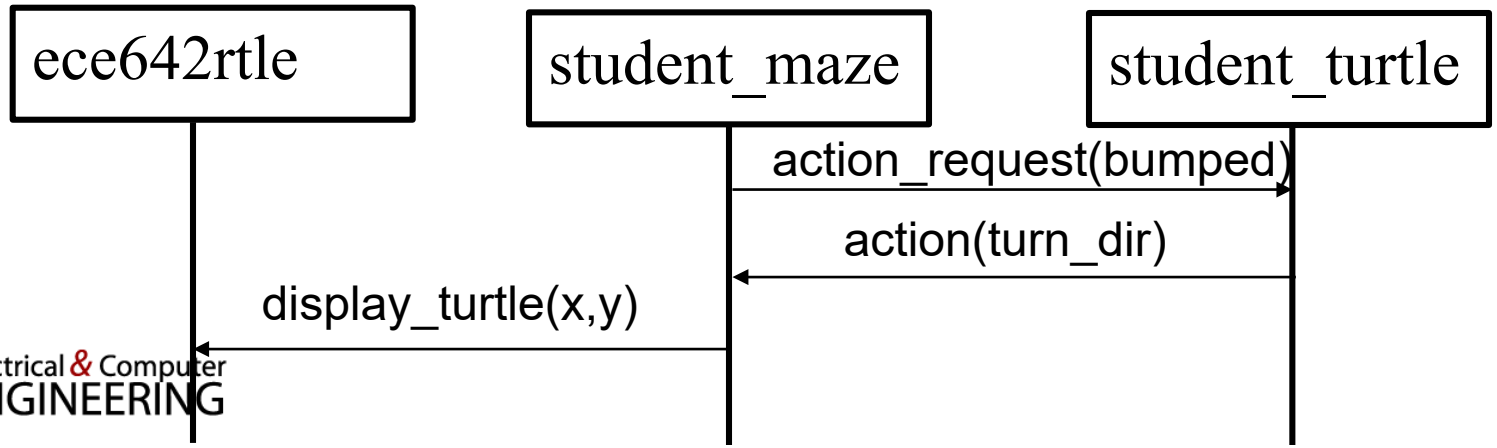
- Student_turtle decides to move straight
- Updates its local coordinates *x',y'* and local array of visits counts
- Tells student_maze it moved straight, which updates absolute coordinates *x,y* and visits count at *x,y*

Review:

- student_maze calls decision function in student_turtle
  - Hint: pass return value of bumped() to turtle using this function
- student_turtle decides how to move based on internal logic
- student_turtle updates internal array of cell visit counts
- student_turtle returns an action to student_maze
- student_maze translates to absolute coordinates, calls displayTurtle(), etc

Sequence Diagrams Preview

- You will see this in class this week
- Describes a scenario (interaction between components)
- Time flows top to bottom
- Boxes indicate components
- Arrows indicate messages/calls from one component to another
- Use them to describe interaction between components in Proj 5 writeup



Project 5 Questions?

- Review:
  - Keep track of the times a cell has been visited
  - Split into maze and turtle
  - Run on new maze (but don't have to solve)
    - You'll need to solve it in a future project
- Builds a foundation for Project 6
  - Will introduce more mazes not solvable by LHR/RHR

PROJECT 5 IS HARDER

I warned you, but did you listen to me?
Oh, no, you knew it all, didn't you?
*Oh, it's just a harmless little **project**, isn't it?*
Well, it's always the same. I always tell them--

https://montypython.fandom.com/wiki/Tim_the_Enchanter

Review

- SCC vs MCC
- Equivalence Classes
- MCDC

MCC and SCC

- McCabe's cyclomatic complexity
  - Counts # of if/while/for conditionals in the code
- Strict cyclomatic complexity
  - Includes +1 for every condition within a branch
- `if (a < 0 && b > 0)` adds +1 to MCC, +2 to SCC

MCC/SCC in student.cpp

- Check whether your tool computes MCC or SCC
- Find branch statements with multiple conditions

```
mod = true;
if(z == true && aend == false) {
if (nw_or == 1) pos_.setY(pos_.y() - 1);
```

- Add # of extra statements to MCC to get SCC
  - Or subtract from SCC to get MCC
- For student.cpp, SCC = MCC + 1

Equivalence Classes

- Any input in an equivalence class is expected to cause the code to behave the same
  - If input a and input b are in the same equivalence class, they will both take the same branches in the code
- Always remember special case values:
  - NULL for pointers
  - NaN for floats
  - Overflow values (INT_MIN, INT_MAX)

To achieve MC/DC coverage

```
if((a > 5) && ((b < 17) || (b > 97) || (b == 42)))
```

- Inputs to make entire decision true and false
- Inputs to make each conditional true and false
  - a > 5, b < 17, b > 97, b == 42
- Each condition affects decision independently
  - Hold all other conditionals fixed at some value
  - Changing Boolean value of the condition changes the outcome of the decision
  - Ex: b < 17 affects decision?
    - Hold a = 6 (making a > TRUE), b < 42 (making b > 97 FALSE, b == 42 FALSE)
    - If b < 17, decision is TRUE, else decision is FALSE

Questions?

- **FAIR WARNING:**
  - Significant increase in complexity/difficulty for projects 5/6/7 compared to projects 2/3/4
  - START EARLY!
  - Plan to be done BEFORE recitation in case you hit an algorithm roadblock!

Homework Discussion

- 8-4a. *(Driver age?)*
- 8-4b. *(Should Govt Regulate?)*
- 8-4c. *(Driver released from Jail)*
- 8-4d. *(UA Class Action)*
- 8-4e. *(Police Officer Trial)*
- 8-4f: *(What if it's you?)*
- 8-4g: *(Should this man be in jail?)*
- 8-4h: *(Should SW designers be liable?)*