The Problem

Definition
Every engineering problem begins with a need that winds up driving the solution. Engineering is the art of finding a feasible solution to the problem that meets all of the constraints (be they cost, resources, durability, etc.)

- Engineering is the study of trade-offs
- Techniques for managing the complexity apply across domains
- The right solution can be elegant, but is often only visible only in hindsight
The Cat
The Cat
The Door
The Door
The Problem

- The cat needs to be indoors by nightfall
- Getting him in during the evening can be difficult
  - Chasing the cat
  - Bribing him with food
  - Not always successful
- Cat comes in approximately every two hours during the day
The Solution

• Need to make the cat door one-way, so the cat can reenter the house, but not leave again.
• Set it to one way at 5PM, cat will be in by 7PM (before darkness).
• Don’t have to be home to get the cat in.
Commercial Solution

- Try to go for a commercial solution
- Buy rather than build
- Not exactly what you need, but can perhaps be made to work.

- “Hav-a-Hart” live animal trap.
Commercial Solution
Commercial Solution
Commercial Solution
Need a new approach

FAILURE!
The Door
New approach: Inside Door

- Add a second door inside the first on the wall
- Hinged up for normal operation
- Down to form a “double seal” or one way valve for the cat door
- Larger than opening, cannot be pushed through.
How would I make a product?

Simple two "paw" button action to select modes

Cat and moon shaped lights to indicate various modes

Porch sensor reads only your cat's microchip number

Magnetic latch to stop the flap opening in the wind

450

1005

CMPE-013/L: “C” Programming
Robust In-Only Option

- Use the existing guillotine door rails
- Allow the cat entry, but block the door from opening to the outside
- Needs to be robust to clever cat
- Needs to be accepted by cat
  - Doesn’t work if cat won’t use it
- Will need some refinement to get right
The Prototype
The Prototype in Action
One-way Action Confirmed

- Cat can get in
- Cat cannot get out

- Prototype successful, move on to better implementation
Need to refine prototype

FAILURE!
Happy Cat!
Conclusion

Inspiration and Iteration are two very necessary parts of finding your way to a solution that works well. Failure is the genesis of further experimentation, which leads to better design.

- Fail early and often
  - Early reduction of less promising ideas
- Be flexible
  - Don’t get married to the first solution you try
  - Be ready to jettison something that isn’t working

Experimentation leads to more understanding
New Problem
CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
More FSM
Our Roach State Machine

Hiding

- Light Goes On
  - Drive Fwd.

- Light Goes Off
  - Stop

Driving_Forward

- Hit Obj
  - Left Reverse, Start Timer

- Timer Expires (Light On)
  - Straight, Drive Fwd.

Backing_Up

- Hit Rear Obj
  - Forward Right, Start Timer

- Timer Expired
  - Left Reverse, Start Timer

Evade Forward

Timer Exp (Light Off)
- Stop
Entry and Exit Events

- We can combine these duplicated actions into entry and exit events
- Instead of having a repeated actions we now have actions associated with states
  - For example: everytime we enter hiding we should stop the roach
Entry Roach State Machine

- **Hiding**
  - Entry: STOP
  - Light Goes On ➔ Light Goes Off

- **Driving Forward**
  - Entry: Drive Forward
  - Timer Exp (Light Off)
  - Hit Obj ➔ Timer Expires (Light On)

- **Backing Up**
  - Entry: Left Reverse, Start Timer
  - Hit Rear Obj ➔ Timer Expired

- **Evade Forward**
  - Entry: Forward Left, Start Timer
Coding Entry and Exit Events

- State machines with these events can no longer merely change state. They need to make use of a `makeTransition` flag.
  - This is needed to ensure the entry and exit events are handled correctly.

  ```
  nextState = BACKING_UP;
  makeTransition = TRUE;
  ```
Making the state transition

- The actual state transition requires a **recursive** call to ensure that the events are handled correctly.

```c
if (makeTransition == TRUE) {
    RunRoachStateMachine(EXIT_EVENT);
    myState = nextState;
    RunRoachStateMachine(ENTRY_EVENT);
}
```
FSM to HSM

• With the addition of the exit and entry events we can now combine similar actions and remove duplicate behavior.

• This allows for a cleaner design but more complex problems still cause “state explosion.”
  – Many of these extra states come from repetition.

• HSM’s address this problem by introducing superstates with substates below them.
Messy State Machine

- **Off**
  - Entry: turn motor off
  - Exit: halt main timer
  - buttons: press
  - transitions:
    - OUT_OF_CHARGER: reset brush state
    - ES_TIMEOUT(Resume): reset brush state
    - ON_CHARGER(Vbat <)
      - reset brush state
    - ON_CHARGER(Vbat >)
      - reset brush state
    - ES_TIMEOUT(Vbat >): start timer
    - ES_TIMEOUT(Vbat <): start timer
  - states:
    - Charged
      - Entry: LED on
      - Exit: LED off
    - Charging
      - Entry: blink LED
      - Exit: LED off
      - transitions:
        - ES_TIMEOUT(Vbat <)
        - ON_CHARGER(Vbat >)
    - Brushing
      - Entry: motor on
      - Exit: motor off
      - transitions:
        - BUTTON_PRESS: start timer
        - ES_TIMEOUT(hiccup): if count <= 3
    - Hiccuping
      - Entry: counter++
      - Exit: set hiccup timer 30 sec
      - transitions:
        - BUTTON_PRESS: start hiccup timer

- **Charging**
  - Entry: LED on
  - Exit: LED off
  - transitions:
    - ES_TIMEOUT(Vbat <)
    - ON_CHARGER(Vbat >)

- **Brushing**
  - Entry: motor on
  - Exit: motor off
  - transitions:
    - BUTTON_PRESS: start timer
    - ES_TIMEOUT(hiccup): if count <= 3

- **Hiccuping**
  - Entry: counter++
  - Exit: set hiccup timer 30 sec
  - transitions:
    - BUTTON_PRESS: start hiccup timer

**Reset Brush State**
- count = 0
- stop hiccup timer
- stop main timer
- set hiccup timer 30s
- set main timer 2 min

**Electric Toothbrush FSM**
- ES_TIMEOUT(main): reset brush state
- ES_TIMEOUT(hiccup)/(count > 3): reset brush state
- BUS_TIMEOUT(main): reset brush state
- BUTTON_PRESS: stop resume timer
- BUTTON_PRESS: start resume timer
- OUT_OF_CHARGER: reset brush state
- ES_TIMEOUT(Resume): reset brush state
- ON_CHARGER(Vbat <)
  - reset brush state
- ON_CHARGER(Vbat >)
  - reset brush state
- ES_TIMEOUT(Vbat >): start timer
- ES_TIMEOUT(Vbat <): start timer
- transitions:
  - OUT_OF_CHARGER: reset brush state
  - ES_TIMEOUT(Resume): reset brush state
  - ON_CHARGER(Vbat <)
    - reset brush state
  - ON_CHARGER(Vbat >)
    - reset brush state
  - ES_TIMEOUT(Vbat >): start timer
  - ES_TIMEOUT(Vbat <): start timer
  - transitions:
    - OUT_OF_CHARGER: reset brush state
    - ES_TIMEOUT(Resume): reset brush state
    - ON_CHARGER(Vbat <)
      - reset brush state
    - ON_CHARGER(Vbat >)
      - reset brush state
    - ES_TIMEOUT(Vbat >): start timer
    - ES_TIMEOUT(Vbat <): start timer
In Charging State Machine

InCharging HSM
(lower level)

INIT

Charging

- entry blink LED
- turn charging on
- exit turn charging off
- set/start resume timer

Charged

- entry steady LED
- exit set/start resume timer

ES_TIMEOUT(resume)
[Vbat > FULLY_CHARGED]

ES_TIMEOUT(resume)
[Vbat < FULLY_CHARGED]
Line follow

I.O.O.D. Amm0

Find Enemy
gcc -Wall

RPG
make kilp
Environment Setup

- **Lab Computers**
  - Work from your X:\ Drive and open files directly. This is simplest. Notepad++ recommended.

- **Windows**
  - Use Putty or other terminal program to connect to the Unix servers to run and compile code.
  - Use Notepad++ with the NppFTP plugin to edit the files.
  - Cygwin for the brave.

- **Linux/Mac:**
  - You have support for this. Up to you to figure it out.

- Regardless: Must run on UNIX cluster for grading
Room Format

Binary: RPG

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(repeated)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Room format is the same as discussed in the file I/O lecture except that the files are encrypted with XOR and unique encryption keys.
- To figure out the key for a particular room you add the room number to a base key given in game.h

_encode absolute path RoomFiles/room32.txt_
Reading Room Files

- Each time you enter a room you will read in its entire contents and parse it appropriately into a struct.
- After that (for drawing the screen or determining which room to load next) you will reference the struct only.
- Re-reading the file for this data will result in lost points.
- You will only re-read the file when you have exited the room and re-enter it.
Terminal Programs

• While standard users typically use GUI’s these days, terminal programs are still very important.
  – Especially for remote systems where you do not have direct access to the machine.

• There are several frameworks to make terminal programs (curses comes to mind) but we will use one of the oldest: VT100
  – Originally designed when the monitors had serial connections
VT100 Escape Codes

Every sequence starts with the ESC character or 0x1B and is followed by a command.

For example "\x1b[2J" will clear the screen.

Similar commands will change colors and allow the position of the cursor on the screen to be modified.

\90\% \91\%