CSE 466: Course Project

- Complete a large project that embodies the major course topics
- Project should be simple but expandable
- The project should include:
  - Multiple device communication
  - Deal with constrained resources
  - Control hardware by directly manipulating the I/O
  - Introduce an embedded OS
  - Participate in a multi-agent project – team effort
  - Use current technology

The Flock

- Two week project to tie together everything we've learned in 466
  - Programming microcontrollers
  - Wireless radio communication
  - Embedded operating systems
- A piece of "performance art"
- Allows nodes programmed by different students to work together
- Exposes some problems of scale in building sensor networks

Basic Idea of the Flock Project

- Each node ("bird") sings a song
- It then listens to its neighbors to hear what they sang
- It makes a decision as to which song to sing next
  - This can lead to an emergent behavior – property of the group
  - We'll be trying for an effect that propagates a song around the flock
- If it is startled (by a shadow cast on its light sensor), then it makes a "scared" noise and informs its neighbors who will do the same
- Room for experimentation
  - New songs – not just birds
  - New algorithms for determining next song
  - Using time of day and other sensor data in the algorithm
  - Any other ideas you come up with

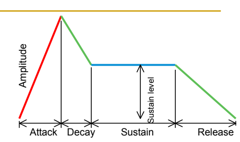
Flock Process Flow:

- A) INITIALIZATION STATE: (only used when birdie is first turned on)
  - set SONG = Local # % 16 (choose one of 16 songs)
  - Wait to receive a packet of type AdjustGlobals then go to C
- B) WAIT STATE
  - Wait to receive a packet of type AdjustGlobals or SingSongN
  - IF (AdjustGlobals) set SONG = random song; go to C
  - IF (SingSongN) set SONG = received in message; go to D
- C) CLEAR STATE
  - With radio off, clear FIFO data (all historical data)
  - Wait for random amount of time (1000- 4000 milliseconds)
- D) SING STATE
  - With radio off, sing birdiesong(SONG)
  - If got to SING STATE from a SingSongN packet goto WAIT STATE after sending a SangSong message
- E) Start the radio and set listen timer for random t ϵ [minListen, maxListen] msec
- F) Set the send timer for minListen/2 milliseconds and send the "I sang song" message
- G) When listen timer runs out, decide next song
- H) Repeat steps D through H.

Flock Details: Sing

- Turn the radio off– we can't handle 416 usec interrupts while generating sound
- Sing the song using Timer1 for PWM and Timer2 for tempo and ADSR control
- Turn the radio on and do any housekeeping required
- We'll get details of sound generation next Monday from Bruce Hemingway – the creator of the flock

Sound Generation



- Used PWM with low pass filter to generate sound
- Taxes the systems memory and CPU
  - Wave and sequence tables used to generate the sound take up a large part of memory
  - Uses many processor cycles, requiring efficient coding
    - We will use a 15.625kHz sampling rate on the mica2dots (4MHz)
    - Processing has to complete in 256 cycles
- Timing errors are easily detectable (<10 ms)
  - Gives quick feedback on program accuracy
  - Code inaccuracies are generally audible

Flock Details: Listen

- Arriving packets need to be time-stamped
- Packets from Node 0 must be specially treated – they may contain global parameters
- Arriving packets must be strength-stamped for RSSI value – special radio stack required

Flock Details: Decide

- Need algorithm for what song to sing next
- Similar to Cellular Automata, like Conway's Game of Life
- Goals:
  - Sing the same song for a little while
  - Songs start, then spread, then die out

What are Cellular Automata?

- Computer simulations which emulate the laws of nature
  - Rough estimation – no precision
- Discrete time/space logical universes
- Complexity from simple rule set
  - Reductionist approach
- Deterministic local physical model
  - Ensemble does not have easily reproducable results due to randomization and limits of communication

History

- Original experiment created to see if simple rule system could create "universal computer"
- Universal computer (Turing): a machine capable of emulating any kind of information processing through simple rule system
- Late 1960's: John Conway invents "Game of Life"

Game of Life

- Simplest possible universe capable of computation
- Basic design: rectangular grid of "living" (on) and "dead" (off) cells
- Complex patterns result from simple structures
- In each generation, cells are governed by three simple rules
- Which patterns lead to stability? To chaos?

Simulation Goals

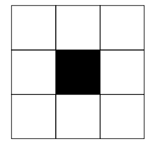
- Avoid extremes: patterns that grow too quickly (unlimited) or patterns that die quickly
- Desired behavior:
  - No initial patterns where unlimited growth is obvious through simple proof
  - Should discover initial patterns for which this occurs
  - Simple initial patterns should grow and change before ending by:
    - fading away completely
    - stabilizing the configuration
    - oscillating between 2 or more stable configurations
  - Behavior of population should be relatively unpredictable

Conway's Rules

- Death: if the number of surrounding cells is less than 2 or greater than 3, the current cell dies
- Survival: if the number of living cells is exactly 2, or if the number of living cells is 3 (including the current cell), maintain status quo
- Birth: if the current cell is dead, but has three living cells surrounding it, it will come to life

For Each Square . . .

- Look at nearest neighbors (8 of them)
- 256 possible states (2^8)
- Decide on square's next state (dead/alive, on/off)

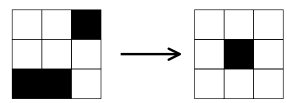


The Rules for Life

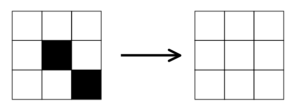
- If a square is black ("on") then it will be black at the next step if 2 or 3 of its neighbouring squares are black
- A white ("off") square will become black only if it has exactly 3 black neighbouring squares
- Otherwise a square will be white the next step (overcrowded or lonely)

Examples

- We can have birth…



- Or death…



- A nice implementation is at: http://www.math.com/students/wonders/life/life.html

Types of behaviour in the Game of Life…

- Still life objects – unchanging (e.g. four-block)
- Simple repeating patterns (oscillations)
- Part of the system can leave the rest and travel (movement - gliders)
- The system can die out completely
- The system grows randomly before stabilising to a predictable behaviour
- The system grows forever (quite rare and difficult to find)

Chaos…

- All behaviour in the Game of Life is chaotic – it is very sensitive to the starting state and is completely altered if the system changes a little (e.g. just like the weather)

Flock Details: Decide

- Goals:
  - Sing the same song for a little while
  - Songs start, then spread, then die out
- Algorithm?
  - Determine nearest songs
  - If our song = any of nearest n, then repeat song
  - If all same, switch to different song
  - If none same, switch to different song

Flock Details: Decide

- Algorithm?
  - Determine nearest songs
  - If our song = any of nearest n, then repeat song
  - If all same, switch to different song
  - If none same, switch to different song
- How do we evaluate this?
- How can we predict it's effectiveness?

For Wednesday

- Try to figure out a way of determining whether 50 birds will ever sing the same song at approximately the same time
- Make three suggestions for improvement to any aspect of the flow or decision algorithm to improve chances of accomplishing this
- Do not consider trivial algorithms

The Concert – Dec 10 – 9:30AM

- Final demo for the class is a concert
- Each student has a mote to contribute (35 motes)
- Same rules, different code in each mote
- The motes have to "qualify"
  - We will have testing scripts to simulate the flock and eliminate nodes that may cause problems
  - Used for grading projects