CS3505/5020
Software Practice II

Project #3 Q/A
Quiz
State Machines in UML – intro
XNA network support
First Quiz Thursday

- Closed note / book / laptop
- Lightweight quiz
- Focus on C# and XNA fundamentals
  - If it’s on a slide (or was talked about because of a slide), it may be asked.
  - I won’t be tricky, but I will seek to know if you can apply what you’ve learned.
A Brief Introduction to State Machines

- State machines are used to abstract the way an object’s state (the machine’s state) changes in response to various events
  - An older, common representation of state machines represented states as a node:

![Diagram of state machines with nodes A, B, and C]
A Brief Introduction to State Machines

- Events, or inputs to the machine, cause the machine to transition from one state to another.
- Transitions are represented with arrows (usually labeled with transition information).
- Start states typically labeled with a special syntax
A Brief Introduction to State Machines

- Let’s ignore the syntax for now. Consider:
  - How many states could exist in an object with 18 Boolean variables?
    » Should we draw nodes for every state?
  - It is possible to have many inputs and many states for an object (for a machine).
  - In some domains, precision is required. Enumerate all states and transitions.
  - For software engineering, we want a representation that helps us clarify our thoughts about state and transitions in a system.
UML State Machines

- Basic structure:

  - Initial state
  - Final state

  Transition

  Trigger [Guard] / Effect

  State
  - State actions
UML State Machines

In the UML State Machines diagram:

- **Inactive**
  - Transition to **Installed**
  - Transition to **Uninstalled**

- **Idle**
  - Transition to **Powered on**
  - Transition to **Powered off**
  - Transition to **Sense thermostat**
  - Transition to **No Call for heat**

- **Heating**
  - Transition to **Sense thermostat**
  - Transition to **Supply heat**

- **Powered on**
  - Transition to **Idle**

- **Powered off**
  - Transition to **Idle**
UML State Machines

Inactive

Installed

Uninstalled

Idle

Sense thermostat
All others off

Call for heat / Activate burner

Powered on

Powered off

Heating

Sense thermostat
Burn gas
Run blower

No call for heat

[Warming up]

Sense thermostat
Burn gas

No call for heat

[Burner temp > set point]

Powered on

Powered off

Powered off

Powered on
**UML State Machines**

- **Inactive**
  - Installed
  - Uninstalled

- **Idle**
  - Sense thermostat
  - All others off
  - Powered on
  - Call for heat / Activate burner

- **Heating**
  - Sense thermostat
  - Burn gas
  - Run blower

- **Warming up**
  - Sense thermostat
  - Burn gas
  - Powered off

- Transition conditions:
  - [Burner temp > set point]
Using Heirarchy

Inactive

- Installed
- Uninstalled

Powered on

Active

Idle

- Sense thermostat
- All others off

Warming up

- Call for heat / Activate burner
- Sense thermostat
- Burn gas
- [Burner temp > set point]

Heating

- Sense thermostat
- Burn gas
- Run blower

Powered off

No call for heat
How Hierarchy Works

1. What is the first state?
2. What happens if you then get a D?
3. Alternately what happens if you get an X?
4. Suppose D, A, B, A, B, C: what state are you in?
5. If in state Bar and you get an X, what happens?

Make sure that you understand entry/exit chains too.
**Concurrent Regions**

Diagram:
- **Search for Bluetooth Devices**
  - Found Device
  - Connection Timeout
- **Issue PAN Connect**
  - Connected
  - No IP Yet / Wait 1 sec
- **Get IP Address**
  - Non-local IP Found
- **Issue Broadcast of my IP**
  - Received Remote IP / READY!!
- **Receive PAN Connection**
  - Connected
  - No IP Yet / Wait 1 sec
- **Get IP Address**
  - Non-local IP Found
- **Wait for Broadcast**
  - Received Remote IP / Send My IP, I'm now READY!!
State Machine Diagrams - Summary

- Based on Harel Statecharts
- Often used to describe a single object

Initial State

Final State

History State

State

State with Internal Events
entry/do something on entry
exit/do a different thing on exit
do/basic task to accomplish

Trigger [guard] / activity

Transition

Foobar

State with Internal Events

A1

AAA

ZZZ

B1

BBB

CCC

YYY

B2

A2

WWW

H

H

H

The University of Utah
Network communications in XNA

- Multiplayer support has been added to XNA
  - Create a network session object
    » We will use local networks, but you could develop your game to work through the entire XBox Live community
  - Transition through various connection states
    » You are responsible for drawing all lobbies
    » See the online tutorials – plenty of examples
  - Once all players are connected, start the game.
    » Players send and receive data packets to update game state.
Network communications in XNA

- Data sends are primitive
  - You specify a player (called a Gamer) to send data to.
  - You construct your data block, either:
    » An array of bytes
    » A ‘packet’ of combined primitives
      - Use (and reuse) a packet writer
      - Most basic data types and XNA vectors and matrices
  - You issue the ‘send’
  - You hope it gets there
Network communications in XNA

- Data receives are equally primitive
  - You specify a player (called a Gamer) to check for data from. If there is data:
    - You read the available data block from the player.
    - You decode it in the same way it is encoded
      » Differentiating packet types is your job.
      » Read the data out in the same order it was written in.
Networked game protocol

- XNA provides several types of in-game message delivery reliability:
  - No guarantee
  - Reliable
  - In order
  - Chat
Abstracting the game state is important!

- Do you want to send the entire game state between players every frame?
  » No.
- Can you only send game state transitions between players?
  » Yes, but either unreliably or untimely.
  » This requires well-defined game state transitions, and advancing frames and user input are usually sufficient as a game state transition events.
Networked game protocol

- What is the right way to send data between players?
  - What should be sent?
  - How should this sent data be applied to the advancement of the game?
  - How should errors be resolved?
  - What are we not considering?
Your next assignment:

- Design a protocol for handling lobby and in-game communications that:
  » Provides no appearance of lag wherever possible
  » Keeps players’ game states in sync (whether in the lobby or in the game)
  » Allows players to join / drop in the lobby or drop in the game without serious side effects
Your next assignment:

- I will provide you with a game where each player uses the keyboard and mouse to control each action.

- You will not get the game or any code or design specs prior to designing a network protocol for the game.

- You will need information, though. You are required to ask in class.
Networked game protocol

- Your next assignment:
  - Address all relevant issues surrounding synchronization of state.
    - What happens if data packets are lost?
    - What happens if data packets are delayed?
    - How will you recover from these errors?
  
    - What data will be sent, etc.
  
    - How will the networked data interface with the game?
  
    - How will the lobby work?
Networked game protocol

Your next assignment:
- Use a formal representation for your solution as appropriate:
  » UML state charts
  » UML activity diagrams
  » Other – TBD
  » No code, loose prose, handwritten work, etc.

Note: We have not yet talked about software process. I will use your experiences with this project to motivate their need!
**Networked game protocol**

- **The end goal:**
  - We want a network protocol that each team can program individually, but that will work together when executed (because they implement the same protocol).
    > Sufficient detail is needed!
  
  - Students will review the protocols to ‘elect’ a winning protocol. Winning team receives bonus points.

  - You will then program the protocol on top of the game I provide.

  - Note – This is not about gaming! It’s about synchronized state machines and networking.
Networked game protocol

- **Expectations:**
  - Students will need to dig through the XNA API and tutorials to discover, um, ‘features’ about their networking libraries.
    » You may want to explore with some test code.
  - Students will work together in the design
    » We will talk about teamwork in future lectures.