UML Sequence Diagrams
(Slides adapted from Michael Mateas)
Upcoming deadlines

- **Friday (Jan. 18): team status reporting**
  - Due by midnight
  - Report on team activities this week
  - Be sure to use team status reporting template
    - courses.soe.ucsc.edu/courses/cmps171/Winter12/01/pages/teamstatus-template
  - Don’t worry about making up for last week

- **Friday (Jan. 18): Technical design document due**
  - Be prepared to discuss in team meetings with Prof. the following week

- **Thursday, January 31: End of Sprint 1**
  - 18 days left in Sprint 1 (including a holiday, Jan 21)
Upcoming events

- Winter Job & Internship Fair
  - Tuesday, February 5
  - Resume & cover letter workshop: Jan. 29, 2-3:30pm
    - See event calendar at Career Center
  - See http://careers.ucsc.edu/ for more information
Technical Design Document

- Details also on web
- Required elements
  - Title section
  - UML structure diagram for your design
    - Printout of 1 or more pages
  - UML sequence diagrams for your design
    - If creating your own game engine, must include sequences for:
      - Initialization, menu system, main game loop, player collision, enemy collision, and end of level
    - If using an existing game engine, must include sequences that show how your code is called from the game engine
      - Less clear to me which sequences are most important here, depends on the game engine. In general want to represent interesting and/or complex sequences
  - Also: need to be prepared to discuss this in weekly meetings with Professor week of Jan. 21
Lab cleanup

- Need teams to sign up for week-long duty to tidy the lab
  - 9 weeks left in Winter
  - We have 9 teams!
  - Each team signs up for one week
  - Signup sheet up front

- Every individual is responsible for cleaning up after themselves
  - Especially food containers, napkins, bottles, etc.

- But, sometimes people forget

- Team should make a task for cleaning, put it on scrum board, assign to a person

- Team on lab cleanup needs to:
  - Ensure overflowing trash cans are emptied to bin outside in 3rd floor courtyard (anytime during week)
  - By 5pm Monday and 5pm Friday (unless things get out of control, the more often):
    - Pick up food containers, bottles, etc.
    - Pick up stray craft materials, pens, etc. and return to drawers
    - Clean off tables in conference rooms and big circular table
    - Report any major soda/food spills to me, so we can call cleanup crews
    - Put controllers/game boxes/etc. away (tidy up game area)
    - Report any cleaning materials needed
Cameron Alston

- Use of team reporting tool
  - [http://cmps172.soe.ucsc.edu/](http://cmps172.soe.ucsc.edu/)

- How to log in
  - Register using Google credentials (@ucsc or @soe email address)
  - Then, click on your team name
    - Please, please, please, get this right the first time
  - Will fix issue around reading the comments you submit
  - Will see other members of the team
  - Scrum Master should make sure everyone on team is signed up
Team Status Reporting

- **Stars:**
  - A number of stars indicating your perception of that person's total individual contribution over the week

- **Like:**
  - This is a list of things that person did during the past week that you liked.

- **Improve:**
  - This is a list of things that person did during the past that could use improvement.

- **ScrumMaster (ONLY):**

  - **Stars:** As above, relative effort for each team member.

  - **Performance assessment:**
    - A subjective assessment of the performance of the team member over the past week. This should include an overall assessment, such as, "exceptional performance", "performance meets expectations (this is normal)", "performance does not meet expectations". For very good and very poor performance, a description of why the performance was good/bad need to be provided ("did twice as much work as any other team member," "personally saved the sprint with their fast coding," "missed all project meetings this week and did not perform any tasks").
Introduction to UML

- The Unified Modeling Language (UML) consists of a collection of diagrams for describing a software design.

- Creating a UML description forces a team to develop a software design before diving into the nitty-gritty of writing code.
UML Sequence diagrams

- Sequence diagrams capture the temporal order of interactions between system objects (might be literal code objects or subsystems)

- You should capture a sequence diagram for each of the important chains of events that happens in your game
  - Collision detection
  - Controller event (player pressing a button)
  - Main loop
  - NPC action selection (if there are significant inter-object interactions)
  - Interface interactions
Lifelines

- Lifelines represent object instances or roles
  - The box at the top of the lifeline names the object instance or role
  - A dotted line under the box indicates how long the object lives
Messages

- Messages indicate method invocations between objects
- Bars on the lifeline indicate the period of time during which execution/handling of the message takes place
- Dotted lines indicate return
  - Returns are optional, though it’s recommended to use them if you are returning a value
Instances may create new instances

- If you need to show one instance creating another, use the <<create>> message stereotype
Self messages and call stacks

- Objects/roles can send messages to themselves
  - If working with object instances, this represents a method on an instance invoking other methods on the same instance
- Bars are nested to indicate the call stack

```
Main
<<create>>
1 : new CollisionManager()
2 : c:CollisionManager
3 : updateCollisions()
4 : handleTerrainCollisions()
5 : handleHeroCollisions()
6 : handleHeroObjectCollisions()
```
Indicating if-then semantics on individual messages

- Guards are used to indicate if-then semantics on individual messages
  - The message is sent only if the test in square braces is true

```plaintext
c : CollisionManager
char : Character
hero : Character
5 : handleHeroCollisions()
6 : checkCollision()
7 : collision
8 [collision] : isAttacking()
9 : handleHeroObjectCollisions()
```

Multi-instance
Combined fragments

- Combined fragments frame a subset of object interactions
- They are used to show that a subsequence has alternatives (if then blocks) or loops
- Here’s a loop example

```c
[foreach character] loop
c : CollisionManager
char : Character hero : Character
5 : handleHeroCollisions()
6 : checkCollision()
7 : collision
8 [collision] : isAttacking()
9 : handleHeroObjectCollisions()
check hero attacking
```
Alt fragments

- Alt fragments indicate if-then blocks
- Use interaction operands to indicate alternative sequences

```c
foreach character

8 [collision] : isAttacking()

alt Test Invincible
[invincible]

9 : doDamage()
10 : kill()
11 : damage()
```

```
c : CollisionManager
char : Character
hero : Character
```
Miscellaneous features

- All the message types used in the example are blocking message types (normal method invocations)
- Non-blocking messages (e.g. sending a request via an IPC mechanism like sockets) are indicated with open arrowhead
  - May need to use annotation if UML tool doesn’t support this
- Parallel processes can be indicated with a *par* combined fragment
Modeling Issues
Modeling issues

- A major UML modeling divergence is whether your game:
  - Creates its own game engine
    - I.e., using some game library like XNA that handles only a small part of game needs (scene graph, collision, pathing, NPC AI, etc.)
  - Uses an existing game engine

- Create own game engine
  - Relatively few connections to underlying game framework
  - Majority of code running the game is created by the team
  - Will have classes for gathering input, managing scenes (object lists), collision, AI, animation, scrolling, etc.
  - Team has large degree of control over the structure of software
Modeling issues (cont’d)

- **Using existing game engine**
  - Varies by engine
  - Typically takes the form of creating subclasses of existing classes of game engine framework
  - Can also involve creating code using scripting language inside game engine
  - UML diagram tends to look like a series of mostly disconnected classes (since connections occur inside game engine)
  - Important to show which game engine classes are subclasses off of
  - Need to understand clearly responsibilities handled by engine, and by code your team writes
Refresher: Strategy Pattern
Problem: Changing AI Behavior

- Consider:
  - AI behavior of an opponent often changes while the game is running
    - If it gets close to the player, or some other game event occurs

- How can this be accomplished in code?
  - Do not want to destroy opponent object, and create new one with changed AI behavior
    - I.e., creating a separate subtype of Opponent for each separate opponent behavior isn’t dynamic enough
  - However, also do not want to encode all possible behaviors inside each type of opponent
    - Ideally want to re-use AI behaviors across many types of opponent
    - I.e., putting a big switch/case statement inside each Opponent type won’t work either
      - “Switch statement” and “duplicate code” bad code smells
Strategy Pattern

- Client creates instance of IStrategy subclass
  - myStrategy = new IStrategySubclass();
  - Or, can be given subclass instance in constructor
- Inside the client, write code that relates only to IStrategy
  - myStrategy.Algorithm();
  - Will call the Algorithm method on subclass currently assigned to myStrategy

Show example of Strategy pattern in Visual C#
Design Principles

- Two design principles at play here
  - Favor composition over inheritance
    - More flexible to compose IStrategy subclass with Client than to make lots of Client subclasses
  - Program to Interfaces, not implementations
    - If you program to an interface, are not tied to a specific class that implements the interface
    - Can easily create another implementation of the interface, and use that instead
      - If you program to an interface, substituting a new subclass of that interface is a small change