Course Overview

Review of Scrum. Introduction to UML.

This lecture is based on two SCRUM presentations:
Agile Software Development with SCRUM by Shveta Mehtani (http://www.scribd.com/doc/6578688/SCRUMAEG)
What is Scrum? by Richard Fennell
(http://www.slideshare.net/businessquests/black-marble-introduction-to-scrum)
… as adapted by Michael Mateas
The year-long game design studio sequence

- **CS 170**
  - Exposure to a variety of alternative game designs
    - Indie, serious games, political games, art games, etc.
  - Individual concept development
  - Team formation and game design

- **CS 171**
  - The heart of making the game
  - Course is process-based, providing a series of milestones for completing game
  - Some game design work will continue
  - Focus on software engineering issues

- **CS 172**
  - Emergency design revisions (the “oh my god” moment)
  - Build out game content (level design)
  - Final playtesting and tuning
  - Finish game
  - Win awards at indie game competitions
Class mechanics

- Syllabus online at
  - courses.soe.ucsc.edu/courses/cmps171/Winter13/01
- Piazza
  - piazza.com/ucsc/winter2013/cmps171/home
- Team feedback tool
  - http://cmps172.soe.ucsc.edu/
Grading Criteria

- A combination of individual and group performance
- Individual assignment performance – 15%
  - 2-3 assignments during quarter (personal branding, level analysis)
- Individual (Sprint) project performance – 36%
  - Pre-sprint planning activities, release planning – 4%
  - Sprint I – 9%
  - Sprint II – 9%
  - Sprint III – 9%
  - Special team role performance – 5%
- Team project performance – 49%
  - Pre-sprint planning – 5%
  - Sprint I – 8%
  - Sprint II – 8%
  - Sprint III – 8%
  - Release performance – 20%
- Ungraded, but required to pass:
  - Game website
Team firing process

- A team member can be fired from a team for lack of performance, or poor interactions with the rest of the team (“bad apple” behavior)
- Detailed firing process is on course website
- Briefly, two step process:
  - Remediation
    - Identify and try to fix the problem.
    - Letter describes concrete steps and timeline for improved behavior
  - Removal
    - 2/3 vote of CS 171 members of team will fire team member
- Fired team member has two weeks to find a new team, or fails the class
Learning Goals

- Scrum software development process
  - Release and Sprint planning
  - Experience performing several Sprints and one Release
  - Project management using Scrum (burndown charts, task boards, daily scrums)
  - Hands-on experience in the Scrum Master role
  - "Bad apple" behavior and how it affects teams
  - Coordination with artists and musicians
- Software design
  - Unified modeling language (UML)
  - UML structure and sequence diagrams
  - Experience using UML to represent software designs
- Game testing
  - Game playtesting, concept and application
  - Gameplay metrics
- Software testing
  - Unit testing
  - Black box and white box testing
  - Different classes of code test coverage (all paths, def-use, etc.)
- Personal branding
  - Concept of a personal brand, development of personal brand
- Level design
  - Elements of level design, analysis of levels for their design
- Elementary typography
Upcoming deadlines

- **Wednesday (Jan. 9): Release Plan due**
  - A big effort. Start now.
  - Due by midnight
  - See website for information and template

- **Thursday (Jan. 10): Sprint 1 Plan due**
  - Due by midnight
  - See website for information and template

- **Wednesday (Jan. 9): Sprint 1 begins**

- **Friday (Jan. 11): team status reporting**
  - Due by midnight
  - Report on team activities this week

- **Friday (Jan. 18): Technical design document due**
  - UML diagrams describing the technical design of your game
  - A big task, need to spin 2-3 people up on this by end of the week
Sammy Awards

- Sammy Awards
  - Friday, June 14
  - Rio Theater (pending final confirmation)
Introducing Cameron Alston

- Ace TA for the class

- Details on how to submit assignments
  - Continue to submit using SVN, using same repository as last quarter

- Need to set up time to have Cameron be present during one scrum meeting each week
  - Part of Sprint 1 plan document
Artists, Musicians, etc. Independent Study

- Most artists should be taking companion class in Art Department
  - But, there might be some artists who are unable to take this class
  - Musicians may need to take an independent study, if they want credit
- Filling out independent study form
Photos

- On Monday, January 14, I will be taking photos of each team during class.
- Please invite your artists and musical team members to come to class that day, if they can, to be part of the team photo.
- Photos will be posted on the class website, along with your name.
  - Very useful for putting names to faces
  - Let me know if you do not want to be part of the team photo, and/or have your name online
Professor meetings

- I will be meeting with each team weekly, for up to 1 hour

- On Friday, Cameron will match teams to available timeslots.

- As preparation, your team needs to collect schedules from every team member, and find time blocks where most of the team can meet.

- I don’t expect that every team member will be able to make the meeting with me.
Review of Scrum
Release

- A **release** is a major milestone in the development of a software project
- A release contains a series of product **features**
- Features are expressed in the form of **user stories**

- The goal of **release planning** is to determine which user stories (features) will be included. This involves:
  - Taking the game concept and **decomposing** it into user stories
  - **Estimating** the time required to perform each user story (using story points)
  - **Prioritizing** the user stories
  - The release plan forms the input into the Sprint planning process
User stories

- A product feature is expressed in the form of a user story.
  - This can be viewed as a specific technique for eliciting and writing software requirements.
  - A user story is a software requirement

- User story format
  - As a {user role}, I want {goal} [so that {reason}]
  - Examples:
    - As a player, I need control over a laser pointer so that the cat will follow it.
    - As a player, I need to pick up gameworld objects so that I can collect food and ammunition.
    - As a playtest manager, I need automated collection of gameplay metrics so that levels can be analyzed for areas that are too difficult.

- Class exercise developing a few user stories for your game
Estimating size of user stories

- The relative size (implementation effort) of each user story is estimated using measure known as story points.

**Story points are unitless**

- Are not person-months, meters, hours, etc.
- Key idea is to focus estimating effort on *relative* size
- Use of unitless numbers avoids arguments
  - “That won’t take a week to implement – that’s easily a week and two days”
  - … but the point is trying to determine which tasks are $O(\text{days})$, $O(\text{weeks})$, and $O(\text{months})$ – +/- a few days doesn’t matter!

**Story points are linear**

- A user story requiring 0.5 story points takes half the time to complete as one requiring 1 story point
- Similarly, a user story requiring 3 story points is the same size as one requiring 1 story point and another requiring 2 story points
Output of Release Planning

- At the end of release planning:
  - A prioritized list of user stories, with implementation time estimated in story points, organized into Sprints.

<table>
<thead>
<tr>
<th>Plan for Release #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td><strong>Sprint 1</strong></td>
</tr>
<tr>
<td>1.</td>
</tr>
<tr>
<td>2.</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td><strong>Sprint 2</strong></td>
</tr>
<tr>
<td>N.</td>
</tr>
<tr>
<td>N+1.</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Sprint planning

- Team re-evaluates user stories from the release plan and product backlog they can commit to completing
- Sprint backlog is created
  - User stories are subdivided into tasks
  - Tasks are identified and each is estimated (~8 hours)
  - Collaboratively, not done alone by the ScrumMaster
- High-level design is considered

As a vacation planner, I want to see photos of the hotels so I can have a better idea of facilities

Priority 4 [10 Story Points]

Code the middle tier (8 hours)
Code the user interface (4)
Write test fixtures (4)
Code the foo class (6)
Update performance tests (4)
Sprint planning (2)

- Task estimation
  - Performed as a group, using Planning Poker
  - Here, units of estimation are “ideal work hours”
    - The amount of work you can get done under ideal conditions
      - Full knowledge, no interruptions
    - Actual hours elapsed will be greater than ideal hours
  - Task estimates are a **commitment** to accomplish a development task in a certain period of time

- How many ideal work hours can each person perform?
  - Good question – so far, your group has no track record on this
  - For now, pick a conservative figure, such as 10-12 ideal hours/week
  - So, each group member can do 30-36 ideal hours of work per Sprint
Output of Sprint planning (for CS 171)

- Task listing (with time estimate), organized by user story (prioritized)
  - User story 1:
    - Task 1 (time estimate)
    - Task 2 (time estimate)
    - ...
  - User story 2:
    - Task 1 (time estimate)
    - Task 2 (time estimate)
    - ...
- Team roles
  - Team member 1: role
  - Team member 2: role
    - ...
- Initial task assignments
  - For each person, what is the first task they are working on?
- Initial task burndown chart
- Initial scrum board set up
- Schedule of Scrum meetings
  - When/where for 3 weekly face-to-face scrum meetings
Introduction to UML
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.
Class diagrams

- A class diagram describes static aspects of your object oriented design.

- Classes are drawn as boxes.

- Members are listed inside the box. Fields appear in the top sub-box, methods in the bottom sub-box.
  - Access indicated by + (public), - (private), # (protected) and ~ (package).

- Classes are connected together with lines indicating class relationships.

```
SequentialBehavior
- stepCounter: int
- child: Step
~ getNextStep(): Step
~ addChild(s: Step)
~ removeChild()
```
Generalization links

- Generalization links indicate subclass relationships
  - Parent/child relationships
- An open arrow points to the parent

**Behavior**

- `#parent: GoalStep`
- `#priority: int`
- `#signature: String`

- `~failBehavior()`
- `~succeedBehavior()`

**SequentialBehavior**

- `#stepCounter: int`
- `#child: Step`

- `~getNextStep(): Step`
- `~addChild(s: Step)`
- `~removeChild()`
Aggregation links

- Aggregation indicates that instances of one class will contain instances of another class
  - In aggregation, the lifespan of the enclosed instances is independent of the lifespan of the enclosing instance

- Container classes (lists, hashtables, etc.) will always have aggregation links to what they contain, though many classes will contain member instances of other classes
Composition links

- Composition links indicate that one class contains instances of another class, but the contained class is created and destroyed with the instance class
  - The contained instances will be destroyed when the containing instance is destroyed

- In C++, this is the difference between a member variable of type MyClass* and MyClass
Realization

- Realization links relates a class that implements (realizes) a behavior specified by another model element, to the model element that specifies this behavior.

- In Java, classes that implement an interface realize the interface.
- In C++, classes that are children of a pure abstract class realize behavior specified by the pure abstract class.

```
Realization

Realization links relates a class that implements (realizes) a behavior specified by another model element, to the model element that specifies this behavior.

In Java, classes that implement an interface realize the interface.
In C++, classes that are children of a pure abstract class realize behavior specified by the pure abstract class.
```
Dependency links

- Dependency links represent arbitrary relationship between classes, where a change made to one class may require a change to another class
  - The arrow points from the dependent towards the independent class

- You’ll want to use link labels for dependency links

- On the class diagram, only indicate important dependency relationships (ones that help communicate in the team)
UML Tools

- There are many good, free UML tools
- Some that seem interesting:
  - Gliffy (http://www.gliffy.com/) - web based
  - Creately (http://creately.com/) – web based
  - UMLet (http://umlet.com/)
  - Dia (https://live.gnome.org/Dia)