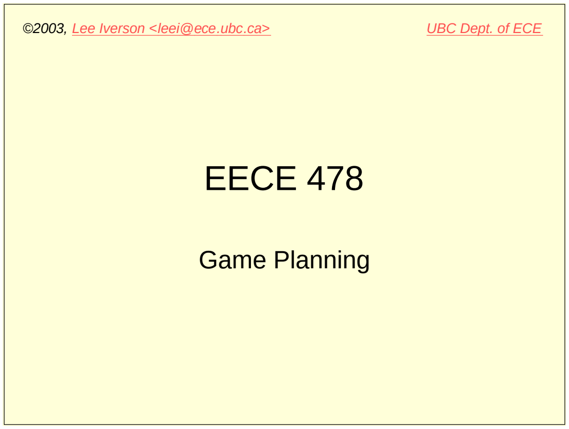
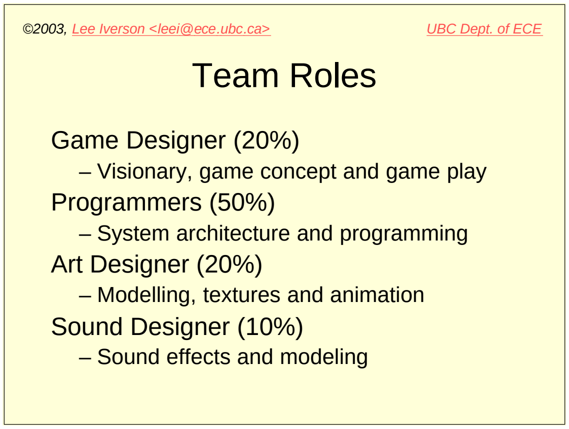


EECE 478

Game Planning



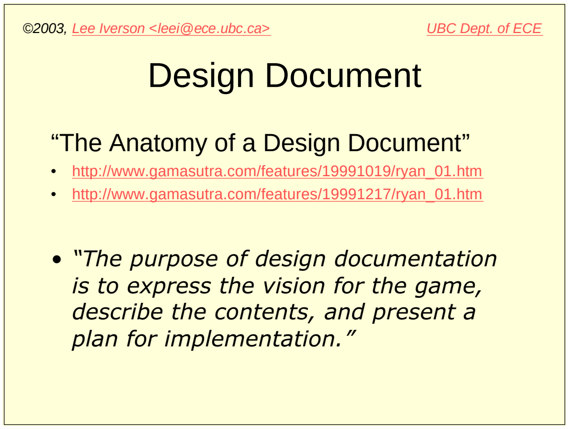
Team Roles

Game Designer (20%)
- Visionary, game concept and game play

Programmers (50%)
- System architecture and programming

Art Designer (20%)
- Modelling, textures and animation

Sound Designer (10%)
- Sound effects and modeling



Design Document

“The Anatomy of a Design Document”
- http://www.gamasutra.com/features/19991019/ryan_01.htm
- http://www.gamasutra.com/features/19991217/ryan_01.htm

- *“The purpose of design documentation is to express the vision for the game, describe the contents, and present a plan for implementation.”*

Other Documents

(From Stanford CS248)

Storyboard
- "Comicbook" of your game in action – this is a tool, not an art piece focus on important screens (start, end, game over, win!, etc.), use of graphics advanced features, and interaction

Task list
- List of work items, **priorities**, time estimates and owners
- Priorities should be: Must Have, Priority 1, Priority 2, CUT

Schedule
- High level calendar when/what should be done, constraints your teammates may have

Content/Artwork Map
- List of 3D models, textures and images you think you need and where you will get them from

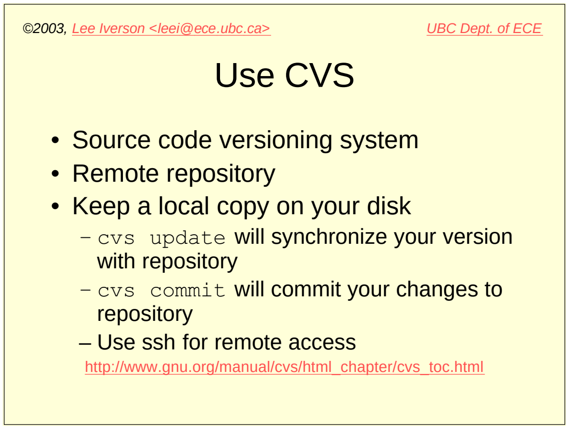
http://www.gamasutra.com/features/20020903/london_01.htm

Task Priorities

1. Basic visual mockup
   - Get anything on screen with some bare-bones interaction
2. A working demo
   - Minimal visual "style" but nearly full interactivity and game play
3. Full first version
   - Complete world, basic graphics and sound
4. Completion
   - Details. Higher quality. Optional features.

Incremental Development

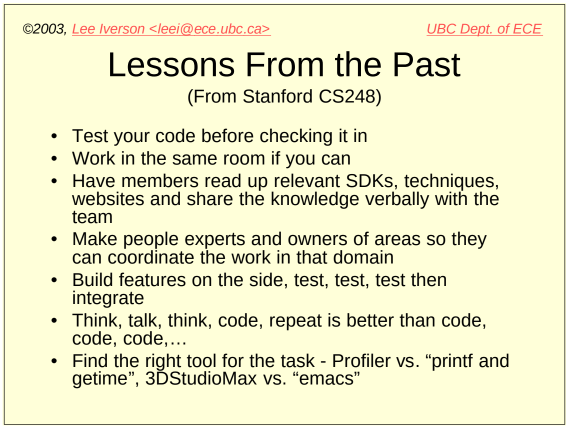
- Always maintain a *working* version
  - Have a clear definition of *working!*
  - Test constantly
  - *Never* reduce playability of game
- Modularize high-risk development
  - Branch for features that take some time to complete
  - Don't merge with mainline until *all working*
  - Avoid committing entire team to branch



Use CVS

- Source code versioning system
- Remote repository
- Keep a local copy on your disk
  - `cvs update` will synchronize your version with repository
  - `cvs commit` will commit your changes to repository
  - Use ssh for remote access

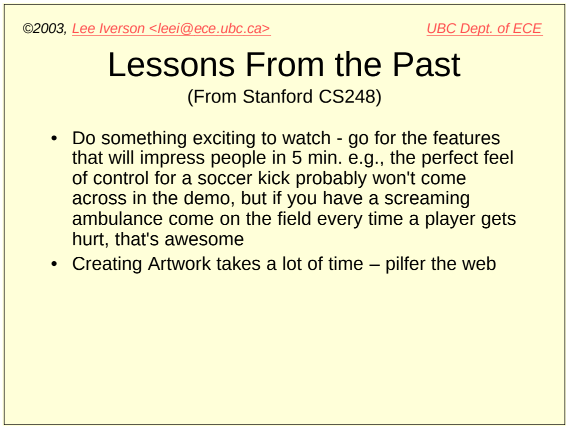
http://www.gnu.org/manual/cvs/html_chapter/cvs_toc.html



Lessons From the Past

(From Stanford CS248)

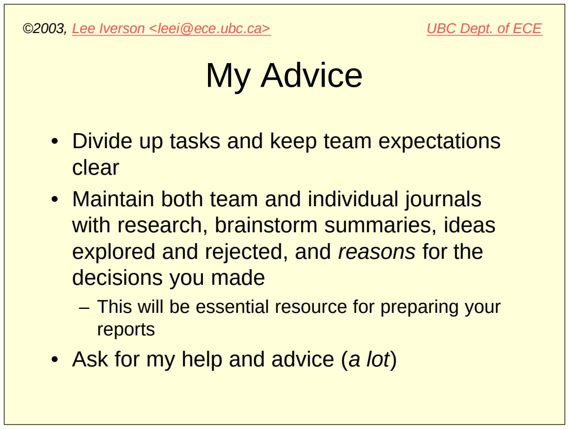
- Test your code before checking it in
- Work in the same room if you can
- Have members read up relevant SDKs, techniques, websites and share the knowledge verbally with the team
- Make people experts and owners of areas so they can coordinate the work in that domain
- Build features on the side, test, test, test then integrate
- Think, talk, think, code, repeat is better than code, code, code,…
- Find the right tool for the task - Profiler vs. “printf and getime”, 3DStudioMax vs. “emacs”



Lessons From the Past

(From Stanford CS248)

- Do something exciting to watch - go for the features that will impress people in 5 min. e.g., the perfect feel of control for a soccer kick probably won't come across in the demo, but if you have a screaming ambulance come on the field every time a player gets hurt, that's awesome
- Creating Artwork takes a lot of time – pilfer the web



My Advice

- Divide up tasks and keep team expectations clear
- Maintain both team and individual journals with research, brainstorm summaries, ideas explored and rejected, and *reasons* for the decisions you made
  - This will be essential resource for preparing your reports
- Ask for my help and advice (*a lot*)