Introductions...

► Name
► Group number
► Favorite videogame
About Me

► Kristen Kho
  - 5th year undergrad, starting MS this fall
  - AIM: silverdart21
  - kmkho@ucsd.edu

► Spring 2006, Group 4
  - Graphics person
    - Models, animation, cel-shading

► Experimental Game Lab (Calit2)
  - Libraries for the Cell Processor

► CSE 167, 169, 168 (this quarter)
Kablooey!

- 6 members

- Sorta like bomberman/smash brothers...
  - Throw items
  - Simple power-ups

- Cartoony, cel-shaded characters
  - Wacky animations
  - Funny voice-overs

- Fun to play and had fun making it
Where do we start?

► Decide on a game idea that everyone is excited about
  ▪ Gameplay, style, etc.
  ▪ Be creative and have fun with it!

► Choose a language that everyone is comfortable using (C/C++/Java)

► Decide on which libraries to use (may need to do some research beforehand)
  ▪ DirectX
  ▪ OpenGL
  ▪ SDL (Simple Directmedia Layer)
  ▪ etc.
More Advice

► Assign tasks to each person (may overlap and change over the quarter)
  - Don’t want anyone to be idle
► Set up deadlines/benchmarks
  - Integrate early: can find and fix any major bugs
  - Don’t wait until the last 2 weeks to actually play your game
► Start with existing code
  - Web tutorials
  - Past projects (CSE 167, CSE 169)
  - DirectX samples
Keep in Mind…

► Your goal is a 15-minute demo
► You have less than 10 weeks to create a fully functional game
  ▪ Be realistic: you may not have time to implement everything, so prioritize features (must-have, would-be-nice, only-if-we-have-time)
Team Dynamics

► Get to know your team members
  ▪ Go out to eat
  ▪ Have a lan party, play Smash Brothers, etc.

► Enforce good communication
  ▪ Meet up often
  ▪ Work together (try to XP if you can)
  ▪ AIM and webboards are not a substitute!
Kablooey! In Retrospect...

► Overall design was good
  - Everyone contributed ideas and agreed on concept

► Priority/schedule conflicts
  - Some team members too busy with work from other classes

► Always behind schedule
  - Not enough time for testing (very bad!)
  - Technical difficulties during demo

► Don’t let this happen to you!
Intra-Group Relationships

► Work together
  - Groups often encounter the same problems
  - Share solutions (but not code)

► Last year...
  - Todd (modelling expert)
  - Drugs... I mean, candy
  - And, of course, the “Bear”

► I will make the model available on pisa
NightSky (Group 1)
Petals of Pandora (Group 2)

I never want to love again.
UCSDOOM (Group 3)

Player was killed by mrr
mrr was killed by Goniff
mrr was killed by Goniff
Kablooey! (Group 4)
DirectX vs. OpenGL

- DirectX is a collection of libraries
  - Each deals with a particular component of your game
  - DirectX 9 SDK Components
    - Direct3D
    - DirectSound
    - DirectInput
    - DirectPlay & DirectShow are now deprecated

- OpenGL is just a graphics library
  - GLUT adds some window management features
  - SDL adds more DirectX-like features
  - Can be used with DirectX components
Graphics

► OpenGL
- You probably have more experience working with it
- Can be faster if you know what you’re doing

► Direct3D
- Has a learning curve for new users
- Lots of cool features (HLSL, HDR…)
Audio

► Joey Hammer is an audio nut (but don’t bother him too much)

► Some libraries…
  - DirectSound
  - SDL
  - OpenAL
  - FMOD

► Suggestion: create your own voices/sound effects
  - Audacity
  - ACI Dxpress
Input

► For basic input (keyboard and mouse) you can use Win32 events
  ▪ Might be difficult to write a keyboard map
  ▪ Mouse is a little clunky

► For more advanced input (joysticks, gamepads, steering wheels) it’s easier to use a library to manage devices for you
  ▪ DirectInput
  ▪ SDL
Networking

► Decide on a model
  - Client/server
    ▶ Server holds global game state, receives input from clients
  - Peer-to-peer (P2P)
    ▶ Clients communicate directly
  - Either way, you’ll need a way for players to find other players
    ▶ Lobbies/meeting areas
    ▶ Specify server ip address

► Libraries
  - **Winsock** – easiest to use (according to past groups)
  - **OpenTNL**
  - **Raknet**
Art and Models

► Design your own
  ▪ 3DS Max, Maya, Milkshape, etc.
  ▪ Photoshop or photos for textures

► Find freebies
  ▪ Google.com is probably the best way
  ▪ Forums (gamedev.net)

► Note: Direct3D supports .x files...
  ▪ use a file converter
    ▪ Deep Exploration, Panda DirectX Exporter
  ▪ write your own code to load specific 3D format
    ▪ see MeshFromObj sample in DirectX Sample Browser
Visual Studio

► Setting up VS can be tricky…
  ▪ Check out Joey’s Tutorial on the class webpage
  ▪ Will have in-class tutorial next week…

► Set compiler to warning level 4 (/w4)
  ▪ Fix as many warnings as possible

► Debug and Release modes
  ▪ Don’t wait til last minute to compile Release
Directory Structure

► Root directory (i.e. “gdevroot”)
  - Main .sln file
  - Assets folder (audio files, models/textures, images)
  - Module folders (audio, graphics, input, networking, physics, etc.)
    - Source code
      - .vcproj project file (build major components separately and test them)
Version Control (CVS & SVN)

► Setup early
► Use often
  ▪ Don’t lose your work
  ▪ Save integration/merging time
► Update before committing changes
► Make sure your code works before you check it in
► When you do a check-in, let others know what you checked in (use the mailing-list)
C++ Programming Tips

► Code needs to be fast and efficient
  ▪ Avoid using the copy constructor (operator=)
    ▶ Use pass-by-reference-to-const instead of pass-by-value
    ▶ Use constructor initialization lists
  ▪ If possible, allocate everything before gameplay or use a fixed number of objects (especially for particles)
    ▶ Ring buffers
    ▶ Memory pools
► Pass references instead of pointers to avoid dereferencing bad/uninitialized addresses
C++ cont.

► Check return values from system/library calls

► For defining constants, use static const’s instead of #define’s (helps with debugging later on)

► Using the STL
  - Can save you some time
  - But overhead can affect performance
Memory Management

► No memory leaks!
  ▪ Don’t want to run out of memory during the game
  ▪ Remember: when dynamically allocating arrays with `new[]`, use the corresponding `delete[]` operator

► Can use Task Manager to check performance or implement a Memory Manager (see Llopis, chapter 7)

► But if your game can run for 15 minutes without crashing, it might be okay…
Minimize Compilation Dependencies

► For large projects, compilation times can be HUGE
  ▪ Change one file --> recompile every file that includes it

► Solution:
  ▪ Use class declarations instead of class definitions whenever possible
  ▪ When using only pointers/references, full definition is not needed
Debugging

► Use a Debug Console
  ▪ Only defined for Debug build
  ▪ Joey’s Tutorial/Code
    ▷ Cool features like printing in color, stalls, stream redirection, etc.

► Assert on unexpected situations, don’t code around them
  ▪ But be careful what you put in them…

► Step through code as it executes
Team Coding Considerations

- Have team agree on coding standards
  - Spaces or tabs?
  - Curly braces at end of line or next line?

- Have team agree on naming conventions
  - Prefix “g” for global vars, “m” for member, etc.

- Use a separate file for each class and put all your declarations in header files
  - Easier to find things, especially for large projects
Comments

- Good code should not require extensive commenting
  - Use long descriptive class/file/variable names
  - Enforce correctness with “const” keyword (use everywhere)
  - Asserts can be good comments

- Comment hacky things to let your team know where you made assumptions
Other Tools/Tips

► Use config (.ini) files
  ▪ Quick and easy way to change and test assets without having to recompile
  ▪ Win32 API has built-in functionality for reading ints and strings

► Use a physics library only if having “real physics” is essential to your game (“fake physics” works for most cases). Same for AI.

► Create/find a particle system
  ▪ This will make your game look cool…
Radioactive Monkey
Resources

► Group webpages from previous quarters
► Read the papers on the course Resources page
  ▪ good intro to game engines and C++ refresher
► DirectX samples and documentation are installed (or will be) on the lab computers
  ▪ Start->Programs->Microsoft DirectX SDK
► List of websites and books on class webpage
  ▪ “C++ For Game Programmers”, Noel Llopis (1st ed)
► Your TA 😊