Learning from Postmortems
(slides adapted from Michael Mateas)
Lab Update

- Ongoing issues
  - Accounts on machines
  - PS3 dev kit web access
  - More trash cans
    - Request is in, I need to follow up
  - Speakers for lab addition computers (wires?)
  - More games (LBP 2)
    - Not yet started
  - Any book requests
    - C++ Bjarne’s book
  - Others?
Upcoming deadlines

- **Tuesday (Feb. 1)**: Sprint 2 plan
  - User stories, broken into tasks, which have been estimated and prioritized
  - Also, updated Release 1 plan
- **Friday (Feb. 4)**: team status reporting
  - Due by midnight
  - Report on team activities this week
  - Be sure to use CS 171 team status reporting template
    - See http://www.soe.ucsc.edu/classes/cmps171/Winter11/team-status-report-template.html
    - Scrum Masters use different template from rest of team
- **Friday (Feb. 4)**: updated Sprint burndown charts
  - Ideally these are updated daily, so they’re useful for your team
- **Thursday (Feb. 10)** Web site skeleton
  - URL, design template, minimal content
- **Thursday (Feb. 10)** Game playtesting plan
  - More details to come
  - Who will conduct tests and write playtest report? How will you recruit testers? Where will you conduct playtests? What specific gameplay issues do you want to focus on in first few weeks of playtesting?
Postmortems

- Game postmortems are a regular feature in both Game Developer Magazine and Gamasutra

- Focus on 5 things that went right, 5 things that went wrong

- Reading postmortems helps you to recognize good and bad development patterns happening on your own game

- Doing your own project postmortems solidifies lessons for future projects
What went wrong: testing

- Not leaving enough time for testing – end up focusing on stability rather than gameplay details of levels and enemies

- Beta too late to have any real impact

- Insufficient communication between QA and development

- Didn’t take early advantage of automated testing

- Testing with the wrong audience
What went wrong: design

- Mistakes in level design
  - Not progressing game mechanics and obstacles along with the story
  - Not adequately melding design styles from multiple designers

- Design decisions (like temporal variety) that multiply content needs

- Balancing AI quantity (handling many units or characters) and quality (AI looking good when focus is on a single unit)

- Lack of specified design lead to divergent efforts during balancing and tweaks

- Lack of focus on controls

- Lack of high-level vision

- Overloading player with new concepts and systems
What went right: design

- **Prototyping**

- **For level-based game, focusing from the beginning on level design**

- **Early establishment of physical metrics for player character/world interactions**

- **Nailing the gameplay mechanic before content development**

- **Playtesting provides feedback on difficulty arc and gameplay and builds a fan community**

- **Sticking with 2D**

- **Physics-based gameplay**
What went wrong: project management

- Grossly underestimating the amount of work

- School projects versus full development

- Adding stuff at the last minute

- Didn’t investigate outsourcing companies sufficiently leading to bad experience

- Positive feedback leads to feature creep

- Crunch

- Inadequate analysis and planning for technical risk
What went right: project management

- **SCRUM**

- Internal art person had great art community connections, leading to good outsourcing experience

- Empowering creativity across the team
What went wrong: tools and tech

- Networking code started too late

- The lure of shared technology

- Loosing iterative design time due to refactoring

- Inefficient art pipeline

- Custom tech (driven by unique gameplay features) leading to inefficiencies (“transformation tech too expensive”)

- Physics problems
What went right: tools and tech

- Successfully building on tools from last project

- Custom engine supported experimentation and fast turnaround

- Using an existing, well-established engine allowed focus on design

- Good use of middleware