Games and Software

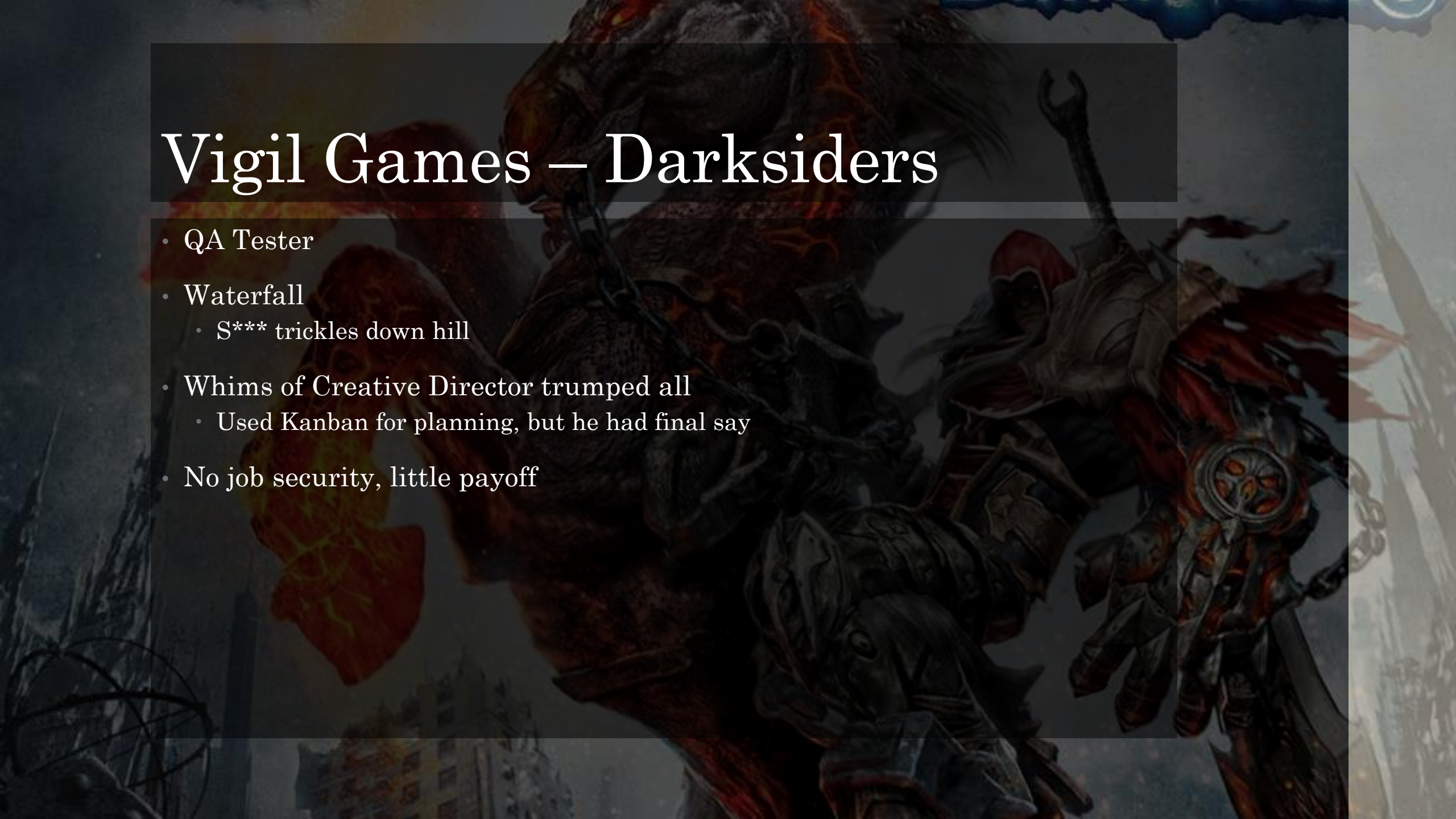
How what you learn here will actually be useful

Today's talk

- My background
- Tools I've used and continue to use
- Difficulties of timeline estimations
- Joys of planning and architecture

My background

- Graduated Dec. 2008 (bad luck)
  - Grades are important, skills and projects much more so
- Time in games industry
  - Vigil Games – Waterfall
  - Sony Online Entertainment – Agile/Scrum (Established)
  - GameSalad – Agile/Scrum (Startup)
- Microsoft
- Independent Consultant
- Univ. of Missouri research lab



Vigil Games – Darksiders

- QA Tester
- Waterfall
  - S*** trickles down hill
- Whims of Creative Director trumped all
  - Used Kanban for planning, but he had final say
- No job security, little payoff

SOE – DC Universe Online

- QA Tools Engineer and Designer
- Agile/Scrum
- Creative Director kept vision, but teams & producers set schedules
  - Kanban and sprints
- Constant iteration, stand-ups, support from all directions
- Low job security, little payoff

GameSalad – Scriptless Game Creation

- QA Engineer
- Agile/Scrum
- Software, not game, headed by business needs
- First time using Git (in-house repo)
- First time setting up CI
- Start-ups are fun, learn a lot, take on many roles
- Low job security, high payoff if success

Powered by GameSalad®

Microsoft

- Program Manager Intern
- Agile/Scrum
- Independence is expected
  - No hand holding after first few weeks
- Well oiled machine, but bureaucracy
  - Older company = established process
  - Older company = more politics
- High job security, high payoff

Independent Consulting

- Web app/site development
- Agile approach
- Responsible for everything, including customer interaction, finances
- Used everything I had learned, went well, hit deadlines
- Very fun, but stressful
- Constant grind, hard on family life
- Very low job security, high potential payoff

Univ. of Missouri

- Lead developer on educational game
- Waterfall-ish
- Semi-independent-ish?
- Lack of outside experience
  - Problems where solutions already exist
  - Cannot bring in industry experience
- Highest job security, but lowest payoff
  - Basically unfireable
  - No stocks, low pay, fall behind industry

Tools, now and then - Undergrad

- Version Control
  - Never heard of it, never taught
- Issue Tracking
  - Paper and pencil?
- Continuous Integration
  - Never heard of it
- Project Planning
  - Never taught, code myself into corners, hack together working submission

Tools, now and then - Older

- Version Control
  - Perforce – industry standard
  - SVN – predecessor open source
- Issue Tracking
  - TestTrack Pro – bulky, old, high learning curve
  - Microsoft TFS – bulky, old, high learning curve, modernizing
- Continuous Integration
  - Custom, if any
- Project Planning
  - Kanban Board – literally a board with index cards

Tools, now and then - Newer

- Version control
  - Git – high learning curve, open source
  - Perforce – industry standard (still)
- Issue Tracking
  - JIRA (Atlassian product)
  - Site-specific issue tracker (e.g. Gitlab tracker)
- Continuous Integration
  - Hudons/Jenkins
- Project Planning
  - Trello – Kanban board, but not physical!
  - JIRA

Project management – 80/20 rules

- Time: 80% planning, 20% building
  - But work in Sprints, of course
- 80% of time will be spent writing 20% of the code
- 80% of processing happens in 20% of your code
- 80% of your problems will come from 20% of your code
- Lots more…

Project management – Time estimates

- When starting, double, then double again
  - Less experienced → Estimated time x 4
  - More experienced → Estimated time x 2
  - Final goal → Estimated time x 1.5
- No one is immune
  - **Every** engineer overestimates their ability
  - Concepts are easy, implementation gets everyone
- Track expected vs actual time in class projects
  - Much less risky to test now than when working a job

Project management – Monetary Costs

- See – time estimates
- Highest $$$ cost of software is labor
  - More time = more $$$
- Quadruple expected budget before raising money
- Many startups run out of money
  - “We’re genius coders, we should be able to knock this out in 6 months!”
  - 2 years later they’re either broke, or in cubicles

Joys of Architecting

Modularity

- Modular projects = easier to plan
- Cutting a feature won't destroy project
  - Important for deadlines
- Adding features won't require rewrites
- Easy to put into 2 week sprints
- Fun to break down complex systems
  - Small, manageable chunks are best
- Allows code reuse between projects
- Easy to use in portfolios ;)

Joys of Architecting

80/20 (again)

- Time
  - 80% architecting
  - 20% coding
- Well laid plans = easy work
- No coding yourself into holes
- No coding other people into holes
- Coding is now on finding awesome solutions to difficult problems, not finding memory leaks or infinite loops

Joys of Architecting

Should be fun

- **<u>Your time to be creative</u>**
- Mix of engineer, artist, psychologist
- Get to experiment
- Leave your signature on a product
- How people see and use your product is determined during architecting

Joys of Architecting

Not just for leads

- Good company lets their workers work
- Everyone needs to know how to make a spec
- Everyone needs to know how to use a spec
- Everything modular = tons of specs
- Learn now = big payoff in the future

One last thing

- No matter your position, ***you are a contractor***
- Your skills are yours, and you're selling those services
- If you're good, companies will work to keep you
- If you're good, companies will fight to have you
- But *you* decide
- Never compromise your ideals for someone else's portfolio

Questions?