IMGD 1001:
Level Design

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

Outline

- Gameplay (done)
- Level Design (next)
- Game Balance

Your Projects: Selecting Features

- Note! First …
  - Work on core mechanics (movement, shooting, etc.)
  - Get bugs worked out, animations and movement smooth
- Then, have
  - prototype with solid core mechanics
  - tweaked some gameplay so can try out levels
- Need
  - 25 levels!
  - Rest of features!
- Problem … too many ideas!
  - If don't have enough, show it to some friends and they'll give you some

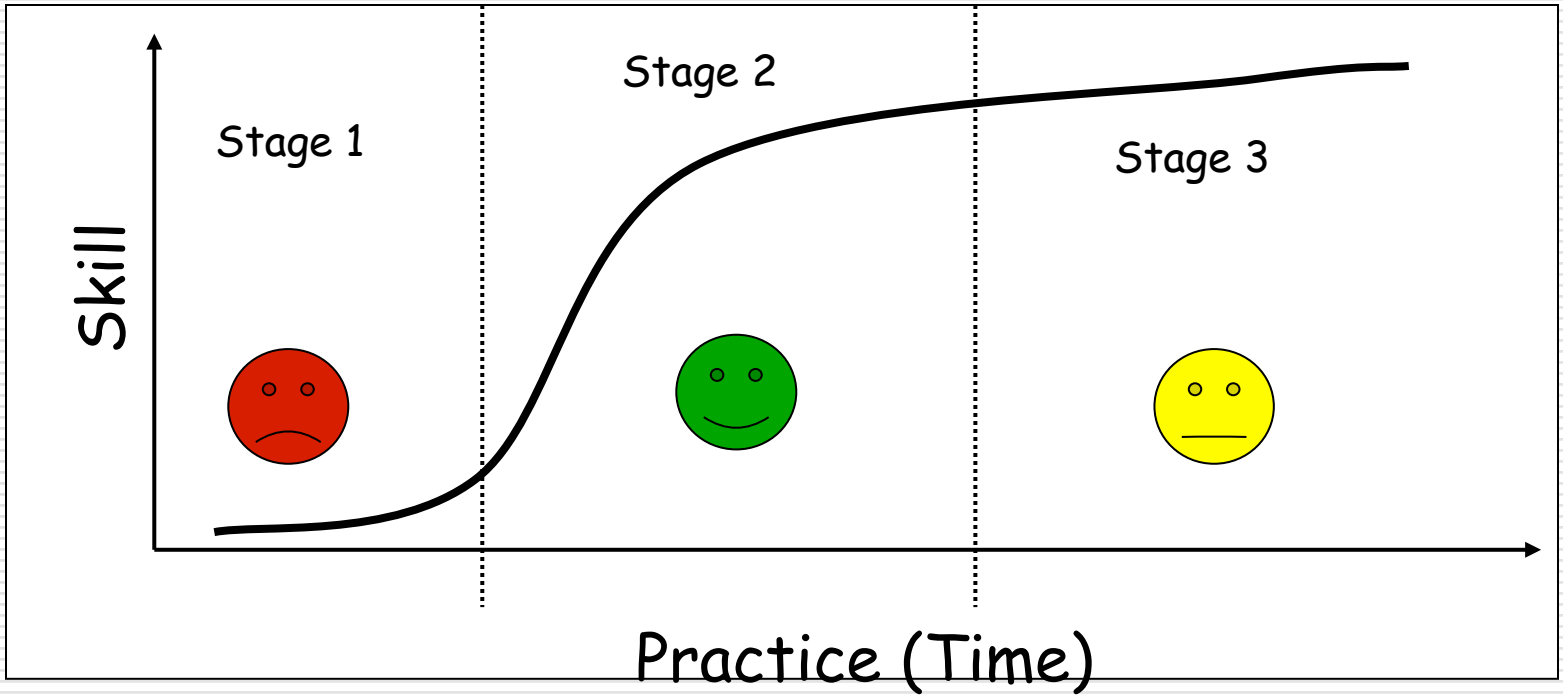
Your Projects: Types of Features

- Player can use
  - Abilities (attack moves, swimming, flying)
  - Equipment (weapons, armor, vehicles)
  - Characters (engineer, wizard, medic)
  - Buildings (garage, barracks, armory)
- Player must overcome
  - Opponents (with new abilities)
  - Obstacles (traps, puzzles, terrain)
  - Environments (battlefields, tracks, climate)
- Categorizing may help decide identity
  - Ex: Game may want many kinds of obstacles, or many characters. What is *core*?

Your Projects: Tips on Vetting

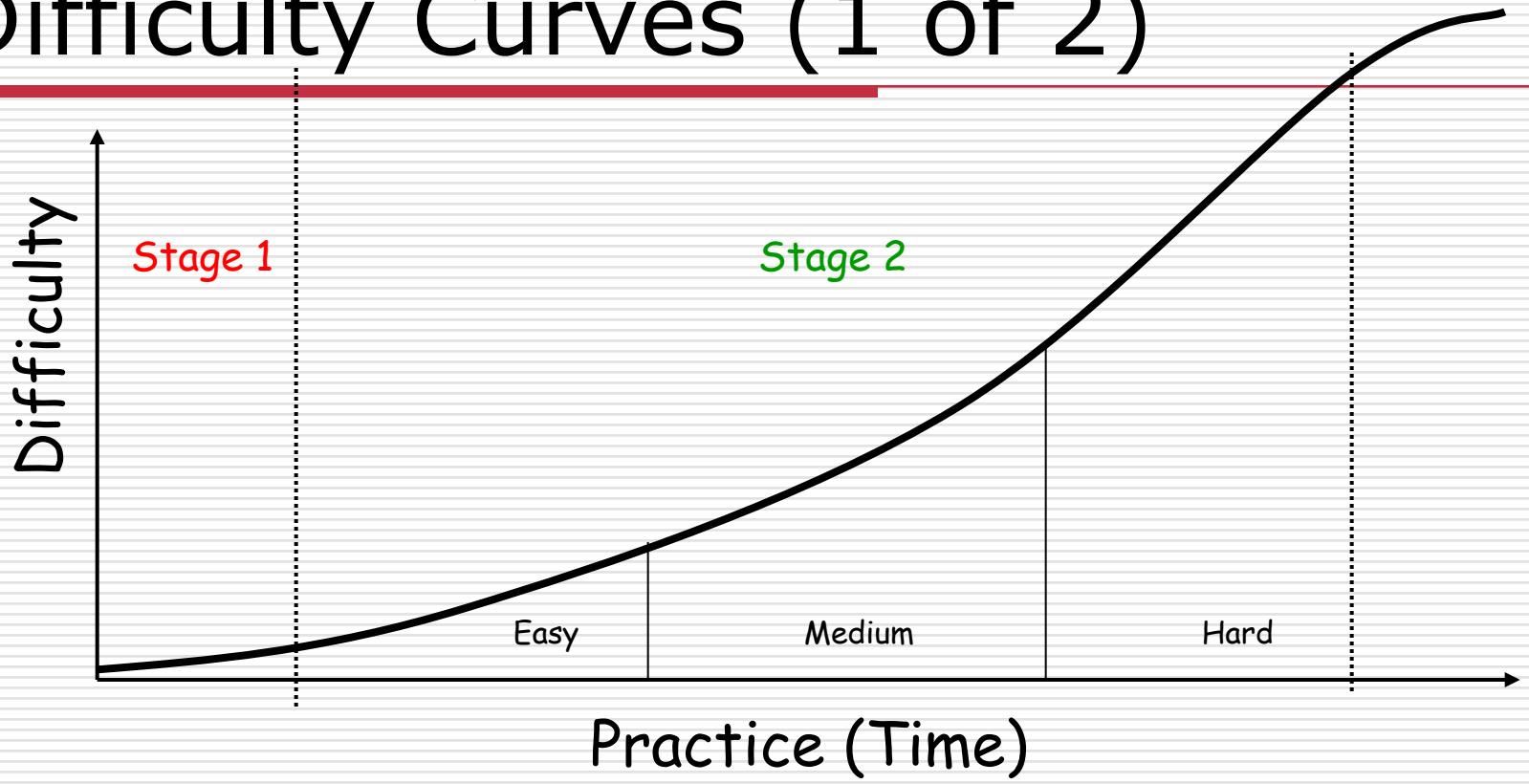
- Pie in the Sky
  - "The Koala picks up the jetpack and everything turns 3d and you fly through this customizable maze at 1000 m.p.h…"
  - Beware of features that are too much work
  - Don't always choose the easiest, but look (and think) before you leap
  - And don't always discard the craziest features … you may find they work out after all
- Starting an Arms Race
  - "Once the Koala's get their nuclear tank, nothing can hurt them. Sweet! No, wait …"
  - If you give player new ability (say tank) they'll like it fine at first
  - But subsequently, earlier challenges are too easy
  - You can't easily take it away next level
  - Need to worry about balance of subsequent levels
- One-Trick Ponies
  - "On this one level, the Koala gets swallowed by a giant and has to go through the intestines fighting bile and stuff…"
  - Beware of work on a feature, even if cool, that is only used once

Learning Curves



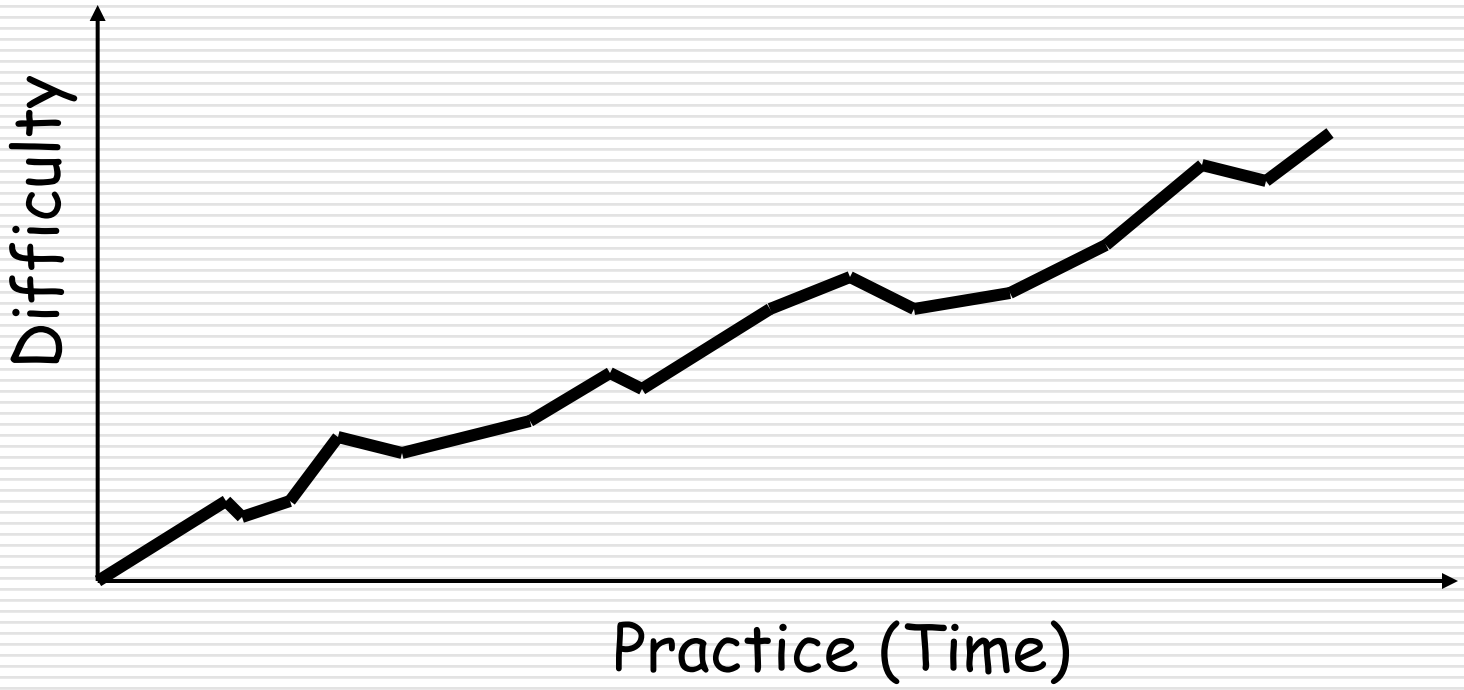
- *Stage 1* – Players learn lots, but progress slow. Often can give up. Designer needs to ensure enough progress that continues
- *Stage 2* – Players know lots, increase in skill at rapid rate. Engrossed. Easy to keep player hooked.
- *Stage 3* – Mastered challenges. Skill levels off. Designer needs to ensure challenges continue.

Difficulty Curves (1 of 2)



- Maintain Stage 2 by introducing new features!
- Too steep? Player gives up out of frustration. Too shallow? Player gets bored and quits.
- How to tell? Lots of play testing! Still, some guidelines…

Difficulty Curves (2 of 2)



- In practice, create a roller coaster, not a highway
- Many RPG's have monsters get tougher with level (*Diablo*)
  - But boring if that is all since will "feel" the same

Your Projects: Guidelines

- Decide how many levels (virtual or real)
- Divide into equal groups of EASY, MEDIUM, HARD (in order)
- Design each level and decide which group
  - All players complete EASY
    - Design these for those who have never played before
  - Most can complete MEDIUM
    - Casual game-players here
  - Good players complete HARD
    - Think of these as for yourself and friends who play these games
- If not enough in each group, redesign to make harder or easier so about an equal number of each
- Have levels played, arranged in order, easiest to hardest
- Test on different players
- Adjust based on tests

Make a Game that you Play *With*, Not *Against*

- Consider great story, graphics, immersion but only progress by trial and error … is this fun?
- Ex: crossbowman guards exit
  1. Run up and attack. He's too fast. Back to save point (more on save points next).
  2. Drink potion. Sneak up. He shoots you. Back to save.
  3. Drop bottle as distraction. He comes looking. Shoots you. Back to save.
  4. Drink potion. Drop bottle. He walks by you. You escape!
  - Lazy design!
- Should succeed by *skill and judgment*, **not** *trial and error*
- Remember: Let the player win, not the designer!

Based on Chapter 5, *Game Architecture and Design*, by Rollings and Morris

Specific Example - The Save Game Problem (1 of 3)

- Designer talking about RPG
  - *Designer*: "I've got a great trap!" … platform goes down to room. Player thinks it leads to treasure but really flame throwers. Player is toast!
  - *Tester*: "What if player jumps off?"
  - *D*: (thinks it's a loophole) … "Ok, teleport in then toast"
  - *T*: "What is the solution?"
  - *D*: "There isn't one." (surprised) "It's a killer trap. It will be fun!"
  - *T*: "So, there's no clue for player? Charred remains on platform or something?"
  - *D*: "No. That's what the 'Save' function is for."

Based on Chapter 5, *Game Architecture and Design*, by Rollings and Morris

Specific Example - The Save Game Problem (2 of 3)

- Player needs to destroy 3 generators before leaving level (or next level, powerless ship doesn't make sense)
- Level designer puts up enemy spawner at exit:
  - Infinite enemies prevent exit
  - May think: "kill X enemies and I'm done!" (like *Uncharted*)
  - Only way to realize can't leave is to die.
- D: "After dying a few times, player will realize can't leave and will finish objectives"
- Lead: "At which point, s/he throws console at the wall!"

Specific Example - The Save Game Problem (3 of 3)

- Should be used only so players can go back to their Real Lives™ in between games
  - Or maybe to allow player to fully see folly of actions, for exploration and dabbling
- Don't design game around *need* to save
  - Has become norm for many games, but too bad
  - Ex: murderous level can only complete by trying all combat options
- Beginner player should be able to reason and come up with answer
  - Challenges get tougher (more sophisticated reasoning) as player and game progress, so appeals to more advanced player
  - But not trial and error

Based on Chapter 5, *Game Architecture and Design*, by Rollings and Morris

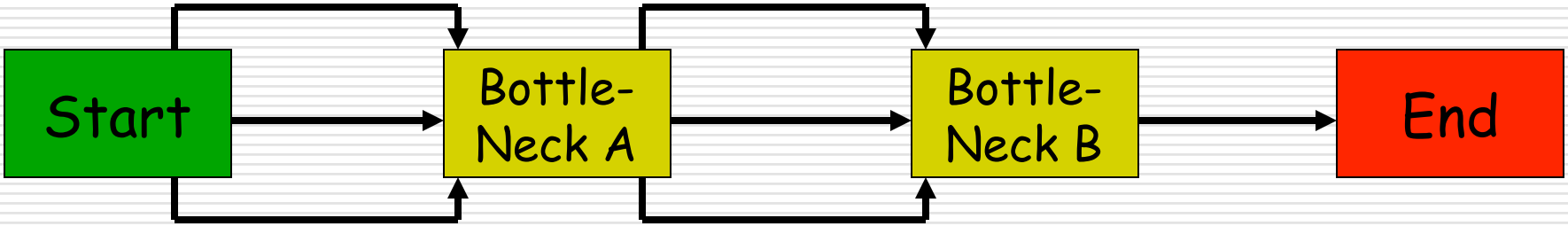
Different Level Flow Models

- Linear
- Bottlenecking
- Branching
- Open
- Hubs and Spokes

Level Flow Model: Linear

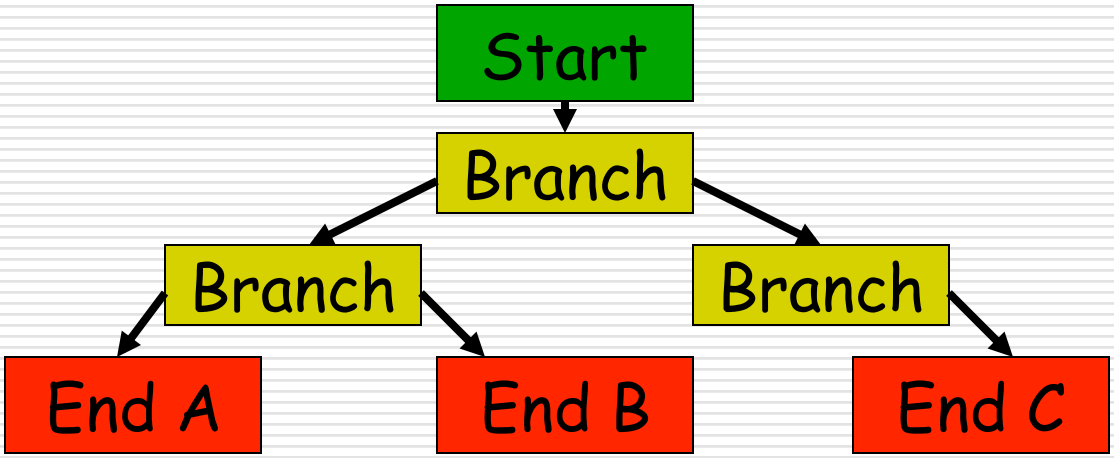
- Start on one end, end on the other
- Challenge in making a truly interesting experience
  - Often try with graphics, abilities, etc.
  - Ex: *Half-life*, ads great story
- Used to a great extent by many games

Level Flow Model: Bottlenecking



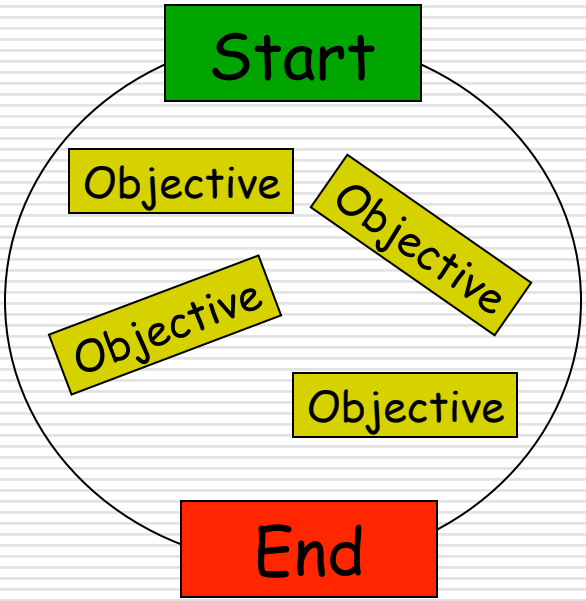
- Various points, path splits, allowing choice
  - Gives feeling of control
  - Ex: Choose stairs or elevator
- At some point, paths converge
  - Designer can manage content explosion
  - Ex: must kill bad guys on roof

Level Flow Model: Branching



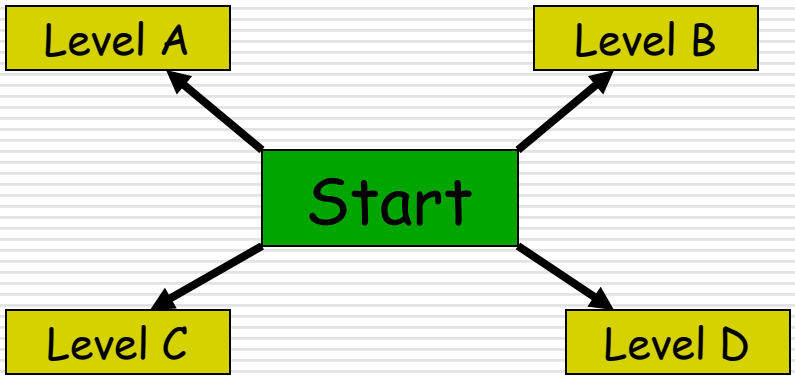
- Choices lead to different endings
- User has a lot of control
- Design has burden of making many interesting paths
  - Lots of resources

Level Flow Model: Open



- Player does certain number of tasks
  - Outcome depends upon the tasks.
- Systemic level design
  - Designer creates system, player interacts as sees fit
- Sometimes called “sandbox” level. (Ex: GTA)

Level Flow Model: Hub and Spokes



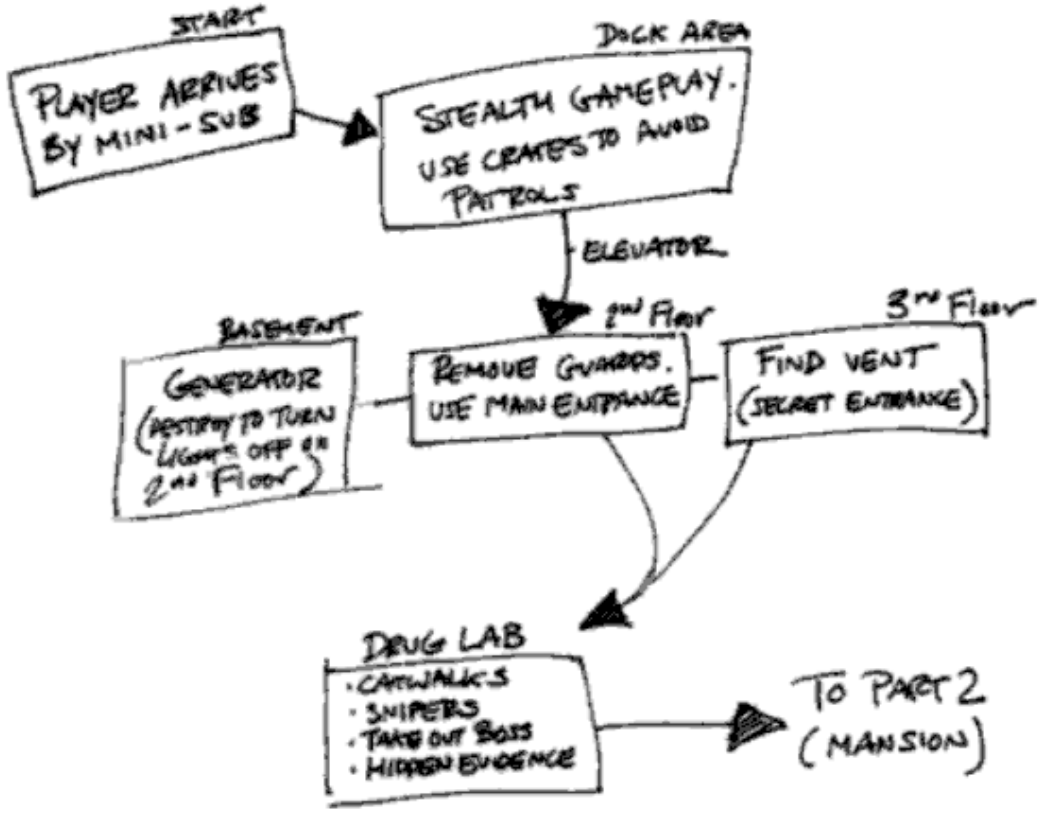
- Hub is level (or part of a level), other levels branch off
  - Means of grouping levels
- Gives player feeling of control, but can help control level explosion
- Can let player unlock a few spokes at a time
  - Player can see that they will progress that way, but cannot now

Designing a Level: Brainstorming

- An iterative process
  - You did it for the initial design, now do it for levels!
- Create wealth of ideas, on paper, post-it notes, whatever
  - Can be physical sketches
- Can include scripted, timed events (not just gameplay)
- Output
  - Cell-diagram (or tree)

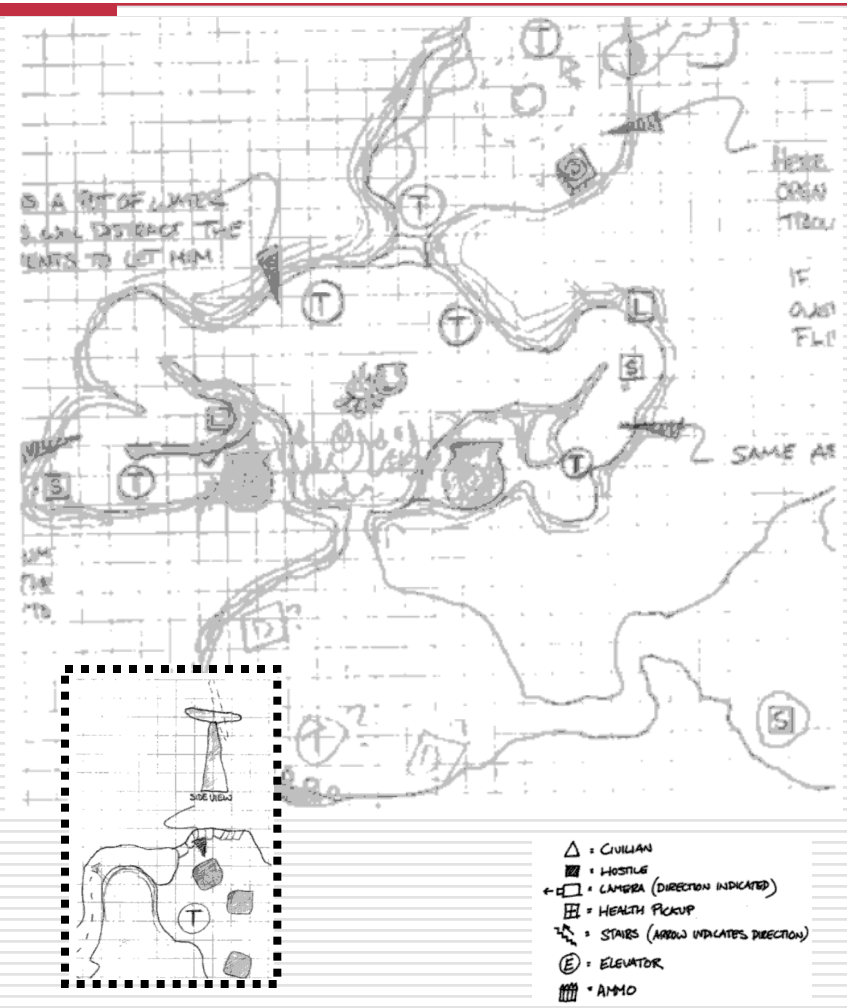
Designing a Level: Cell Diagram

- String out to create the player experience
- Ordered, with lesser physical interactions as connectors (i.e., hallways)



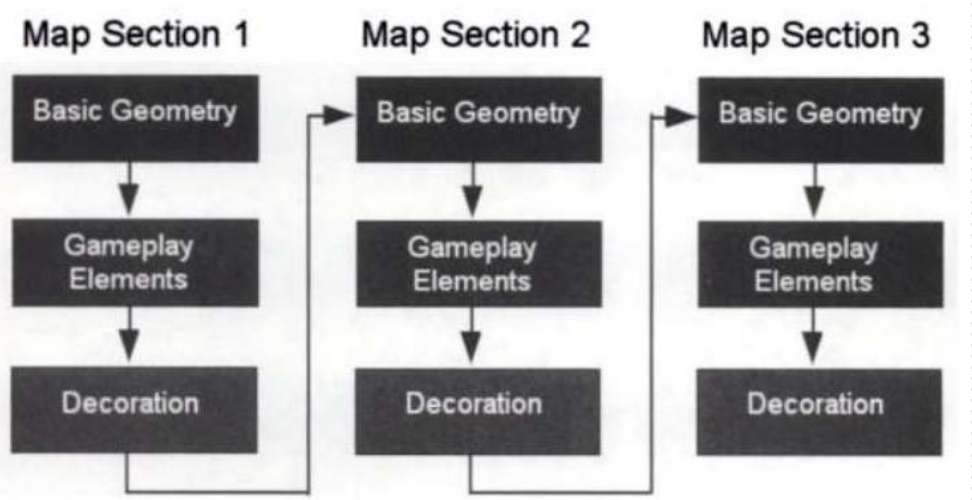
Designing a Level: Paper Design

- Graph paper
- Do whole thing, then fill in
- Starting in middle can be good
  - Don't use all creative juices early
- Include a key (enemies, health, …)
- Once started, **iterate**
- Can use *callouts* to zoom in (picture or notes)



Designing a Level: Sections

- Build a single level in sections
  - Basic boxes
  - Functional geometry
  - Add gameplay (puzzles, enemies, routes)
  - Textures, lights, sounds
- Repeat
- Good
  - Can build on and tune
  - Get feedback, try out early
  - Scales easily (can cut short, if out of time)
- Bad
  - May be working with partial assets
  - May have to go back



Designing a Level: Layers

- Build a single level in layers
  - Start to end:
    - Basic geometry
    - Gameplay elements
    - Decoration
- Good
  - Allows proper pipeline
  - Assets done when all done
- Bad
  - Needs more discipline
  - Final feedback only on end

QuakeII-DM1: An Example

- Video (Q2DM1_Layout.avi)
  - level layout and architecture

QuakeII-DM1: Architecture

- Two major rooms
- Connected by three major hallways
- With three major dead-ends
- No place to hide
- Forces player to keep moving
  - Camping is likely to be fatal

QuakeII-DM1: Placement

- Cheap weapons are easy to find
- Good weapons are buried in dead ends
- Power-ups require either skill or exposure to acquire
- Sound cues provide clues to location
  - Jumping for power-ups
  - Noise of acquiring armor
- Video (Q2DM1_Weapons.avi)

QuakeII-DM1: Result

- A level that can be played by 2-8 players
- Never gets old
- Open to a variety of strategies