Unit 11: Game Development

Games with a Purpose

Learning Objectives

Readings/Lectures

- Lecture Slides 11.01: Video Games (/bjc-course/curriculum/11-game-development/readings/01-video-games-slides.pdf)
- Reading 11.02: Fundamentals of Game Design (/bjc-course/curriculum/11-game-development/readings/02-fundamentals-of-game-design)
- Reading 11.03: Game Design Principles (/bjc-course/curriculum/11-game-development/readings/03-game-design-principles)

Labs/Exercises

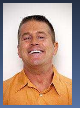
- Portfolio Project 11.01: Programming Portfolio Project (/bjc-course/curriculum/11-game-development/labs/03-internet-portfolio-project)

VIDEO GAMES

The Beauty and Joy of Computing/CS Principles

UC Berkeley EECS
Lecturer SOE
Dan Garcia

UC Berkeley EECS
TA-in-Training
Glenn Sugden



GAMIFICATION OF BUSINESS!

Channeling the "gamer addiction" to earn virtual points, companies are now adding badges and rewards to things.

E.g., Nike + (exercise game), Mint.com (encouraging savings), Foursquare (location-based social network), etc…

tech.fortune.cnn.com/2010/09/03/the-game-based-economy/

www.theesa.com/facts

How big is US video game market?

a) $100,000,000
b) $1,000,000,000
c) $10,000,000,000
d) $100,000,000,000
e) $1,000,000,000,000

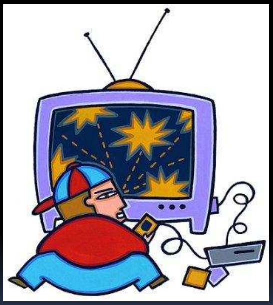
www.theesa.com/facts

How big is US video game market?

a) $100,000,000
b) $1,000,000,000
c) $10,000,000,000
d) $100,000,000,000
e) $1,000,000,000,000

Video Games : Overview

- History
  - Inventors & Games
- How
  - Design
  - 2D & 3D graphics
  - Motion Capture
  - Artificial Intelligence (AI)
- Good, Bad, Ugly
  - GWAP, RSI, Violence
- Future



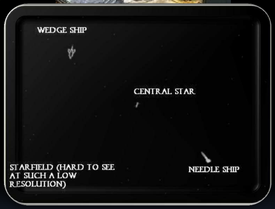
Documentaries on Video Games

- History: Video Games: Behind the Fun (2000)
  - Available on Netflix
- PBS: The Video Game Revolution (2004)
  - video.google.com/videoplay?docid=-4729348985218842392
- Discovery: History of Video Games (2006)
  - video.google.com/videoplay?docid=3637639460474263178
- ON Networks : Play Value (2009)
  - www.onnetworks.com/videos/play-value
- History of Video Games (WWW)
  - en.wikipedia.org/wiki/History_of_video_games

en.wikipedia.org/wiki/List_of_films_based_on_video_games#Documentaries_on_video_games

The Beginning : Spacewar!

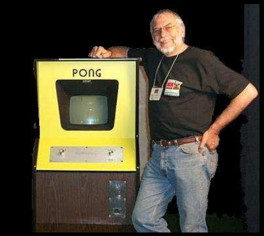
- First to gain recognition
  - Others had games before
  - “Conceived in 1961 by Martin Graetz, **Stephen Russell**, & Wayne Wiitanen”
  - Written for PDP-1 @ MIT
  - Inspired lots, widely ported
- Can still play this!
  - 1 Working PDP-1 … in CHM
  - Java version available



www3.sympatico.ca/maury/games/space/spacewar.html
en.wikipedia.org/wiki/Spacewar!
www.computerhistory.org
spacewar.oversigma.com

The Founding Fathers

- Ralph Baer
- Nolan Bushnell



www.onnetworks.com/videos/play-value/the-founding-fathers
(also on iTunes in HD 720p)

Shigeru Miyamoto

- The “Walt Disney” of computing gaming
  - Chief Game designer at Nintendo
  - 1st elected to Hall of Fame
- Designed (among others):
  - Donkey Kong
  - Super Mario Bros
  - The Legend of Zelda
  - Super Mario 64
  - Nintendo DS, Wii

www.onnetworks.com/videos/play-value/shigeru-miyamoto
www.time.com/time/asia/2006/heroes/bl_miyamoto.html
en.wikipedia.org/wiki/Shigeru_Miyamoto

History of Video Games : 1970s

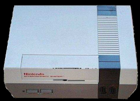
- Golden age of video arcades
  - Pong, Space Invaders, Asteroids, Pac Man
- 1st gen consoles (1972–1976)
  - Magnavox Odyssey
- Mainframe computers
  - Hunt the Wumpus, Rogue
- Home computers
  - Type the program in!
  - Floppies, Tapes. Zork, others.
- 2nd gen consoles (1977–1984)
  - Atari 2600, Intellivision, Colecovision, Activision

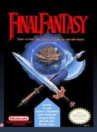
ASTEROIDS
ATARI
• Explosive rapid-fire space action • 1 or 2 players are challenged to destroy asteroids and enemy spacecraft • New Atari-designed QuadraScan™ display system • New personal high score table display • Optional “Hyperspace” • Optional coinage including Susan B. Anthony coin slot • Bonus play at 10,000 points.

en.wikipedia.org/wiki/History_of_video_games
www.thegameconsole.com

History of Video Games : 1980s

- Genre innovation
- Gaming computers
  - Apple II, Commodore 64, Atari 800
- Early online gaming
  - Mostly text only, MUDs
- Handheld LCD games
- Video game crash of 1983
  - Atari buried millions of ETs in dump
- 3rd gen consoles (1985–1989)
  - Nintendo Ent. System (NES)
    - Super Mario Bros, Zelda, FF I
    - Gamepad introduced



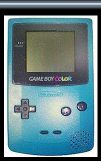


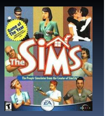
History of Video Games : 1990s

- Decline of arcades
- Handhelds come of age
  - GameBoy, Sega Game Gear
- Mobile phone gaming
- Fourth generation consoles (1990–1994)
  - Sega Genesis, Super NES
- Fifth generation consoles (1995–2000)
  - Playstation, Nintendo 64 (with Super Mario 64)
- Transition to 3D, CDs
  - Crash Bandicoot, Tomb Raider

History of Video Games : 2000s

- Mobile games
  - iPhone (games ½ apps)
- Sixth generation consoles (since 2001)
  - PS2, Xbox, GameCube
  - Return of alternate controllers (DDR, guitars)
- Online gaming rises to prominence
  - WoW, Ultima Online
- Rise of casual PC games
  - Bejeweled, The Sims



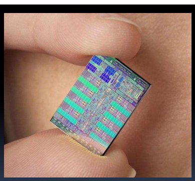


History of Video Games : 2005+

- Seventh generation consoles (since 2005)
  - Portables
    - Nintendo DS, PSP, iPhone
  - Consoles
    - PS3, Xbox 360, Wii
  - Increases in development budgets
  - Motion control revolutionizes play
    - Wii controller, iPhone

Example: Playstation 3 Hardware

- State-of-the-art system
  - But SW determines success!
  - (also, cool controllers helps)
- 9 3.2GHz Cores (1PPE, 8SPE)
  - Power Processing Elt (PPE)
    - Supervises activities, allocates work
  - Synergystic Processing Elt (SPE)
    - Where work gets done
    - During testing, one "locked out"
      - I.e., it didn't work; shut down



en.wikipedia.org/wiki/PlayStation_3
www.us.playstation.com

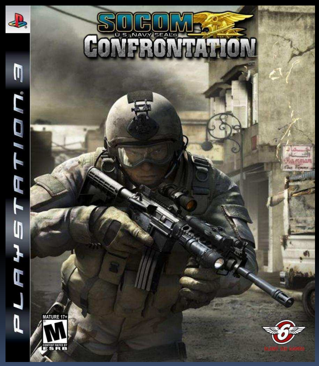
Design of a *Casual* Video Game

- Staff requirements
  - Can be done by one person, ala days of old
  - Bigger teams also (< 10)
  - Lots of new developers
- Phones great platforms
  - iPhone dominates field
  - Students are signing up!
- Time to completion
  - Often only a few months!

www.apple.com/iphone/apps-for-everything/fun-and-games.html
blog.entertonement.com/2009/07/7-addicting-casual-games
en.wikipedia.org/wiki/Casual_game

Design of a *Core* Video Game

- Staff requirements
  - Cross-disciplinary
  - Producer, programmers, game, graphic & sound designers, musicians, testers, …
  - 100+ person teams
- Similar to film
  - Often, games->film, and film->games
  - Lucasfilm, etc. want to tie assets together

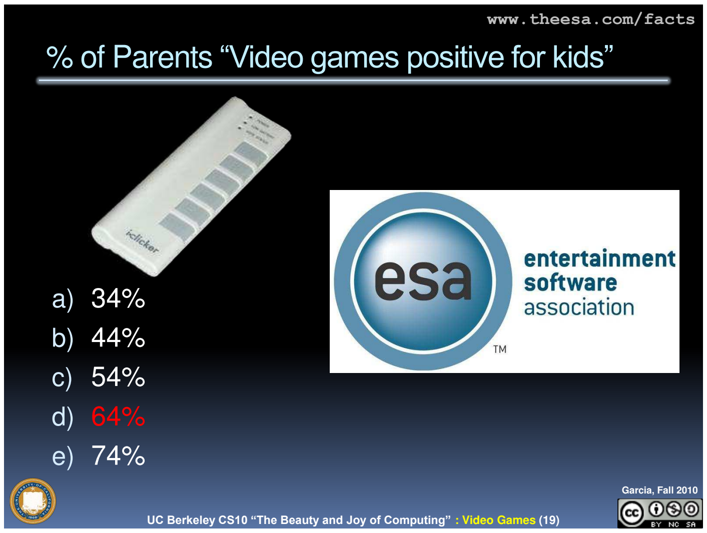


en.wikipedia.org/wiki/Video_games

www.theesa.com/facts

% of Parents "Games positive for kids"

a) 34%
b) 44%
c) 54%
d) 64%
e) 74%

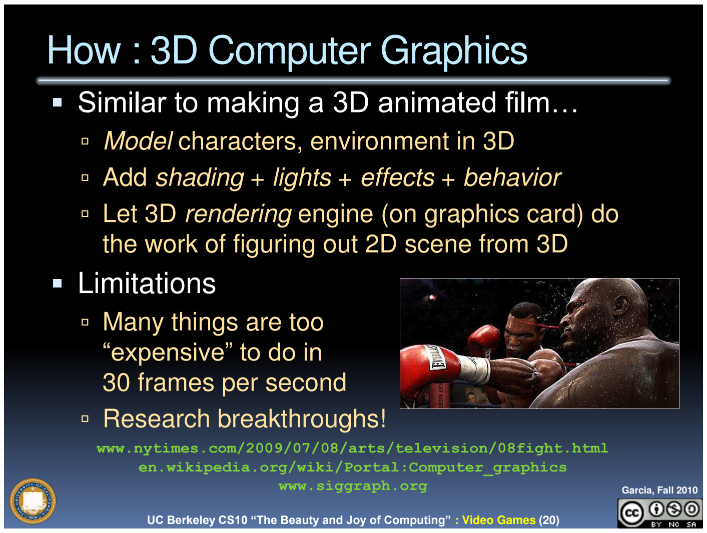


www.theesa.com/facts

% of Parents "Video games positive for kids"

iclicker
esa
entertainment software association
TM

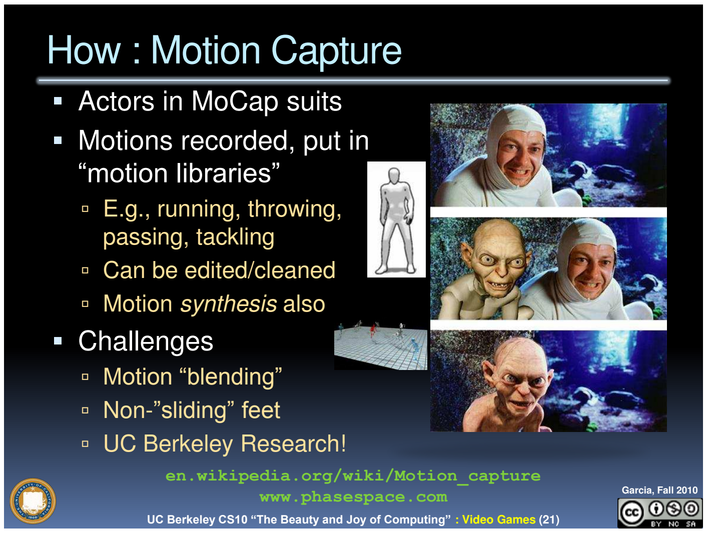
a) 34%
b) 44%
c) 54%
d) 64%
e) 74%



How : 3D Computer Graphics

- Similar to making a 3D animated film…
  - *Model* characters, environment in 3D
  - Add *shading* + *lights* + *effects* + *behavior*
  - Let 3D *rendering* engine (on graphics card) do the work of figuring out 2D scene from 3D
- Limitations
  - Many things are too "expensive" to do in 30 frames per second
  - Research breakthroughs!

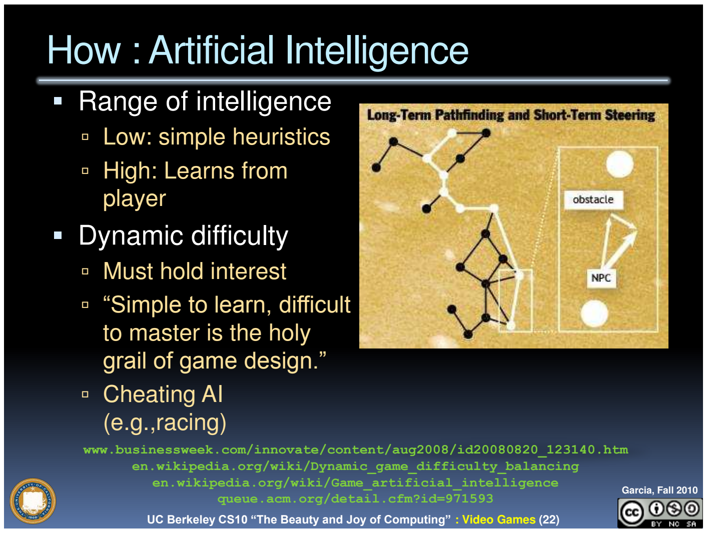
www.nytimes.com/2009/07/08/arts/television/08fight.html
en.wikipedia.org/wiki/Portal:Computer_graphics
www.siggraph.org



How : Motion Capture

- Actors in MoCap suits
- Motions recorded, put in "motion libraries"
  - E.g., running, throwing, passing, tackling
  - Can be edited/cleaned
  - Motion *synthesis* also
- Challenges
  - Motion "blending"
  - Non-"sliding" feet
  - UC Berkeley Research!

en.wikipedia.org/wiki/Motion_capture
www.phasespace.com

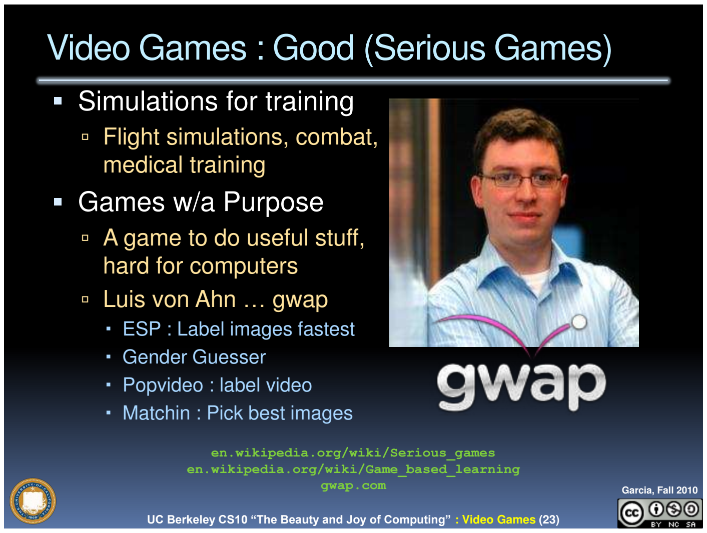


How : Artificial Intelligence

- Range of intelligence
  - Low: simple heuristics
  - High: Learns from player
- Dynamic difficulty
  - Must hold interest
  - "Simple to learn, difficult to master is the holy grail of game design."
  - Cheating AI (e.g.,racing)

Long-Term Pathfinding and Short-Term Steering
obstacle
NPC

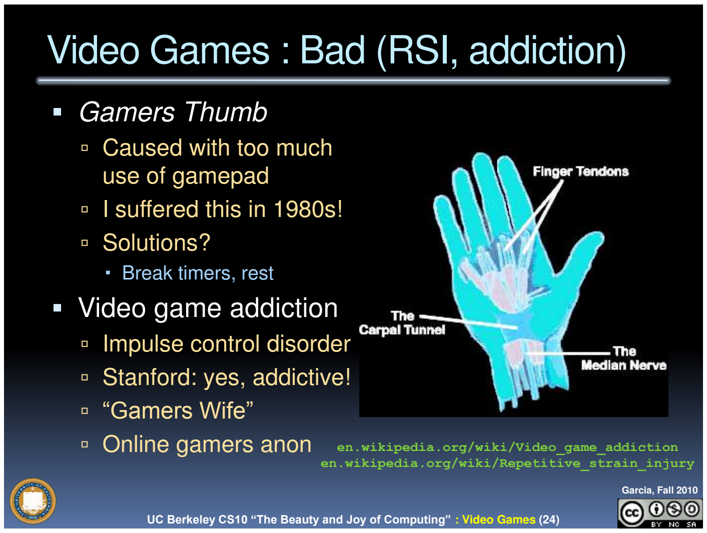
www.businessweek.com/innovate/content/aug2008/id20080820_123140.htm
en.wikipedia.org/wiki/Dynamic_game_difficulty_balancing
en.wikipedia.org/wiki/Game_artificial_intelligence
queue.acm.org/detail.cfm?id=971593



Video Games : Good (Serious Games)

- Simulations for training
  - Flight simulations, combat, medical training
- Games w/a Purpose
  - A game to do useful stuff, hard for computers
  - Luis von Ahn … gwap
    - ESP : Label images fastest
    - Gender Guesser
    - Popvideo : label video
    - Matchin : Pick best images

en.wikipedia.org/wiki/Serious_games
en.wikipedia.org/wiki/Game_based_learning
gwap.com



Video Games : Bad (RSI, addiction)

- *Gamers Thumb*
  - Caused with too much use of gamepad
  - I suffered this in 1980s!
  - Solutions?
    - Break timers, rest
- Video game addiction
  - Impulse control disorder
  - Stanford: yes, addictive!
  - "Gamers Wife"
  - Online gamers anon

Finger Tendons
The Carpal Tunnel
The Median Nerve

en.wikipedia.org/wiki/Video_game_addiction
en.wikipedia.org/wiki/Repetitive_strain_injury

Video Games : Ugly (Violence)

- Violent video games
  - Increase aggression, decrease "helping"
  - Others found no link
- High-profile incidents
  - Columbine kids loved the Doom video game
- Ratings help
- Games "folk devil"
  - Billions $, kids at stake

I HATE THOSE BOGUS STUDIES! THEY'RE SO FULL OF IT!
SERIOUSLY! WE SEE TONS OF VIOLENCE AND WE'RE FINE!
HERALD-TIMES VIOLENT TV AND GAMES LINKED TO AGGRESSION
SELTZER
MAN, THOSE GUYS TICK ME OFF!
BUNCH OF QUACKS!
HERALD-TIMES
PARENTSTV.ORG

en.wikipedia.org/wiki/Video_game_controversy
www.apa.org/science/psa/sb-anderson.html

Future of Video Games

- Media producers connecting assets
  - Disney, Lucas big players
- Controllers and sensors expand
- Games on Demand
  - OnLive
- Brain-Computer Interface (BCI)
  - Invasive and Non-

[redacted]

en.wikipedia.org/wiki/Brain-computer_interface

The Fundamentals of Game Design

October 12th, 2010

I got a request via Twitter for this old essay which had fallen off the Internet, so I am posting it here. This was originally written for Metaplace users… there is nothing here new to anyone who has followed the blog for a while, but since it was requested, here it is.

The fundamentals of game design

Starting out creating an interactive experience, of any sort really, can be rather daunting. In this tutorial, we'll run through the basic components of a game, so we can get a handle on what the next steps are when you make the jump from the training tutorials to your own projects.

Often people have trouble when conceptualizing a game. The idea, after all, is often the easy part. It's actually making it, and figuring out where to start, that is the hard part.

A friendly warning, though! Just like writers have different ways of working, and some composers write music in their head and others at an instrument, different game designers are going to have different ways of working. Some work better "in the code" and others like doing everything on paper beforehand. Some think in terms of story and narrative, and others are systems designers first and foremost. So this tutorial may actually run a bit against the grain for you, depending on your natural temperament.

In what follows, I am going to use the language of games, but really, every piece of advice in this article applies equally if you are designing any sort of interactive project whatsoever. So just because I say "game" in what follows doesn't mean this article won't be useful to you when you start making a classroom experience or a chat room or some other application.

The components of a game

The first thing to understand is that games are made out of games. A large game is actually composed of minigames. Even a small game is built out of very very simple small games. The smallest games are ones that are so simple and stupid, you can't lose. You can think of this as "game atoms," if you like.

For example, in the classic puzzle game Tetris, the basic game is beating your high score. To beat that game, you have to master the game of forming lines. There's actually multiple variants there, because you have to learn the games of placing all the different sorts of blocks. And finally, the simplest game is rotating a block, which is just a button and hard to screw up. So games are built out of games. This brings us to key piece of advice #1:

Advice #1: Design one game at a time.

Turtles all the way down

Even if you are making a complex game, built out of many "game atoms," each atom is a game in its own right, and has to feel fun and satisfying. Even the stupid ones with no challenge have to feel good. Imagine how poor a game Tetris would be if the stupidly trivial game of "press a button and watch the block spin" wasn't satisfying.

Many games are ruined at this very fundamental level by poor design. For example, a bad designer might have decided that a random chance of the block not rotating would make sense. After all, we use random chance in gambling, board games, and roleplaying games, right? But it would make Tetris unplayable.

Game atoms

OK, so you’re going to design one of the game atoms. Luckily, every game atom has the same characteristics:

- A player does something.
- The opponent (which might be the computer) calculates a response
- The player gets feedback.
- The player learns from this feedback, and gets to do something again.

You can think of these steps in very abstract terms:

- Input
- Model
- Feedback
- Mastery

Really, that is it. Let’s apply it to our Tetris example again. At the trivial “rotate a block” level, we have

- A player hits a button.
- The computer calculates that this means rotate the L counter-clockwise.
- The player is given the feedback of the block in its new position.
- The player figures out “I bet I can do this with other sorts of blocks too. And there’s probably a rotate clockwise button somewhere. Rotating is my goal!”

Advice #2: make sure the controls match up well to what the player is attempting to do

At a more advanced level we have

- A player can rotate left, right, drop a block, glance at the next block, etc. Lots of choices.
- The computer is going to take its turn and move the block further down regardless, or spawn a new block of a random shape if there isn’t one.
- The block moves down. Maybe it completes a line, maybe it doesn’t.
- The player says “aha! Completing lines is my goal, and different shapes help or hinder that!”

Notice that if any of these four steps is poorly chosen, the whole game sucks.

- A player moves the mouse.
- The computer figures this means rotate a block.
- The player is not shown the block, but instead a stock quote.
- The player is baffled and quits.

Advice #3: make sure the player can actually learn from the feedback you give them.

Where does the fun come from?

The fun comes from the mastery process. But what the player is mastering is the model. All games are mathematical models of something. We often speak, for example, of Chess being like war (we actually speak of lots of games as being like war!).

Even games like Tic-Tac-Toe are expressible as math puzzles. Games of resource management over time (like an RTS or Civilization) are exercises in calculus. RPGs where you make choices in character building are actually examples of exploring possibility spaces searching for local maxima… games lie to us all the time about what they are really about.

Advice #4: try to stop thinking about what your game looks like, and think about what it is actually modeling

The underlying math

All this sounds incredibly geeky, but you don't have to be a math geek to enjoy games. The trick is to make the pill go down easy. And the fact is that some math problems and models are more interesting than others. Nobody is that interested in a game that pushes you to solve "2×5 = ?" over and over. It has to be a sort of problem that you can come to again and again, and explore possibilities looking for alternate solutions and paths.

This means that there's a specific and highly varied set of problems that make for good games. In math, a lot of these problems are what is called NP-Hard problems. You don't need to dig into higher mathematics to be a good game designer, though. Instead, you need to ask yourself a basic set of questions:

- Where?
- When?
- How?
- What?
- With?
- For?
- Few?
- Phooey.

This list seems facetious, but it's a shorthand way of asking yourself the following questions about your game atom:

- Do you have to prepare for the challenge?
- …where prep includes prior moves? …and you can prep in multiple ways?
- Does the topology of the space matter?
- …does the topology change?
- Is there a core verb for the challenge?
- …can it be modified by content?
- Can you use different abilities on it?
- …will you have to in order to succeed?
- Is there skill to using the ability?
- …or is this a basic UI action?
- Are there multiple success states?
- …with no bottomfeeding? …and a cost to failure?

You have to answer yes to all of these for your game atom to be fun. And yes, we mean every atom in the game has to meet these criteria.

Advice #5: check this list for every action a user can take, every decision they make

How do you get there?

There’s really only one way, right now. You prototype and iterate. Don’t get hung up on the visuals, except for worrying about whether you are giving enough feedback. The best games can be played using sticks and stones. (You can play most roleplaying games with pretty much any random chance generator, for example).

You can prototype with all sorts of things. I have a “prototype kit” because I often prototype using physical objects before going into the code. It consists mostly of stuff that I can pick up at a craft store:

- Two decks of regular cards.
- One deck of Uno cards.
- One Go board.
- One Checkers board.
- A half dozen six-sided dice.
- One full set of polyhedral dice.
- A large stack of differently colored index cards.
- Twelve pounds of differently colored beads. Go to the pottery aisle at your local craft store — these are the kind that get put in fish tanks and potted plants. It’s a bit more than a buck for a pound of one color.
- Wooden pieces, also from the craft store. These are found in the aisle with the clock faces:
- wood cubes, various sizes
- colored flat squares, three sizes
- dowel rods
- ‘pawn’ pieces
- wooden chip (circles)
- assorted circles, hexagons, stars, etc
- Blank wooden clock faces that you can draw boards on.
- Wood glue
- Dremel tool
- Square glass chips (also from the craft store, asst colors)

But even that is overkill for most people.

Advice #6: watch others play your game

You’ll quickly see where you didn’t provide enough feedback, or where they can’t figure out the underlying model.

A final thought

Fundamentally, never forget that if you want to design, you have to just go do it. The only way to get better at it is to keep doing it, because gamemaking is in itself a great and varied game to play.

Article copied (and gently modified) from Raph Koster’s Website located at http://www.raphkoster.com/2010/10/12/the-fundamentals-of-game-design/

The 13 Basic Principles of Gameplay Design

By Matt Allmer

Gameplay design is chaotic and full of frustrations and contradictions. More often than not, the request is to come up with something guaranteed to be successful. This condition steers solutions towards the established – which means solutions that have been done before.

But in the same breath, the product must separate itself from the competition or stand out in some way. This immediately pulls the designer in conflicting directions.

Then, whatever the solution, it must fit within the confines of the project's resources. Not to mention scheduling pressure and strategy changes coming from executive positions.

Hup hup! No time for analyzing the previous paragraph! We've got a title to ship! Never mind your lack of proper tools! Quit your sniveling! Don't you know?

Game design is like sailing a ship while still building the hull! Jump out of a plane while still sewing your parachute and you'll get a good sense of pace in this business. The horse is never put before the cart. We race them side-by-side to see which one wins!

With so much urgency, conflict and uncertainty, there must be an anchor somewhere. Call me boring, but I'm a fan of preparation and established fundamentals. They give me a better understanding of which rules I can break, and which rules I should think twice about.

I took a traditional animation class in college and on the first day, the professor handed out the "12 Principles of Animation", introduced by Frank Thomas & Ollie Johnston. If you're not familiar with these two, they were part of the Nine Old Men: The legendary Disney animation crew responsible for the studio's timeless classics, such as, Snow White, 101 Dalmatians, Bambi, Sleeping Beauty, and others.

At first, these 12 principles were difficult to fully grasp. However, by the end of the semester, I noticed the more principles I applied to my work, the better the animation. Remembering that experience, I think to myself, "By George! Game design should have something similar!"

So, George and I scoured the Internet. Unfortunately, I was disappointed after finding so many disjointed theories, strategies, approaches and creeds. There was a lot of broad subject matter like theories on fun, rewarding players' choices, controlling thought activity, mental multi-tasking… and calls to "simplify" (whatever that means. I'm a designer for crying out loud).

I also found principles so apparent, Captain Obvious would roll his eyes: "know your audience", "don't break the player's trust", "give players choice", "know thyself", "one mechanic in the engine is two in the bush". Alright, the last two were made up, but nothing I found really did it for me.

I was perplexed. None of what I found would help a designer on a day-to-day basis. So George, Captain Obvious and I have decided to throw our proverbial hat into the muddled picture. (And quick! For god's sake, before I collect any more metaphorical personalities!)

The 12 Basic Principles of Animation was my starting point. I took the commonalities and added to them based on what I've identified as the different compartments of gameplay design. You'll notice some are described similarly and some even have the same name, but all apply to gameplay.

The purpose of these principles is to cover all your bases before presenting your designs. You might have a principle fully covered in the beginning, but these principles may spark a thought later when circumstances present a new opportunity. Think of this as a reference sheet. And now, without further ado…

Direction

The first three principles have to do with leading and directing the player's experience. Even though this medium is heavily based on personal, interactive discovery, it is still an artistic medium.

Do not underestimate the importance of artistic direction. Just as a painting leads the eye, a book leads the imagery, a film leads the narrative, so too must a game lead the interactivity.

1. Focal Point

Never allow the player to guess what they should focus on. At the same time, always allow secondary subject matter, but it is the designer's job to clearly provide the primary focus at all times. This applies to both visual and visceral aspects of gameplay.

Level design example * Creating clear, apparent lines of sight.

System design example * Clearly defined plot points and objectives during game progression/user experience.

2. Anticipation

Time is needed to inform the player that something is about to happen. Always factor in Anticipation when designing and implementing events and behaviors.

Level design example

- A train sound effect occurs before player sees train.
- System design example
- An energy charge builds before the lightning attack occurs.

3. Announce Change

Communicate all changes to the player. This short step occurs between Anticipation and the event itself.

The important part to remember is maintaining a hierarchy of notable changes.

A good rule of thumb is degree of rarity. If a change occurs a hundred times in an hour, the announcement may not be required. However, if the change occurs five times throughout the entire game experience, a number of visual cues could be needed.

This principle is so obvious, it can be taken for granted and sometimes overlooked. Be diligent in knowing what changes the player should be aware of at the correct time and on the correct event.

Level design example * "Cast-off" animations trigger for NPCs when the player's character boards the ship.

System design example * An on-screen notification occurs when quest criteria have been completed (i.e. "Slay 10 goblins for Farmer Bob")

Behavior

These next four principles address the very important aspect of behavior. This tackles the player's expectations, both conscious and unconscious. This is where common design theories are addressed such as player choice, reward and payoff, etc. These principles are also broader, so they can be applied to additional types of design like UI and story…

4. Believable Events and Behavior

Every event or behavior must occur according to the logic and expectations of the player. Every action, reaction, results, emotion and conveyance must satisfy the players' subconscious acceptance test.

Level design example * Place destructible objects near an explosive object. This way, the explosion looks more believable.

System design example * Weaker enemies run away when the advantage shifts in the player's favor.

UI example * HUD elements are affected when player's mech is near death.

Story example * Villagers are more upbeat and react positively after the player has slain the dragon.

5. Overlapping Events and Behavior

Dynamic is lost if only one change occurs at a time. Discover the right amount of events to occur at any given moment of time.

Level design example * Providing the player the ability to build from an appropriate list of structures.

System design example * The linebacker points to direct fellow players, the defensive end shifts over, the quarterback points and calls out football jargon and the crowd cheers louder because it's third down. All this occurs before the snap.

UI example * Points accumulate in the score while each kill is individually tallied on screen.

Story Example * Multiple plot points are at the forefront of the narrative experience. Example: the king is on his deathbed while his war is being waged and he has yet to announce an heir – all while an unknown saboteur orchestrates a military coup.

6. Physics

The player's primary logic operates within the known possibilities of physics. Keep in mind gravity, weight, mass, density, force, buoyancy, elasticity, etc. Use this as the starting point, but do not be limited by it.

Level design example * Ensuring a hole in the floor is the correct size for the correct purpose. Whether it is part of the path of level progression, or simply for visual aesthetics.

System design example * A spark particle effect occurs when the player's vehicle scrapes the side of the concrete wall.

UI example * The GUI's theme references scrapbook elements. In which case, animated transitions, highlights, etc. follow the physical characteristics of paper.

7. Sound

Ask yourself, "What sound does it make when ________ happens?" "Is the sound appropriate?" "Is the sound necessary?" "Does it benefit the experience or hinder it?" If players close their eyes, the sound alone should still achieve the desired affect.

It's debatable whether this principle should be included since Sound Design can be considered separate from Gameplay Design. I've included it because sound is crucial and can easily be neglected. The more it is considered, the better the experience is for the player.

Level design example * Flies in swamp level make a sound when close to the camera.

System design example * A proximity system where sound effects volume fluctuates depending on distance of game assets.

UI example * Only visually prominent graphics have sound effects attached to them, so as not to muffle the auditory experience.

The next three principles individually touch on other major design components.

Progression

8. Pacing

Keep in mind the desired sense of urgency, the rate in which events occur, the level of concentration required and how often events are being repeated. Spread out the moments of high concentration, mix up the sense of urgency, and change things wherever possible to achieve the proper affect.

Level design example * Create areas for the player to admire the expansive view, versus areas where the player feels claustrophobic.

System design example * Create long, powerful attacks versus short, light attacks.

Environment

9. Spacing

Understand how much space is available both on-screen and in-world, recognize the spatial relationship between elements and take into account the effects of modifying those spaces.

Level design example * Lay out the appropriate amount of space for the appropriate number of enemies to maneuver correctly.

System design example * When an AI character moves through a bottleneck area, walk loops switch to standing idle when the AI character is not moving forward, to show that the character is “waiting” to move through the narrowed space.

Method

10. Linear Design versus Component Breakdown

Linear Design involves solving challenges as they come. All solutions and possibilities hold the same institutional value. Focus can be lost with this method, but it provides creative and spontaneous solutions.

Component Breakdown involves systemic categorization and forming a logical hierarchy of all solutions. This method can restrict innovation but preserves clarity of primary design objectives.

This principle does not mean designers must choose one or the other. There are times during development where one method is more appropriate than the other.

For instance, pre-production provides plenty of time for breaking down a sequence of events. However, when the publisher drops a “must have” change after pre-production, linear design can provide an acceptable solution quickly.

Level design example * Typical blocking of level geometry in an early stage of development, versus adjusting a small area of the same level to implement an idea that wasn't thought of until later.

System design example * Identifying all major systems (combat, AI, input, etc), and progressively filling in various levels of detail versus conceiving the first couple of levels and extracting possible systems based on a linear player experience.

Foundation

The final three principles mark the foundation of gameplay design, which are listed in reverse order of importance. These should be a surprise to no one.

11. Player

How does the player factor into this? How does the player interact with everything that has been designed? More than just device input, address how the player contributes to the experience. If it's a good idea and you're able to convey it correctly but the player is not into it, change it or scrap it!

Level design example * Setting up the player in hopes of making them jump out of their seat.

System design example * Orchestrating progression so that the player feels empowered, determined, anxious, etc.

12. Communication

Is the appropriate team member correctly aware of the objective? Are the appropriate developers clear on the solution? If it's a good idea but you can't communicate it correctly, it might as well be a bad idea because it's very likely to be received as such.

Level design example * Using the elements of the environment so the player is compelled to travel in the correct direction.

System design example * Using visual cues so the player learns when to punch rather than kick, jump rather than strafe, etc.

13. Appeal

When addressing anyone, ask yourself, "Does this draw the audience in?" This applies to (but is not limited to) the player, the spectator, your fellow developers, the publisher, and their marketing team. If it's not a good idea, there's no need to continue until it becomes a good idea or is replaced by something better.

Level design example * Running down the street is not fun, but running down the street while being pursued by government secret agents is.

System design example * Punching can be fun but when the camera shakes on impact, it's even more fun.

Conclusion

So, there you have it. These principles have noticeably improved my designs and forced me to think of components from all angles. I thoroughly believe they will give you an edge on all those impatient carts. So, stick that in your horse and race it!

Article located at http://www.gamasutra.com/view/feature/132341/the_13_basic_principles_of_.php?print=1

This is a test/project grade.

Portfolio Project: Programming

Learning Objectives

The Programming Portfolio Task addresses the following CS Principles Learning Objectives:

- 1: The student can use computing tools and techniques to create artifacts.
- 2: The student can collaborate in the creation of computational artifacts.
- 3: The student can analyze computational artifacts.
- 4: The student can use computing tools and techniques for creative expression.
- 5: The student can use programming as a creative tool.
- 8: The student can develop an abstraction.
- 17: The student can develop an algorithm.
- 21: The student can explain how programs implement algorithms.
- 22: The student can use abstraction to manage complexity in programs.
- 23: The student can evaluate a program for correctness.
- 24: The student can develop a correct program.
- 25: The student can collaborate to solve a problem using programming

Task

This portfolio entry includes work on a team (usually 2 members) and as an individual.

Collaborative portion

- Collaborate with a partner to identify an area of focus or subject of your choosing in which you can write and reflect on several different programs that solve problems or do something that is personally relevant to you. You will need to create programs with a purpose.
- The discussion below uses the term area to refer to the subject or focus you have chosen as this first part of this portfolio task.
- The area you choose should be one that allows you and your partner to write programs that go beyond simple explorations.
- Work together to write a program to solve a problem in the area you chose or to illustrate features and characteristics of the area.

Individual portion

Working alone, write another program to solve a different aspect of the problem or to illustrate some other feature or characteristic of the area you chose.

Working alone, produce a single piece of writing that reflects on each of the following:

- the collaborative experience,
- the program that resulted from the collaboration, and
- the program you created individually.

This reflection piece should include an explanation of how you see the programs as part of the area you chose.

Prepare and submit the following

The program you developed collaboratively with your partner within your area. You and your partner will each submit this same program. * Please turn in your brainstorming and storyboards as well.

The program you developed independently within your area. * Turn in your storyboards as well

An individually written description and reflection document as described below in the Requirements section. Each group member must write her/his own reflection.

Presentation

Use PowerPoint or Prezi to create your presentation. Each group member should participate in the presentation

Group Portion

- Discuss your chosen area of focus/subject
- Discuss your chosen target audience
- Discuss the program your group created
- Purpose of the program
- How it might be used by your focus group

Individual Portion

- Discuss how your individual program could be used to solve a different aspect of the problem or to illustrate some other feature or characteristic of the area you chose.
- Each group member should turn in the presentation

Requirements

You must work with a partner on this project. You will collaboratively identify an area and will collaboratively develop a program within that area. You and your partner will work with your instructor to identify a suitable area that lends itself to creating programs that are reasonably complex.

Each program must include each of the following:

- sequencing,
- selection (decision statements - If),
- abstraction (your own procedures), and
- either iteration (looping) or recursion.

You are free to determine the purpose of your programs, but your programs should leverage the richness of the area you have chosen. For example, your programs might solve a problem or create something creative or artistic (though you are not limited to these two examples).

Furthermore, your programs should be demonstrate the complexity of the programming environment/language you are working in.

The programs you and your partner produce individually must be different, both in content and purpose, from one another and from the one you write collaboratively. All the programs must be relevant to the area you chose.

Your individually written description and reflection document has a maximum length of 400 words.

Your individually written reflection document should include the following:

- A brief description that identifies the area in which you and your partner chose to work, as well as the problem you attempted to solve or the features and/or characteristics you attempted to illustrate.
- A brief description that describes the additional problem that, working alone, you attempted to solve or the features and/or characteristics you attempted to illustrate.
- A description of why the programs are relevant to the area you chose.

An evaluation of the correctness of both your own individual program and the program you produce with your partner. This evaluation should include

- a description of the algorithms these programs implement,
- an explanation of how these programs function at a detailed level, and
- a justification of why these programs perform correctly.
- A description of the collaboration process you used to create your shared program.
- Describe in what ways this process was or was not appropriate for the creation of your shared program.

Turn in

Every team member should turn in all documents.

- Group Planning - Brainstorming, Storyboards (10 pts)
- Group Program (25 pts)
- Individual Planning - Storyboards (10 pts)
- Individual Program (25 pts)
- Individual Description/Reflection document (10 pts)
- Group/Individual Presentation (20 pts)