8/29 - MDA

First vs Second Order Design

- First order
    - Rules mechanics, systems
- Second order
    - Player experiences (aesthetics) by manipulating mechanics and dynamics
- Example:
    - First order: puzzle piece rotation mechanic
    - Second order: players feel accomplished when solving puzzle

Defining game mechanics

- **Basic rule or action available to the player**
- Ex:
    - Super Mario Bros. 3
        - Press A to jump
        - Hold B to run
    - Chess
        - Moving a piece diagonally

Game Mechanics

- Mechanics alone don't make a good game
- Sometimes a platformer game doesn't feel right
    - Jumping is delayed, can't move while in the air, etc

Dynamics

- Dynamics are:
    - The behavior of the system during gameplay (what happens at runtime)
        - The game in motion
    - Actions taken by players that arise from engaging with the game's mechanics
- Example: Chess
    - Mechanics:
        - Pieces have specific movement rules
    - Dynamics:
        - Strategic interactions between players emerge as they plan moves, sacrifice pieces, and adapt to their opponent

    - Aesthetics:
        - Challenge, competition

Aesthetics

- Mechanics + dynamics = Aesthetics

2 approaches to game design

- Mechanics-First
    - Example: paraglider in an obstacle course vs narrative adventure
        - Aesthetics is last, mostly thinking about a paraglider and what it could do
- Aesthetics-First
    - Example: designing for exploration, storytelling, or player expression
- Goal: Meet in the middle

Aesthetic #1: Sensation

9 aesthetics mentioned in the paper, but there are more.

- Immersive sensory experience, memorable audio-visual effects
    - Example: Journey or Just Dance

Aesthetic #2: Fantasy

- role-playing/cosplaying and imagining alternate lives
    - Example: FIFA

Most games have multiple aesthetics, but what is the CORE aesthetic? (for exam)

Aesthetic #3: Narrative

- Drama, immersion, world-building
    - Example: Skyrim

Aesthetic #4: Challenge

- Overcoming obstacles
    - Recommended for new designers, easiest aesthetic to code for
    - Recommended for a platformer

Aesthetic #5: Fellowship

- Social interaction and cooperation

- Example: Among us

Aesthetic #6: Discovery

- Exploring, finding, and learning
  - Example: Minecraft
- Could be a huge land mass to explore (unchartered territory)
- Could be like pokemon, discover new pokemon
- Withholding instructions to discover controls

Aesthetic #7: Expression

- Customization, user-generated content
  - Example: The Sims
- You are inserting your own identity into the game
  - Compared to fantasy where you are becoming something else

Aesthetic #8: Abnegation (Submission)

- Passing time, losing oneself
- Player is doing repetitive actions to make progress in the game (like tetris)
- Usually the opposite of strategy because you have to zone-in to the game in strategy, but abnegation is losing yourself and going into a meditative state
- Not thinking much, almost muscle memory
  - Example: PowerWash Simulator

Aesthetic #9: Competition

- Dominance over others
- Almost all PvP games
- Possible to have fellowship and competition: Team vs Team games
  - Example: Super Smash Bros

Summary

The 9 Aesthetics of Games

1. Sensation - Immersive sensory experience
2. Fantasy - Role-playing and imagining alternate lives
3. Narrative – Drama, immersion, world-building
4. Challenge - Overcoming obstacles
5. Fellowship - Social interaction and cooperation
6. Discovery - Exploring, finding, and learning
7. Expression - Customization, user-generated content
8. Abnegation (Submission) - Passing time, losing oneself
9. Competition - Dominance over others

09/05

Superman 64

- What went wrong:
    - Bad flight controls as a core mechanic
    - Unclear objectives
    - Bad collision detection
- Lessons learned
    - Need for responsive and intuitive controls
    - Use playtesting and iteration to refine mechanics
    - Need for clear in-game guidance and feedback

Anthem

- What went wrong
    - Repetitive mission design
    - Lack of meaningful endgame
    - Long server load times
- Lessons learned
    - The importance of post-launch content planning and robust gameplay loops in live-service games
    - Need for performance testing and infrastructure planning

Analyzing menus and controls

- Menus reveal design priorities and what the developers consider important
- Frequently used actions are often mapped to the most accessible buttons
- Adjustable settings reflect design choices and compromises

First-Order Optimal Strategies (FOOS)

- Strategies that provide high power with minimal effort
    - Not always bad, can help new players (bridge the gap between novice and expert players)
- Players naturally exploit these strategies unless forced to evolve
- Competitive games encourage skill development beyond FOOS
    - But single-player and co-op games may break
- If your FOO is cheating out of the aesthetic of the game, it is a BAD FOO

Street Fighter II

- Crouch block defensive strategy
    - Holding down-back to block attacks and only attacking when the opponent makes a mistake
- Created a low-risk playstyle that frustrated aggressive opponents
- How it was fixed
    - Chip damage
    - Grab moves

Fortnite

- Spam building
    - Rapidly building walls and ramps to create an impenetrable defense without engaging in combat
- Provided instant cover with little planning
- How it was fixed
    - Resource caps
    - Build delays

Breakpoints in Games

- Identity where games fail to function as intended
- 2 types of breakpoints
    - Systemic breaks - unintended interactions that exploit game mechanics (ex: infinite money glitch)
    - Technical breaks - bugs that affect gameplay (ex: clipping, camera issues)

Other systems for defining “why we play games”

- Wizards of the Coast Profiles
    - Tammy/Timmy
        - Seeks excitement and big, memorable moments. Enjoys flashy, powerful plays
    - Jenny/Johnny
        - Prefers creativity and self-expression. Enjoys winning with style and unique strategies
    - Spike
        - Focuses on winning at all costs. Enjoys competition nd optimization
    - Melanie/Melvin

        - Engages with game mechanics deeply. Enjoys understanding and exploiting intricate systems
    - Vorthos
        - Values theme and storytelling. Enjoys immersive worlds and rich narratives

Player Mode
- Determine what are your player modes
- A player mode is a way in which the player interacts with the game

Player Interaction Patterns
- Single vs game
- PvP
- Multi indiv vs game
- Unilateral competition (1 vs many)
- Multilateral competition (every man for himself)
- Cooperative play
- Team competition

Objectives
- Anything the player is striving for

Objective categories
- Capture
- Race - race against something
- Alignment
- rescue/escape
- Forbidden act - get someone else to break rules
- Construction - build something
- Exploration

Procedures and rules
- Procedures: actions hat players can take to achieve their objectives
- Rules: define the game constraints and set limits on the player

Resources

- Elements of the game that hold some value and have some notion of scarcity
- Examples:
    - lives/health
    - Time
    - Currency
    - Ammo

Conflict

- Not only defined as player v player
- Conflict can also be player v game
    - Ex: obstacles put up to keep player from the goal
- game/level difficulty is a touch balancing act
    - Hard enough to promote player interest and play, but easy enough that the goal is eventually reachable (usually)

Boundaries

- Can be defined by rules
- Social norms of the game are a type of boundary

Outcome

- If there is a definitive outcome, it's kind of boring
- There must be a "finish" of some kind

Formal elements framework

- These elements provide a framework in which you can begin to formalize your game
- They are not an end-all, be-all list, but certainly should make you consider things that are important to the game

Questions to ask yourself when designing games

- What actions are allowed in your world?
- How will the game world respond to those actions?

Advice: when designing your games in this class...

- Try to limit the procedure, rules, and resources at first

- Once you get the feel for how your basic procedures, rules, and resources interact, adjust one of these in a later level ("riff on a mechanic")
- Introduce the player to the world
    - Draw them into the "magic circle"
    - Don't overwhelm them

<u>09/12</u>

What are the unique aspects that differentiate video games from other types of games?

Event-Driven Programming

- The event-driven paradigm works great for many systems:
    - Desktop apps
        - You type something, press a button, and the program responds
    - Mobile apps
        - You tap on the screen, and it reacts
    - Web applications
        - Works well asynchronously (ex: getting data from a server)
- But is it ideal for games?

Events in Games

- Examples of events in video games:
    - Button press → action triggered
    - Character collision → logic executed
    - Timer countdown → game state changes
- There are some things that happen "behind the scenes"
    - Updating tallies
    - Shuffling the deck
- But not all games are like this
- In Mario, without user input, there are ongoing background processes. One example is the ongoing collision detection and gravity.

How are Video Games Structured?

- Video games are not entirely event-driven
    - Physics, animations, AI behaviors, etc. are updated continuously
    - Even a "static" board game experience has things going on as a video game

- Video games are structured around the concept of a game loop
- For every frame of animation on the screen the game

The Game Loop

- Basically an infinite while loop
- Update and Draw forever until the game ends
- Update:
  - Receive player input
  - Process player actions
  - Process NPC actions
  - Post-process (like physics)
- Draw:
  - Cull non-visible objects
  - Transform visible objects
  - Draw to backing buffer
  - Display backing buffer

Step 1: Player Input

- Input is typically polled during game loop
  - What is the state of the controller?
  - If no change, do no actions
- However, we can only read input once per game loop cycle
- But good frame rate is short and most events are longer than one frame

Step 2: Process Actions

- Remember you are working in fractions of a second
- At this split second, what do you need to do?
- Consider: Pressing A to jump
  - Is this the first frame in which the jump button is pressed?
- Does this action create an interaction?
  - Is there a collision with an object? Resource? NPC?
  - Did you create a new object with the action?
  - These interactions are added to the current game state

Step 3: Process NPCs

- An NPC (Non-player character) is anything that has agency in the world that isn't you
- NPCs are driven by AI or simple rules, not by player

Step 4: Process the World

- Physics
- Lighting
- Networking
- And more

Step 5: Prepare to Draw

- Only draw what is actually on screen
- Drawing tings off screen takes up WAY too much processing for the time frame you have to work with
- Decide how the player is viewing things (angle, depth, etc)
- Painter's algorithm: draw from background to foreground (map to sprite)
    - Whatever is drawn last is what is in front

Step 6: Pre-draw to buffer

- Video cards/drivers have robust buffering
- Draw all the pixels of the frame to a temporary buffer
- When the buffer is full/ready, swap it with the current frame on screen
- This reduces "screen tearing" as much as possible
    - Screen tearing can occur with information from 2 or more frames gets intermingled

<u>09/17</u>

Procedures and Rules

- Procedures - actions that players can take to achieve their objectives (basically dynamics in MDA)
- Rules - define the game objects and set limits on the player procedures
- Example: Chess
    - Procedure: the player can move the knight
    - Rules:
        - The knight must move in L-shape

        - Must be player's turn

Procedures

- At their most basic, procedures map to the input device you are using
- You will fall into one of a few categories:
    - Gamepad
        - Controller input device with a specified set of directional and interactive command buttons
    - Mouse
    - Keyboard
    - Combination

Controller complexity

- Complex control schemes can overwhelm new players, but appeal to experienced ones
    - Know you target audience to find balance
- Example: Nintendo Wii
    - Simple motions, fewer buttons
    - Accessibility through simplicity
- Creativity comes out of limitations

Actions vs Interactions

- Action - a procedure that is mapped to a control input
    - Ex: jump, move, run, shoot, slide, etc
- Interaction - an outcome of the game state
    - - may not be the result of a direct action from the player
    - Can happen without any input
    - Ex: collisions, line of sight, resource change

Game mechanic

- Game mechanic - the relationship and combination of any number of actions and interactions
- Each relationship/combination could be considered a separate rule in the game world
- Ex: Super Mario Bros
    - Bottom of Mario + top of Goomba = goomba dies

Designing actions

- Start by brainstorming verbs that make sense in the world you are building
  - Define the types of verbs
  - Define the scope fo the verbs
- Do the verbs directly help the player achieve the goal?
- How many verbs do i need?
  - Enough tot avoid being too simple
  - Too many clutters the game
- Some of the most successful games are built around just one or 2 verbs
- Ex:
  - Flappy bird: flap
  - Pac-man: move, eat
  - Tetris: rotate, move

Primary vs secondary verbs

- Q: Imagine you had no obstacles/challenges in a game, what verbs would you actually need?
- Example platformers
  - Goal: reach the exit
  - Movement is only verbs you need
  - Killing enemies is secondary
- Primary actions - things you must do to overcome obstacles and complete the objective
  - They are the ore of the gameplay loop
  - Need to feel perfect
- Secondary actions - enhance gameplay but aren't required to complete the core objective
- Concentrate on the primary verbs
  - Too many secondary verbs leads to game bloat!

Finding good verbs

- Keep number of verbs to minimum, utilize interesting interactions
- Avoid verb proxies
  - Use an item → what is it doing?
  - Shoot → what does the weapon do?
- Use outcome-oriented verbs

- Does the bern help the player reach the goal?

Combining actions

- Verbs can combine in interesting ways
    - Super mario bros - run and jump
- Q: How does Mario's jump mechanic change based on the environment?
    - Example: land vs water

Emergent behavior

- When simple mechanics combine to create complex or surprising gameplay that wan't explicitly programmed
- Ex: legend of zelda breath of the wild: arrow + fire = fire arrow
- Not all combos are emergent
    - Ex: combo moves in fighting games that are explicitly programmed

<u>09/19</u>

Interactions

- Specifically NOT the direct action of a player
    - Outcome of the game state
    - Can happen without player input
- Ex: collisions
    - Can happen by player movement OR can happen by game state changing

Procedures vs rules

- Rules are formal schemas
- Operational Rules
    - Rules of the game as if you were explaining them to a friend
    - "In Mario, you can run and jump and land on top of goombas and they die"
    - The instruction book approach to rules - highest level of abstraction

Constitutive Rules

- Operational rules as understood by the game system itself
- A goomba dies only if the bottom of Mario's sprite collides with the top of the goomba's sprite
- This is how the game is actually programmed

Implicit rules

- Agreed upon rules of a game that are not part of the formal rule set, but are important to make the game work
- For instance, a time limit on a move on a board game - not an actual time limit, but you know when someone is taking too long

Designing good rules

- Should lead players to interesting choices
  - Player MUST be able to make some decisions
  - System MUST respond and give feedback
- Bad rules
  - Pure luck based
  - Lack of interaction
  - Doesn't relate to goal

Mechanics vs Rules

- Mechanics are created in the framework of rules
- Dynamics are created by players as interpretations of mechanics within the rules
- Rules are the formal implementation of the game world
- You design mechanics AROUND those rules

Depth vs Complexity

- Depth - the amount of meaningful choices that come from the gameplay experience
- Complexity - the cognitive load of the player (and all the things that go into this)
- Goal: high depth with low (or appropriate) complexity
- How does this relate to procedures and rules?

Graphics vs Visual Design

- Sometimes you don't have to have AMAZING graphics
- You need to have the RIGHT graphics
- Tetris wouldn't be "better" with sweet bump mapping
- Just because we're using 2D, that doesn't mean you're being held back

Back to sprites

- Sprites are an abstraction of all graphical content

- Definition of sprite may differ
    - A class with movement, collision, etc (in some engines)
    - A simple graphic reference (in others)

Drawing sprites

- Typically, you want to draw back to front
- But just because you draw something, it doesn't mean it's necessarily going to be seen
- We care about three coordinate systems at the same time:
    - Screen
    - World
    - Object

09/24

Scrolling

- What moves?
    - Sprite? Camera? World?
    - Each uses different coordinate math
- Off-screen objects:
    - Drawing them? The engine won't display them, but…
    - They still consume resources (memory, processing power)
    - Every frame: position updates, physics checks still happen
- Impact on game loop:
    - Managing and updating off-screen objects = extra work
    - Optimize how you create/destroy objects

Parallax scrolling

- Multiple layers of background, scrolling at different speeds to give the illusion of depth

Types of cameras - orthographic

- Orthographic
    - Perpendicular to the plane of the game
- Fixed along one axis
    - Z-axis: top down perspective
    - Y-axis: side scroller

    - X-axis: vertical scroller
- Handy for artists
    - Art is done as tiles and easy to manage
- Forces 2D gameplay only

Types of cameras

- Trimetric
- Dimetric
- Isometric

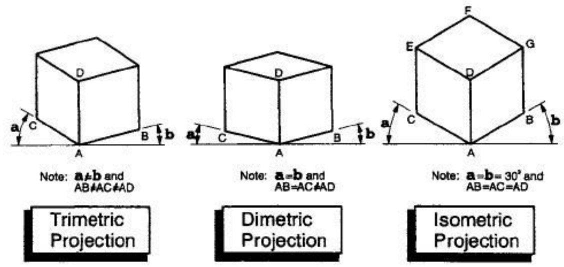

Figure 2.4 *The three types of axonometric projections*

Game physics is hard but doable

- Physics as a whole is complex
    - Imagine mario jumps. What's happening?
- Game physics doesn't have to be realistic to be fun
- Do not reinvent the wheel when it comes to physics
    - It depends
- Ex: Portal
    - Standard physics engines won't work for unconventional mechanics
    - Custom physics logic may be required

Kinematics vs dynamics

- Kinematics → motion (position, velocity, acceleration)
    - Games use kinematics for control and predictability

The role of physics

- Detect collisions: responsive forces
- Accumulate forces: equations of motion
- Numerical integration: updated world at delta t in the future
- Enforce constraints: adjust world to maintain user constraints

How do you accomplish this?

- Roll your own
    - Use real physics, or make up your own
    - Unusual physics can be a game mechanic
- Breakout example

    - What calculations are needed?
    - How do values change per frame?
- Nerdy teachers platformer example
    - Uses acceleration, friction, gravity - that's it!
    - Smooth movement = better control

Pre-built physics engines

- Unity
- Advanced engines
    - Havoc
    - physX
- Open source libraries
    - Chipmunk
    - Farseer physics

Newton's laws

- Newton's laws in games
    - First law: objects stay in motion unless acted upon
    - Second law: force = mass x acceleration
    - Third law: every action has an equal and opposite reaction
- Players expect these rules to apply, even if they don't realize it

What if there are no collisions?

- Simpler calculations
- Pick your forces

Physics with collisions

- Force directly changes acceleration
- Bigger mass => bigger force
- Force puts things in motion, but also can bring things to a halt

Momentum and impulse

- Momentum (P) - the force that keeps an object moving
    - P = m*v
- Impulse (delta p) - change in momentum
    - Delta p = Fj* delta t

Coefficient of restitution (COR)

- COR - measures an object's elasticity in collisions
  - Elasticity - how much an object regains its shape after impact
  - 0.0 → inelastic (no bounce, energy absorbed)
  - 1.0 → elastic (perfect bounce, momentum conserved)
- Higher COR = more rebound

Has a collision occurred?

- How do we know when a collision actually has happened?

Creating hitboxes

- Goal: "good enough" precision
- Use bounding boxes/spheres for raster checks
- Can be layered for more precision
- Hitbox (attack zone) - deals damage
- Hurtbox (vulnerable area) - can be hit
- collision/pushbox (physical space) - prevents overlap

Bounding boxes

- Axis-aligned vs object-aligned
- Axis-aligned bounding box change as object moves
  - Approximate by rotating bounding box

Overlap testing

- Most common method of collision detection
- For each delta t, check if objects overlap

Finding the moment of collision

- Using sub-stepping to detect the exact moment of collision

Problems with overlap testing

- What happens if delta t is too big?
- Fails with objects that move too fast

Raycasting (swept collision detection)

- Predict future collisions

- How it works
    - Extend hitbox along movement path
    - Sphere biomes a capsule for detection
    - Move to impact point, resolve collision, continue

Problems with raycasting
- Issue with networked games
    - Future predictions rely on exact state of world at present time
    - Due to packet latency current state not always coherent
- Assumes constant velocity and zero acceleration over simulation step
    - Has implications for physics model and choice of integrator

Determining hitbox collisions
- Collision detection complexity
    - Worst case: O(n^2) (checking every object against every other)
- Optimizations for speed
    - Sort objects in one dimension (bucket sort)

Cheaper distance tests
- Avoid expensive square root calculations
    - Compare d^2 instead of sqrt(d^2)
    - Use manhattan distance for an approximate fast check
- Formulas for faster comparisons
    - Squared distance
        - D = sqrt((x1-x2)^2 + (y1-y2)^2)
    - Manhattan distance (cheaper alternative)

Achieving O(nlogn) complexity
- Plane sweep geometry
- Partition space algorithm

Collision resolution
- What do you do after a collision?
- 3 parts: prologue, collision, epilogue
- Prologue
    - Collision known to have occurred

    - Check if collision should be ignored
    - Other events might be triggered
        - Sound effects
        - Send collision notification messages

Collision

- Place objects at point of impact
- Assign new velocities
    - Using physics or some other decision logic

Epilogue

- Propagate post-collision effects
- Possible effects
    - Destroy one or both objects
    -