Game def: goals, rules, feedback, voluntary participation

MDA

First order design: rules, mechanics, systems

Second order: player experiences (aesthetics) by manipulating mechanics/dyn

- ex: 1st=puzzle piece rotation mechanic
- 2nd=players feel accomplished when solving puzzle

Goal of MDA: Bridge game design and game analysis

Mechanic: basic rule or action available to the player (rules and systems of the game)

- ex: press A to jump, hold B to run

Dynamic: behavior of the system during gameplay (game in motion). Actions taken by players that arise from engaging with the game's mechanics

- ex: chess.
- Mech=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: emotional responses define the feel of a game; mech + dyn = aesthetics

Approaches to game design

1) mechanics-first: paraglider in an obstacle course vs narrative adventure
2) aesthetics-first: designing for exploration, storytelling, or player expression

Aesthetics

1. Sensation: immersive sensory experience, memorable AV fx (just dance)
2. Fantasy: role-playing/cosplaying and imagining alternate lives (FIFA)
3. Narrative: drama, immersion, world-building (Skyrim)
4. Challenge: overcoming obstacles (super meat boy)
5. Fellowship: social interaction and cooperation (among us)
6. Discovery: exploring, finding, learning (Minecraft)
7. Expression: customization, user-generated content (the sims)
8. Abnegation (submission): passing time, losing oneself (powerwash simulator)
9. Competition: dominance over others (smash)

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

- strategies that give high power with low effort
- Players naturally exploit these strays unless forced to evolve
- Competitive games encourage skill development beyond FOOS

Street Fighter II

- crouch block defensive strategy created a low-risk playstyle that frustrated aggressive opponents
- Fixed with chip damage and grab moves

Breakpoints in games

Identify where games fail to function as intended
2 types of breakpoints:

- systemic - unintended interactions that exploit game mechs (infinite money glitch)
- Technical - bugs that affect gameplay (clipping, camera issues)

Fullerton's Ludicrous structure of games

Players - relationship between players and the game
Objectives - goals, both designer-driven and-driven
Procedures - actions of play, limited by rules
Rules - define game params, part of the "magic circle"
Resources - valuable objects as defined by rules
Conflict - hindrances to objective
Boundaries - limits of the game world
Outcome - uncertainty toward an ultimate goal

Player mode

A way in which the player interacts with the game
Ex: combat mode = engaging in battles or challenges

Wizards of the coast profiles

1. Tammy/timmy - seeks excitement and big, memorable moments. Enjoys flashy, powerful plays
2. Jenny/johnny - prefers creativity and self-expression. Enjoys winning with style and unique strategies
3. Spike - focuses on winning at all costs. Enjoys competition and optimization
4. Melanie/melvin - engages with game mechanics deeply. Enjoys understanding and exploiting intricate systems.
5. Vorthos - values theme and storytelling. Enjoys immersive worlds and rich narratives.

Bartle's 4 types of online games

Achiever - overcome challenges, gather rewards
Explorer - discover, understand game worlds
Socializer - interact and roleplay
Griefer - distress other players in the game

Formal elements

Elements easy to define early (players, primary objectives, outcomes, boundaries)

Player interaction patterns

- Single player v game (Solitaire)
- Player vs player (chess)
- Multiple players vs game (bingo)
- Unilateral competition (Scotland Yard- 1 v all)
- Multilateral competition (monopoly)
- Cooperative play (overcooked)
- Team competition (team v team)

Procedures and rules

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

Resources

Elements of the game that hold some value and have some notion of scarcity (lives, health, time, currency, ammo)

Conflict

Not only pvp, also pvg

- hard enough to promote player interest and play, easy enough that goal is eventually reachable

Boundaries

Can be defined by rules, but also by the nature in which the game is being played
Types of boundaries

- physical (edges of map)
- Conceptual (defined by the games rules and mechanics)
- temporal (time-based constraints on when the game begins/ends
- Psychological (everyone agrees to enter the "magic circle"
- Social (multiplayer interactions and social norms)

Outcome: must be a finish of some kind, uncertain while game is played

Objectives

- anything the player is striving for
- Primary - main objective of game
- Secondary - achievements, high score, etc
- Player driven - self created goal

Categories

Capture - take or destroy something
Race - race against something
Alignment - perfect positioning
Rescue/escape - get out of dungeon
Forbidden act - get someone to break rules
Construction - build something
Exploration - uncover all hidden things

Event-driven programming

Paradigm where program behavior is triggered by events like user actions or system signals

- works well with desktop apps, mobile apps, web apps

Events in games: button press, character collision, timer countdown

Video games are not entirely event-driven, things are updated continuously

- structure around a game loop
- For every frame of animation on the screen the game must update and draw the game state

Update: receive player input, process player actions, process NPC actions, Post-process (physics)

Draw: cull non visible objects, transform visible objects, draw to backing buffer, display backing buffer

Game state: comprised of the current values of all variables for all objects in the game world at that precise moment. Snapshot of the game

Game loop: update=get all input from the user, making calls based on rules/ procedures/actions, changing all the values in the world that need to change.

Draw=put new game state onto screen

Step 1: Player input
Typically polled during game loop, but can on,y read input once per game loop cycle
Step 2: process actions
Collision? New object created? Add to current game state
Step 3: process NPCs
Anything that has agency in the world that isn't you. Sense-plan-act: what can it see? How smart? Computer can move things perfectly, do we want that?
Step 4: process the world
Physics, lighting, networking, etc
Step 5: prepare to draw
Only draw what is actually on screen
Step 6: pre draw to buffer
Video cards/drivers have robust buffering. Draw all the pixels of the frame to a temporary buffer. When buffer is full/ready, swap it with the current frame on screen. Reduces screen tearing
Step 7: swap buffer to screen
This step is rarely done by a game designer with a modern game engine

Procedures and rules

Procedures - actions that players can take to achieve their objectives
Rules - define the game objects and set
Ex: chess - procedure: the player can move the knight.
rules: knight must move in L-shape, must be players turn
Procedures map to the input device you are using
Controller complexity

- complex control schemes can overwhelm new players, but appeal to experienced ones

Actions vs interactions

Action - procedure that is mapped to a control input: jump, move
Interaction - outcome of the game state, may not be the result of a direct action from the player. Can happen without any input. Collisions, Line of sight, resource change
Game mechanic - 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. Actions= run left and right, jump
interactions= collision with opponent
rule= if collision is on top of enemy, enemy changes state according to rule set, otherwise take damage according to rule set

Designing actions

- verbs ; enough to avoid being too simple, too many clutters
- Primary =things you must do to overcome obstacles and complete the objective - core of the gameplay loop. Need to feel perfect
- Secondary - enhance gameplay but I red to complete the core objective
- Avoid using verb proxies (use an item = what does it do?). Use outcome oriented verbs

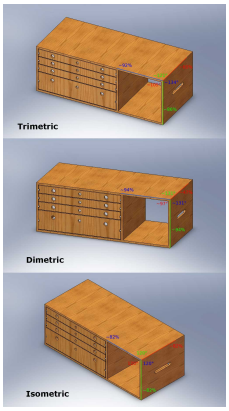
Emergent behavior
When simple mechanics combine to create complex or surprising gameplay that wasn't explicitly programmed (arrow + fire)
Interactions
Specifically NOT the direct action of a player. Outcome of the game state. Can happen without player input. (Collisions)
Procedures vs rules
Rules are formal schemas. In general, 3 types of rules:
- Operational: English rules of a game as the player understands it
  "in Mario, you can run and jump and land on top of goombas and they die".. very high level
- Constitutive: underlying math and logic behind operational rules
  goomba dies only if the bottom of the Mario's sprite collides with the top of the goombas sprite
- Implicit: extra rules understood by the players to make the game move forward
  agreed upon rules of a game that are not part of the formal rule set but are important to make the game work (time limit)

Designing good rules
- should lead players to interesting choices. Players must be able to make decisions and 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 by game designers 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.
Depth vs complexity
Depth=amount of meaningful choices that come from the gameplay experience
Complexity = cognitive load of the player
Goal is high depth with low complexity
Graphics vs visual design
You need to have the right graphics, not the best
Sprites
Abstraction of all graphical content
Common sprite components: image/texture, position, animation frames, collision geometry, additional game-specific data
Sprites and sprite sheets
Asses management is important for any game. It's impractical to save every single frame of animation in a separate file. A sprite sheet is a single image, broken up in a regular pattern where all the images needed for animation are stored.
Animation
sprite sheet is split into frames and then sets of frames are assigned to various actions
Drawing sprites
Draw back to front to create proper layering of objects. 3. Ordinate systems at the same time: screen, world, object
Scrolling
- What moves? Sprite, camera, world? Each uses diff coord math.
- Offscreen objects: drawing them? The engine won't display them but they still consume resources. Every frame position updates, physics checks still happen.
- impact on game loop: managing and updating offscreen objects is 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
- perpendicular to the plane of the game
- Fixed along one axis: Z=top down, Y=side scroll, X=vertical scroll
- Handy for artists: art is done as tiles and easy to manage
- Forces 2D gameplay on,y.

Axonometric
- off axis view, allows for some 3D gameplay. Isometric is most common form (all axes equal). Can have other forms by shortening/lengthening some axes for artistic reasons.



Trimetric, diametric, isometric cameras

Raycasting (swept collision detection)
Predict future collisions. Extend hitbox along movement path, sphere becomes capsule for detection. Move to impact point, resolve collision, continue

Problems with raycasting
Issue with networked games: future predicts 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 hit box collisions
Collision detection complexity - worst case is O(n^2) checking every object against every other.
Optimizations for speed: sort objects in one dimension (bucket sort)
- use temporal coherence (track nearby object pairs)
- Estimate likely collisions based in speed
- Use simpler, cheaper tests first

Kinematics vs dynamics
Kinematics = motion (position, velocity, acceleration). Games use Kinematics for control and predictability
Dynamics = forces (gravity, friction, collisions). Fake dynamics to make movement feel real
Physical simulation in games
- Goal is believable physics. Good physics enhances gameplay and creates new dynamics. Game physics != real world physics
- role of physics: detect collisions, accumulate forces, numerical integration, enforce constraints

Newtons laws in games
1. objects stay in motion unless acted upon
2. Force = mass x acceleration
3. Every action has an equal and opposite reaction

Good physics feels natural and when not realistic

Variables
t_x = final time for the period we are examining
t_0 = initial time
Δt = change in time ($t_x - t_0$) for period
Δv = change in velocity ($v_x - v_0$)
Δa = change in acceleration ($a_x - a_0$)
Change in Velocity

$v_t = v_0 + \frac{F}{m}\Delta t$ or $v_t = v_0 + a\Delta t$, you choose Δt

Velocity at a given time is the sum of initial velocity and change due to acceleration

Change in Position : $P_t = P_0 + V_0\Delta t + \frac{F}{2m}\Delta t$

Position at a given time is the sum of initial position, position change due to initial velocity, position change due to acceleration

Rigid body physics: more complexity
- Objects have mass and volume, not just position
- Center of mass determines movement
- forces can cause rotation (torque)
- More calcs needed for realistic physics

Problem with time step physics
If delta t is too big: imprecise physics, no smooth trajectories
If delta t is too small: processor can't handle it
If everything happens in a single frame: all variables snap to next state

What if there are no collisions?
Simpler calcs; pick your forces: gravity, air resistance, the force. Computer acceleration and velocity. Update position, draw the next frame.
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=F*delta t
Law of conservation of momentum
Momentum is conserved in collisions. Total momentum before=total momentum after. Direction may change, but magnitude remains constant. Valid if no external forces act on the system.
Coefficient of restitution (COR) - measures an objects elasticity in collisions (how much an object regains its shape after. 0=inelastic (no bounce, energy absorbed). 1=elastic (perfect bounce, momentum conserved)
Hitboxes
Goal is good enough precision. Use bounding boxes/spheres for faster checks. Can be layered for more precision. Hit box (attack zone) - deals damage. Hurtbox can be hit, collision/pushbox (physical space) - prevents overlap
Bounding boxes
Axis aligned vs object aligned. Axis aligned bounding box change as object moves.
Spherical hitboxes - put everything in a bubble (cheap to detect)
Overlap testing - may cause minor errors (false positives/negatives). For each delta t, check if objects overlap. Optimizations reduce performance issues.
Finding the moment of collision
Use sub-stepping to detect the exact moment of collision.
Problems with overlap testing
What is delta t is too big? Fails with objects that move too fast. Unlikely to catch time slice during overlap. Possible solutions: design constraint on speed of objects, reduce simulation step size

Cheaper distance tests
- avoid expensive quads root calcs. Compare d^2 instead of sqrt. Use ,angsts. Distance for an approximate fast check (|x1-x2|+|y1-y2|)

Achieving O(nlogn) complexity: Plane sweep geometry
Achieving O(n) complexity: partition space algorithm
Collision resolution
What do you do after you detect the collision? Rocket slams into wall: rocket disappears. Explosion spawned na so suns effect, wall charred.
Resolution has prologue, collision, epilogue
Prologue: collision known to have occurred. Check if collision should be ignored. Other events might be triggered (sound,p fx, send collision notifs)
Collision: place objects at point of impact. Assign new velocities using physics or some other decision logic
Epilogue: propagate post collision fx. Possible fx: destroy one or both objects, play sound, inflict damage. Many effects can be done either in the prologue or epilogue.