Lecture 9
Tue Feb 7 Lecture

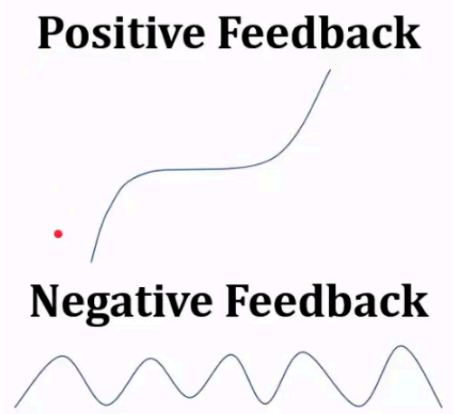
Recap - Game Economies:
Dynamics are about emergent properties, multiple mechanics come together to create a dynamic. We try to create dynamics by carefully selecting different dynamics. One way to do this is through scarcity and utility. With resource flow, we think about how the player will experience scarcity and utility, and how that will change their playing. There are types of resource rewards like rewards of sustenance (i.e. health packs). Sources are when new resources are created in the game, sink is when resources are removed from the game, an adapter takes a resource and converts it into another resource (be strict about what you call an adapter, don't be too loose with the definition). We talked about how we can vary each resource through source, storage, sink, convert, and transport (through resource lens). Not necessarily useful for game balance, or the game in its entirety.

A good resource about game balance is "Game Balance" by Brenda Romero and Ian Schreiber.

Feedback Loops:

Two main types of feedback dynamics:

1) Positive feedback - not equal to "good" or "up"
    - The more you get, the easier it becomes to get more
    - Losing makes it easier to lose more
    - Exs in real life: social media views, stocks
    - Exs in games: crash team racing, cookie clicker, killstreaks in shooters
2) Negative feedback - not equal to "bad" or "down"
    - The more your succeed, the harder it will get
    - The more you lose, the easier it will get
    - You are always being brought back to the "middle point"
    - Exs in real life: progressive tax brackets, repelling magnets
    - Exs in games: mario kart, skill-based matchmaking



The shape refers to resource flow.

Why include positive feedback?

- Player feels like they're advancing
- Tend to be more skill-based so attracts certain types of players
- Ends "dominant" games quickly (games where there's going to be an obvious winner and you're just waiting for the game to end)
- Positive feedback hooks players

What can go wrong?

- Hard to succeed if you're not good at the game
- Hard to balance positive feedback
- Resource gathering could get too fast

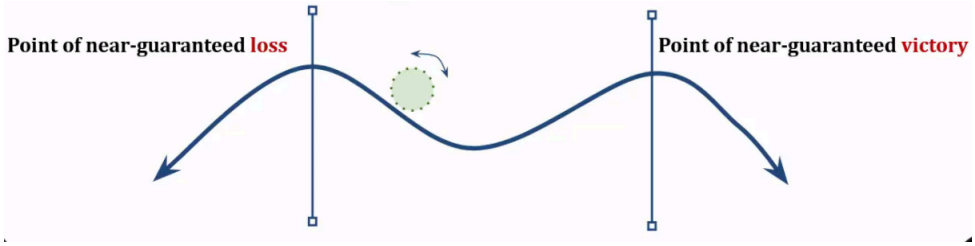
Why include negative feedback?

- Closes gap between players of different skill levels
- Can lead to dramatic events (if someone is about to win and then they don't)
- Game give players a fighting chance to win

What can go wrong?

- Can draw out a game
- Makes experienced players feel artificially weakened (i.e. being in first place in mario kart and getting blue shelled over and over)
- Enforces a specific way of playing

You can mix and match positive and negative feedback to keep players fighting but ultimately lead the game to ending before it draws out.



Player bounces between loss and win points until they eventually "roll over to one side" and win/lose.
Example: trading board game where you exchange resources. Middle area where resources are not very valuable. Then when you build a structure that increases production, you win the game.

Loop priorities: (not tested, don't need to memorize)

- Magnitude: how much will feedback affect the game
- Immediacy: how much time until we see returns of positive feedback
- Investment cost: how much we need to invest to get positive feedback
- Shocks: any disruptions to the feedback loops that change its dynamic
    - Ex: everyone suddenly needs to give away one resource

Feedback loops in practice:

- Decide when players have “won enough” to finish the game
- Simulate expected progression using something like a spreadsheet
- Design with feedback loops in mind
- If the game feels like it’s “dragging on”, inspect the feedback loops

Example of negative feedback in stardew valley: the more buildings you place, the less space you have for growing crops/doing other things

Aside: take a look at
https://www.gamedeveloper.com/design/-quot-the-door-problem-quot-of-game-design

Understanding Randomness:
How do different mechanics of randomness lead to different dynamics?

Why randomness?

- Surprise
- Diversity of play
- Sense of thrill and anticipation
- Provides obstacles

Cons of randomness”

- Frustrating
- Bland
- Removes skill

Deterministic system: values of current state determine the values of the next state

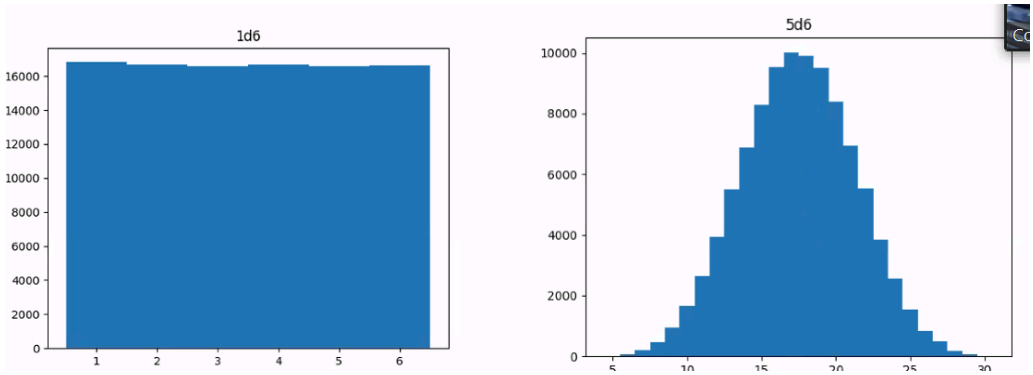
- Tic tac toe
- Mostly deterministic: single playthroughs of games like gone home b/c player choice determines their experience

Stochastic: pure randomness - values of current state cannot determine the values of the next state

- Ex: war, snakes and ladders
- Mostly stochastic: solitaire
-

More randomness: usually more casual play
More deterministic: usually more strategic play

Variation
Sometimes, more randomness leads to less variation.



1 die roll vs 5 dice roll

https://www.tumblr.com/galaxykate0/139774965871/so-you-want-to-build-a-generator
"So your algorithm may generate 18,446,744,073,709,551,616 planets. They may each be subtly different, but as they player is exploring them rapidly, will they be *perceived as different*? I like to call this problem the **10,000 Bowls of Oatmeal** problem. I can easily generate 10,000 bowls of plain oatmeal, with each oat being in a different position and different orientation, and *mathematically speaking* they will all be completely unique. But the user will likely just see *a lot of oatmeal*. **Perceptual uniqueness** is the real metric, and it's darn tough." - Katee Compton

Mechanical variation - ways to achieve variation:

- Draw from a set without replacing
- Never allow same option twice in a row
- Draw from different random pools

Pre-luck/input randomness: act after luck has occurred "here is the game state, what do you want to do with it?"
Post-luck/output randomness: luck after acting; take action, then result occurs

- Abstract reality
- Promote the unknown
- Stumble strategies
- Unexpected failure and strategies falling flat are often results of this, and don't make the player feel great. Imagine shooting the final boss and missing because of RNG.
- A new trend is to only use post-luck positively. Minimum amount of success is guaranteed and additional benefits are randomly generated.

- Ex: You cannot miss in Mario + Rabbids

Perceptions of Probability:
Humans suck at analyzing probability.
Example: Monty Hall Problem

Behind door 1	Behind door 2	Behind door 3	Result if staying at door #1	Result if switching to the door offered
Goat	Goat	**Car**	Wins goat	**Wins car**
Goat	**Car**	Goat	Wins goat	**Wins car**
Car	Goat	Goat	**Wins car**	Wins goat

Designers often lie about probability.

- 90% hit might be closer to 98%
- 50-80 damage may skew to 70-80% damage most of the time
- It's ok to skew things in the player's favour
- https://www.raphkoster.com/2013/04/16/playing-with-game/

Note: usually game design interviews ask this question: "how can we change ____ to get _____ dynamic"?

Lecture 10
Feb 9 Lecture

Today we introduced Godot and started following the Dodge the Creeps tutorial.

A Brief Intro to Godot

- 2D and 2D rendering, sound management
- Physics, collisions
- Scripting language, visual scripting
- Input mapping
- Level editing tolls, 2D and 3D tilemaps
- Networking, multiplayers
- Keyframe animation, inverse kinematics
- User interface tools (ex: tesla uses godot in their cars)
- AR/Vinterfaces
- Scripting behaviour
    - Path finding, visibility detecting, raycast, timers
- Work in GDScript or C# or any other language withGDNative
    - Also work in external editors, like VS Code
- Engine available on Steam or via Godot Website
- Using Godot 3.5x for this course

OOP

- Godot follows object-oriented programming patterns

https://docs.godotengine.org/en/stable/getting_started/introduction/learning_new_features.html

- We think in terms of “things” that have properties, data, behaviors, and other things
- We separate the design of a thing (blueprint) from the instance of the thing (construction)

- A game is a tree
- Each thing in the tree is a node
- Nodes are grouped together in scenes
- Notes communicate with signals

Planning scenes

"We have a character that can run and jump". What do we need to represent this?

- A character sprite → Sprite2D
- Collision with the floor and enemies → KinematicBody2D
- Sound effects when they jump and land → 2 sound nodes
- And on…

Collision Objects

Collision objects detect when other collision objects are colliding with them. They require a collision shape to define the boundaries of the collision.

1) Area2D: detects overlap and emits signals
    - Useful for non-physics collision detections
    - Eg. End of level portals, event triggering, item pickups
2) StaticBody2D: participates in physics, unmoving
    - No dynamic behavior, can not be moved
    - Eg. Walls, floors, boulders that can not be moved
3) RigidBody2D: simulates physics, indirect control
    - Apply forces instead of direct movement
    - Eg. ragdoll characters, bouncy balls
4) KinematicBody2D: code-controlled collisions
    - No physics interactions, manually write collision response
    - Eg. game characters (precise movement)

Lecture 11
Feb 14 2023

Game Frameworks vs. Game Engines

Examples of Engine Code

- How to draw things to the screen
- Detecting hardware input
- Playing sounds
- Performing physics calculations

Examples of Content Code

- How to react to input
- How to move things around
- Visual effects
- Procedural generation
- AI, data

“Engine” does not necessarily refer to Unity or Godot. It refers to the underlying systems behind games.

Many engines, like Unreal Engine, were not created with the purpose of being a publicly-available engine. It was created for a game but then released for the public afterwards.
Unreal Engine is great for creating first-person shooters. Puzzle games… not so much. Important to take this into account when choosing an engine.

Game engines have an “opinion” on what game to make, which changes your workflow. Game Engines are becoming more general though.
Ex: Godot has “opinion” that you should code in OOP

You may want to program an engine with specific functionalities needed to reuse code to create more content faster for your particular game.
Ex: a card game engine that shuffles your hand really well

You may also want to modify just the physics engine in Godot.
Ex. Sonic Colours was created in Godot, but the developers heavily modified the source code of the game engine to suit their game.

Frameworks

Separate from engines b/c it doesn't provide us with tools.

Game frameworks are programming libraries which provide us with lots of the necessary functions for essential game functionality.

- Drawing to the screen
- Playing sound effects
- Detecting user input
- Basic user interface elements
- Basic physics/collision detection

Starting with a framework rather than a game engine requires a lot more work.
We can use a framework to create an engine custom-made for our specific game.

Pygame is a framework. So is Processing. They let us make quick and simple prototypes.

Frameworks have little to no opinion on what kind of game you want to make.

Might pull in basic frameworks for drawing, physics, collisions to create an engine.

Frameworks
Pros:

- Supports experimental games
- Lots of control
- Little influence on design
- Very powerful for a CS student since using frameworks is very similar to programming

Cons:

- Big learning curve
- Tough to do anything complex
- Long time to get engine built

Generic Engines:

- Easy to get started with
- Able to make complex games

- Heavily supported, easy to release game to people

Cons

- Can (and should) influence your design
- Usually have licensing fees
- Generic implementations may not work for your game

Most large companies will start with an existing engine and modify the underlying code to work best with their specific game.

Many CS students end up getting jobs developing game engines.
Networking, security, UI, algorithms, are some popular fields.

The Game Loop

3 separate elements:

- Entities, Data: variables, state of objects and resources
    - Position, speed
- Visual Representations: the images rendered to the screen
- Behaviours: the code which reacts to input and things happening in-game

“If you can program numbers being changed, you can program a game.”

Consider a text adventure game without graphics. This is very basic code. You can have the same code for different games which all are rendered differently.

The Game Loop

1. Retrieve each input
2. Update game data
3. Render the world

There may be other parts in this game loop, for example “get AI inputs”, “update physics”, “render network”.

Each iteration of the loop is a frame. We refer to processing phases as “ticks”.
(Ticks managed by CPU, graphics card manages rendering)

Some older games blend ticks with frames. If you have a really good CPU, you move reallllllllly fast. This is why it’s important to differentiate frames and ticks.

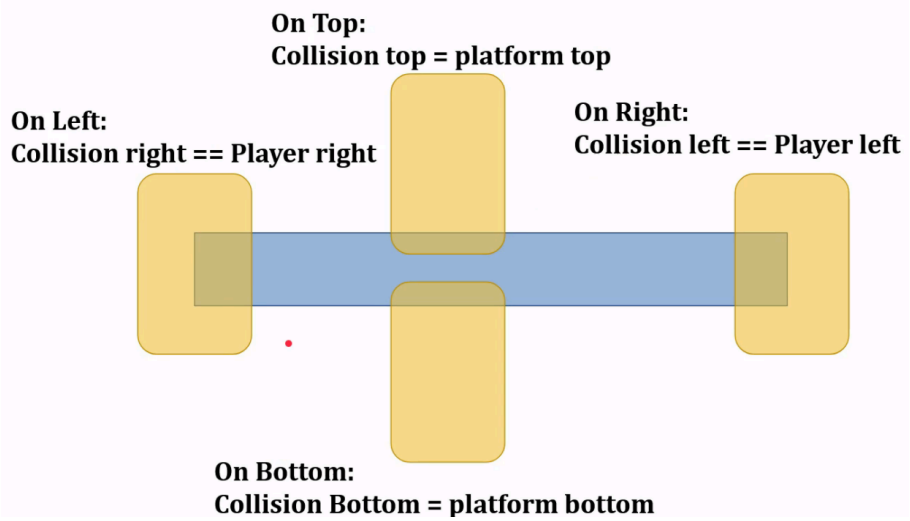
As such, we usually track time between ticks as “delta”.

Collisions → is one shape intersecting with another shape

Writing your own collision functions is not a good idea as a beginner but at the end of this stream it may be feasible for you to write some code that works for your game better than the physics functions provided in a game engine.

Point in a circle: is distance less than radius
Rectangle: is the x coordinate lesser than left side and greater than right side, is y coordinate lesser than top and greater than bottom?

Pygame has a lot of built-in functions. Ex. the “clip rectangle” seen below.



Summary:

- **“Engines” refer to any collection of tools that helps you create your specific game**
- **Generic Engines like Unity and Godot support many games with their tools, but still influence how you make games**
- **Frameworks provide a more “from scratch” feel and allow for more experimentation**
- **All games follow a typical loop:**
  - **Take user input**
  - **Update the data**
  - **Render the data**

Lecture 12
Feb 16 2023

16 - Game Production: Project Management & Communication (will not be tested)

In small teams, the game producer tends to be in charge of everything that isn't actually "making".

- Organize the game features
- Schedule the work
- Resolve blockades
- Advocate the game
- Communicate across the team
- Managing social media
- Keeping morale up
- Scheduling conference visits

This role is extremely important. Games don't get produced without strong production. Having a strong producer can make or break the success of a game.

We will follow a philosophy known as "**Agile Software Development**"
Everyone in the team needs to follow a philosophy in order for this to work.
These techniques are often misused in the software engineering community as a way to rank employees.

In games, we must be flexible. Agile is a philosophy of flexibility and communication.

Agile Manifesto

1. Talking is better than strict formal processes
2. A functioning game is more fun than a thoroughly planned one
3. Work with people instead of against them
4. Change when it will improve the fun

Communicate, stay flexible, try things out.

Scrum
A "Framework" for Agile. A set of practices to remain flexible on projects and to enforce communication and reflection.

Scrum Principles

1. Transparency: keep the work and processes open to all
2. Inspection: watch progress, evaluate goals, detect problems early
3. Adaptation: upon inspection, be open to change

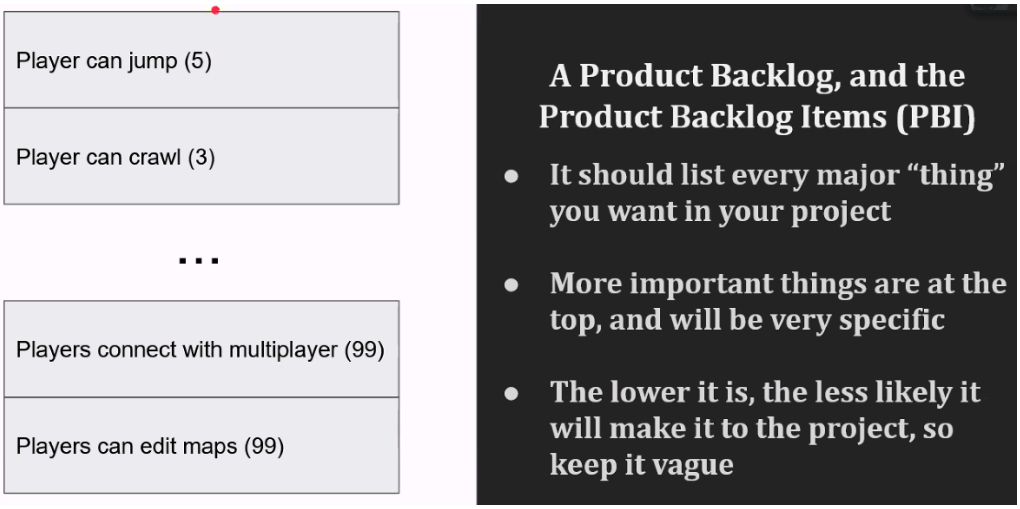
- Values: commitment, focus, openness, respect, courage

https://scrumguides.org/

Agile: a philosophy for how to approach software development projects in a flexible way.
Scrum: A set of tools and processes which help us to work in an Agile way.

Product Backlog; Planning your Game

- All the things we need to get done in the game
- The things that are more fun/more important should be at the top. Prioritize what's necessary.
- Items at top are more specific
- Stuff at the bottom often gets cut due to time constraints. So, by prioritizing the fun and important stuff, you are ensuring that your game is playable.
- Helps everyone get on the same page.

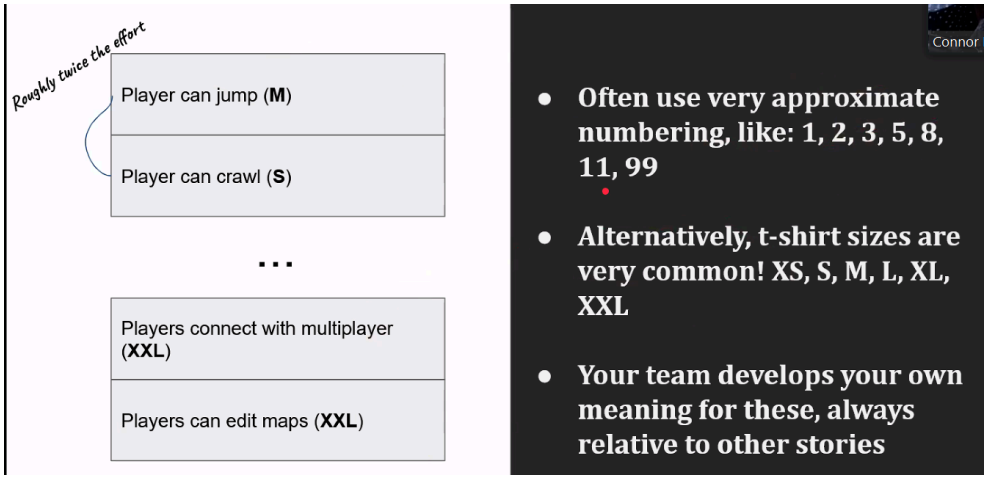


Backlog Items: "As a _____, I want to _____(, so I can _____)."

- Value and Priority
- Promote change and conversation
- Allows details to emerge

“As a player, I want to be able to pick up new weapons so that I have a diversity of combat experiences”

The point is **discussion**.
As a discussion tool, as a team, we estimate the “effort” of an item. The process of estimating brings up a great conversation.



S: small task
M: medium task
XXL: huge task, no way we can get it done

(Another technique is called planning poker)

Each time I mention something, does it promote the three values?

- Does this promote transparency?
- Does this promote inspection?
- Does this promote adaptation?

Scrum is based on the idea that the people doing the work are the people who should determine how the work is done.

The product owner understands the overall vision.

- Are we on track?
- Communicate vision between dev team and higher ups

The dev team understands the work.

- Programming, writing, making art

The Scrum Master understands the process.

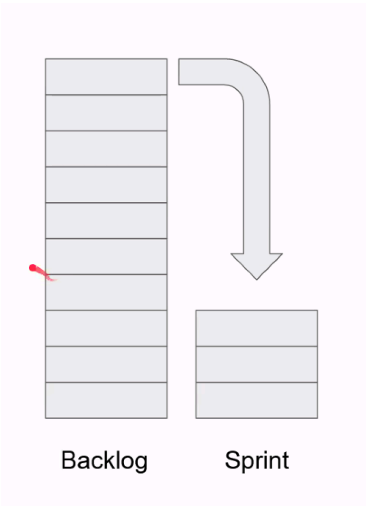
- Someone who specifically understands Scrum.
- Can get Scrum Master certification! 😳

Scrum Meetings; The Flow of a Sprint

4 important meetings in Scrum:

1. Sprint planning (start)
2. Daily Scrum/Daily standup (daily)
3. Sprint review (end)
4. Sprint retrospective (end)
    - Look at team and processes, and try to improve upon them

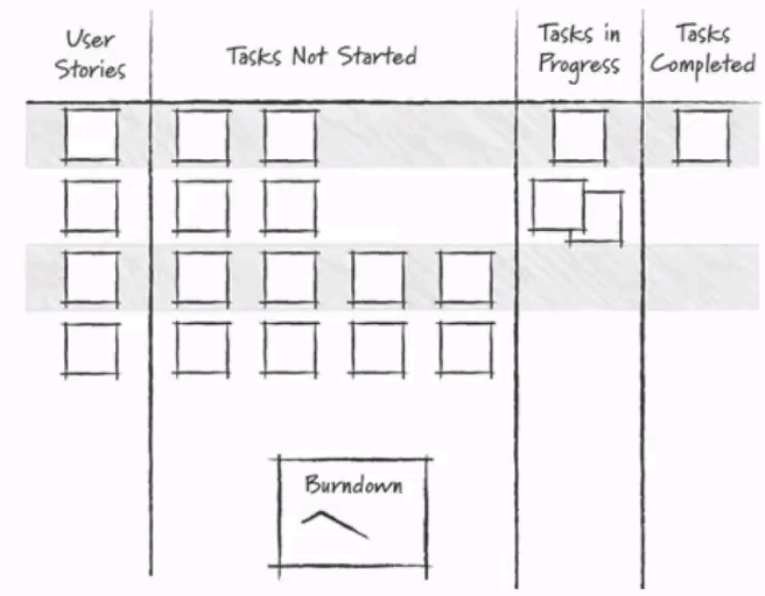
1. Sprint Planning (start)



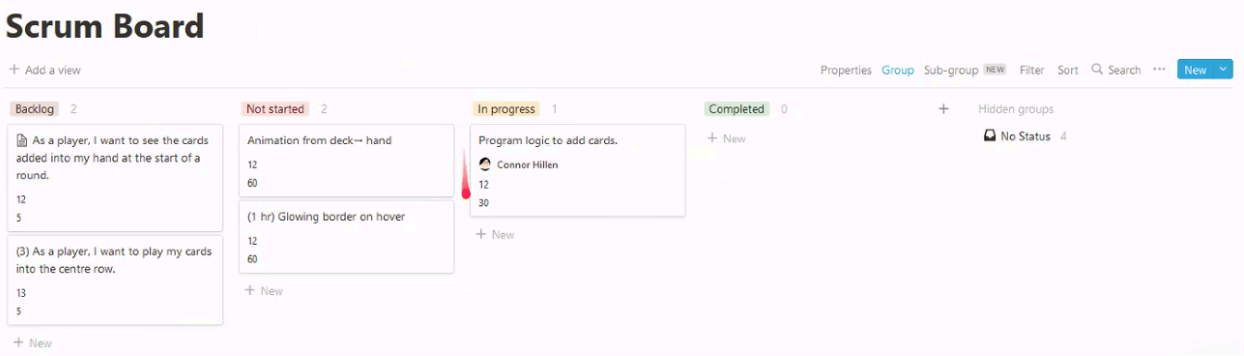
- **The Product Owner decide the hopes for the sprint**
- **We decide which product backlog items we can get done in this Sprint, starting with top priority**
- **We write a tasklist for each PBI**
- **We estimate how the tasks will take, and make sure it can be accomplished**

- How many hours can we work this week? Subtract time for meetings, then halve it.
    - Ex: 5 people who can work 40 hours = 200/2 → ~100 hours of work this week.
- Decide which tasks are going to be prioritized. Reprioritize if not feasible.
- Pull the top task off; each team member writes a task list.
- Tally up how many hours the tasks will take, and subtract the number of hours needed.
    - Task takes 10 hours → 90 hours of work left for the week
- Repeat until you run out of hours.

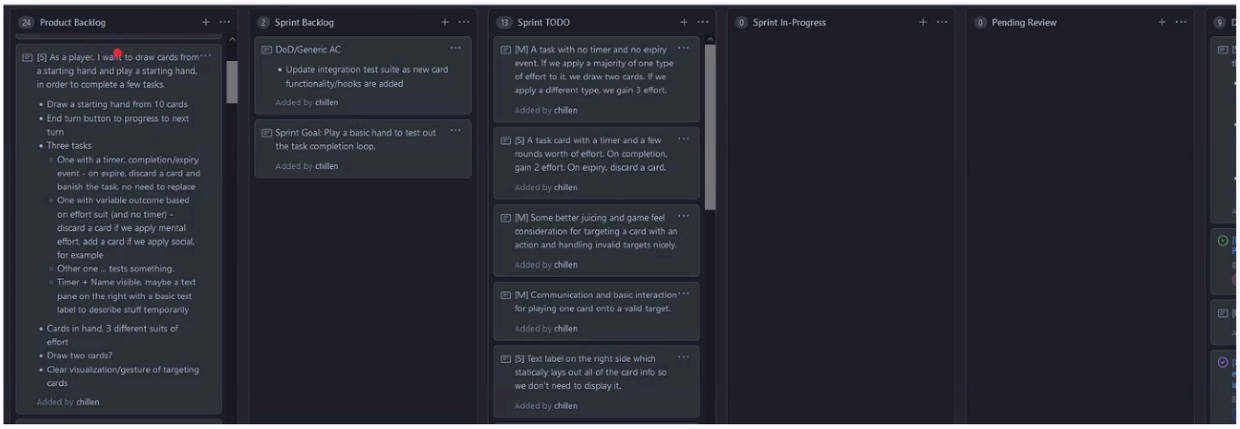
Scrum Board



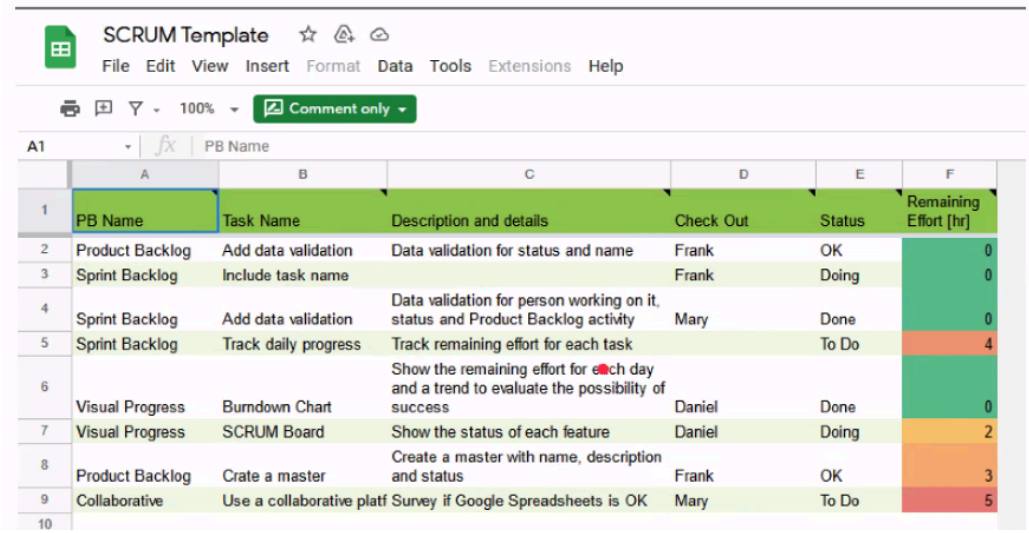
Notion (software) is great for small teams taking notes and maintaining a basic project board.



Github Projects integrates code into project board.



A **spreadsheet** works as well.

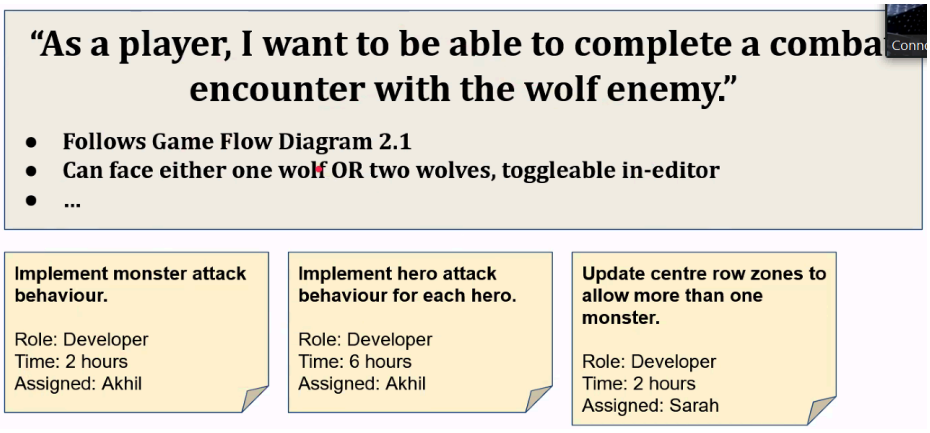


	A	B	C	D	E	F
1	PB Name	Task Name	Description and details	Check Out	Status	Remaining Effort [hr]
2	Product Backlog	Add data validation	Data validation for status and name	Frank	OK	0
3	Sprint Backlog	Include task name		Frank	Doing	0
4	Sprint Backlog	Add data validation	Data validation for person working on it, status and Product Backlog activity	Mary	Done	0
5	Sprint Backlog	Track daily progress	Track remaining effort for each task		To Do	4
6	Visual Progress	Burndown Chart	Show the remaining effort for each day and a trend to evaluate the possibility of success	Daniel	Done	0
7	Visual Progress	SCRUM Board	Show the status of each feature	Daniel	Doing	2
8	Product Backlog	Crate a master	Create a master with name, description and status	Frank	OK	3
9	Collaborative	Use a collaborative platf	Survey if Google Spreadsheets is OK	Mary	To Do	5
10						

Note that **stories (PBI) != tasks**.

Story	Task
• Expresses a feature	• Expresses a work item
• Anyone can understand it	• Meant for a specific discipline
• Has a vague "points" associated with how hard it is	• Has an estimated time-to-complete
• Written by the product owner	• Written by the people doing the work
• Could be as big as a sprint	

Example of a Scrum Board filled out.



You never change what you commit to. You should only change your tasks.

2. **Daily Stand-Up (daily)**

Each day starts with a stand-up. Literally standing up so that's uncomfortable and doesn't take too long. Should be no longer than 15 minutes. Go around in a circle and answer a few questions.

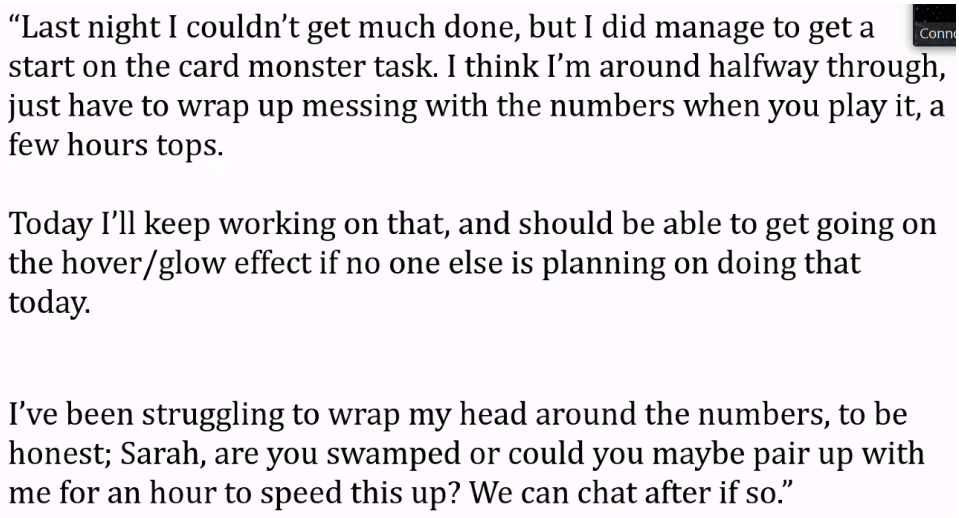
The goal is to speedrun necessary communication and to plan chats later in the day. Not to resolve issues. Not to give management a status update. Meant for the dev team.

The 3 questions asked:

1. What have I worked on since the last-stand-up?
2. What do I plan on working on until the next stand-up?
3. Is anything blocking my work?

- You may want to refer to the scrum board, or have a very very brief demo to show.

Example discussion:



Asynchronous daily stand-up process:

1. Pick a time when everyone will have their stand-up written, answering the three questions
2. Post to a stand-up channel by the specified time
3. At the specified time, go in to follow-up on anything that needs follow-ups

End-of-Day (daily) - NOT a meeting, just some things you should do at the end of each day.

Update the board.

- Update task time estimates to reflect the work you've done.
    - You've worked on a 6-hour task for 3 hours and you estimate that you will need 3 more hours to finish it. You should change the time needed for that task from 6 hours to 3 hours.
- Move tasks to the correct column. If you have finished a task completely, move it to the "done" column.

4. **Sprint Review (end)**

You will schedule a playtest with your team and your product owners/stakeholders, and see the stories that were completed.

At this time, adjust the backlog.

- Any new features?
- Features to cut?
- Features to change?
- Effort to reassess?

Timebox the sprint review (limit the time); keep it short, about an hour for a 1-week sprint.

5. **Sprint Retrospective Meeting (end)**

Check in on the team. Usually just the devs, no product owners.

1. What should we start doing next sprint?
2. What should we stop doing next sprint?
3. What went really well, that we should continue doing next sprint?

You should have a list of things to do next sprint to try and improve next time. At each retro, you try to see if the retrospective changes from last sprint helped or hindered.

Overview of process:

First, we build a backlog. A list of stories, the whole game.

We then start our sprint with Sprint Planning.

We pick some stories to finish, turn them into tasks.

Every day, we plan our day together, and work on tasks.

At the end of the sprint, we review and playtest the game.

We check if we need to adjust the backlog based on the playtest.

We discuss ways to do better as a team next sprint.

Notes and Tips

- Always be honest. We need transparency. Never overcommit to look better.
- Work consistently. We can't leave everything to the end. Even a small task daily is better than a bunch at the end. (Perhaps, you can have a "zero-day" task list with very simple tasks for when you're having a bad day)
- Consider time remaining.
- Your estimates are wrong. The goal isn't perfect accuracy, it's discussion.

Final Project Notes

During the project, labs will be used for 1-week Sprint Planning and Retrospective. They will be brief

The first weeks are to paper prototype and build a backlog.

Daily stand-ups are expected, you can schedule and coordinate yourself.

Communication Tips; Quick Tips about Teams in General
(Game Design Workshop, p. 411; Art of Game Design, ch. 26)

- **Know the goal.**
  - What's the agenda of the meeting?
  - What are you expected to contribute to this meeting?
  - What signified the end of the meeting?
- **Be Prepared.**
  - Do any research you need to do before the meeting.
  - Can you prepare to make the meeting run smoother?
- **Objectivity.**
  - Separate ideas from individuals and emotions.
    - "The spaceship idea" vs "My spaceship idea".
  - Pose questions, rather than voicing opinions.
    - "What if we changed X? rather than "We should change X".
- **Clarity.** Be precise with expectations. Assume they'll misinterpret.
  - "I'll get the UI done by Thursday" vs "I'll email you a 3-5 page description of the combat UI by 5:00 PM Thursday"
- **Comfort and tolerance.** People become aggressive and intolerant when they're uncomfortable. Work towards noticing it.
  - Is the meeting too long?
  - Are you too tired?
  - Is it too hot in your room?
- **Respect.**
  - Always consider: "What if I'm wrong?"
  - Be willing to disagree, yet commit anyway.
- **Love.**
  - Find something to love.
    - Love the game you're working on.
    - Love that part of the game you're working on.
    - Love the audience, and provide the best experience.
  - Find your passion, and make it your goal to deliver on it.

A good guide to Scrum that I found:
https://medium.com/ideas-by-crema/the-4-scrum-ceremonies-made-simple-a-quick-guide-to-scrum-meetings-ea91fe604d88
Guide to building a Scrum board: https://clickup.com/blog/scrum-board/

Lecture 13
Feb 28 2023

We reviewed the **trailers** for some games that were created in this class last year.

- WIZRAGE
- NFThief
- Nutrition Attrition!
- The Chronicles of Jerry Pugsley the Conqueror
- The Darkest Depths
- Garden Guardians

We spent most of class going over the **final project requirements**, which are posted on Brightspace. You can look for a group during this week's tutorial, if you haven't found one yet. Note that if you are not enrolled in a group on Brightspace by March 5th, you will be randomly assigned to one.

Then, we did a quick **brainstorming activity**. Think of:

- 3 strange problems to overcome
- 3 evocative aesthetic themes
- 3 interesting settings

If you came up with three very quickly, throw them away. It should stump you a bit, try again!

Now, randomize the order (you can use random.org) and try to make the game make sense.

A game about overcoming obstacle, set in setting, which evokes a feeling of aesthetic.

What mechanics support this? Consider the elemental tetrad. Repeat this activity a few times.

We finished off by discussing how to find the **best group** for you.
When looking for a group, consider the following:

1. Do you want to make a game with lots of good content, or a game with little content but mechanically varied?
2. What skills do you bring in and want to emphasize?
3. What genres, dynamics, themes, or mechanics inspire you?
4. What time commitment are you planning on?

Lecture 14
Mar 2 2023

Notes: Godot 4.0 has been released :)
Battlesnake tournament is going on this weekend!

18 - AI in Games

Recall: Randomness Dynamics

Games can be more or less random, from a scale of Deterministic ←————→ Random
For the final project, steer away from purely random games, they make the player feel like their choices are not meaningful.

Randomness: having things be randomly generated
Variation: having a wide range of outcomes
Sometimes, randomness leads to too little variation → cereal example

Pre-luck: input: random, output: player; randomly generating a game state and players take an action and know the outcome
Post-luck: input: player, output: random; player makes a move, game randomly generates result

Humans are terrible at perceiving probability → Monty hall problem
Aside: Raph Koster describes our inability to understand probability as a game genre.

We often lie to players about probabilities, to skew things in the player's favour.
Ex: 90% chance to hit is actually a 98% chance to hit

Artificial intelligence is a technique that controls some process with little-to-know human interaction.

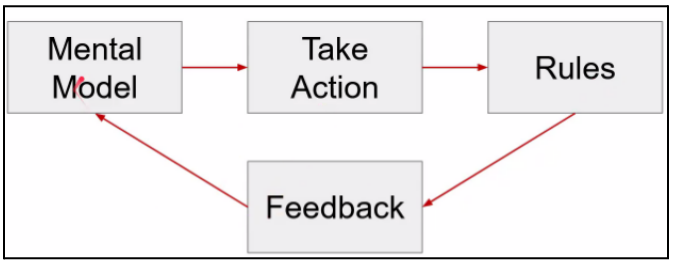
- **Agent AI**
  - Simulate competitive opponents for the player
  - Simulate non-competitive opponents for the player
  - Ally to the player
  - Finding shortest paths between two points
- **Procedural Generation**
  - Generate maps from random noise
  - Generate quests and stories
  - Generate character models, textures
- **Procedural Curation**
  - Reduce enemy aggression when player is doing poor
  - Decrease health drops to manage tension
  - Select the right item drops to keep the player engaged
  - Automatically balancing pricing for game balance

So… what is AI?
Usually a bunch of code and statistics.

Agent AI (How characters behave)
Agent: some entity that has a goal

One model for AI agents is **PEAS**:
Performance: How good am I doing?
Environment: Where am I?
Actuators: What can I do? How can I affect the environment?
Sensors: What can I perceive?

AI has a "mental model" of the environment.



Note: When we're implementing AI in games, we're not really implementing AI like in the rest of computer science.

The real key in game AI is: **How do we decide on an action?**

Ex: Individual ants are simple and not very smart. But ant behaviours are complex.

- If an ant has no food:
  - If it doesn't smell pheromones:
    - Keep looking
  - If it detects pheromones:
    - Follow that smell
- Else if it has food:
  - Turn around and return to colony
  - While returning, drop pheromones

Practical AI has to be simple and manageable
Few simple rules (mechanics) → complex behaviour (dynamics)
As with games, mechanics lead to resulting dynamics.

Ex: Pac-Man

- If Pac-Man doesn't have a power pellet:
    - Ghosts will either be in "chase" mode or "scatter" mode, based on a timer
        - If scattering:
            - Move to a corner of map
        - If chasing:
            - Pursue the player experience

Methods of building AI behaviours

FSM: Finite-State Machines (we will deep dive into this one)

- Set of values (state), how to transition from one to the next, and how we "update" in a given state
- Used largely as a component of many AI systems

BT: Behaviour Trees, Decision Trees

- A hierarchical graph of tasks to complete
- First (widely) used in Halo 2 enemy AI

GOAP: Goal-Oriented Action Planning

- States we want to get to, and pathfind tasks to get there
- First used in F.E.A.R., revolutionary AI
- Now, HTN or Hierarchical Task Networks have mostly replaced it

Utility AI

- Have a set of actions you could take, look at the game state, score each action based on state
- Used extensively in many Ubisoft games (like WatchDog AI)
- Can be entirely controlled using a spreadsheet

Resource: http://www.gameaipro.com/

Questions to ask:

- How competitive should the game be?
- Are we out to make the player feel awesome?
- Are we trying to simulate human components?
- How predictable should the AI be?
- How much will the player need to memorize?

For **AI Actors**, we have a few **key goals**:

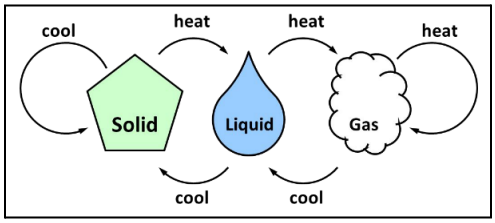
1. Can the player intuit/learn the behaviours?
2. Do the behaviours give the player choice, curiosity?
3. Does it really improve the game experience?

Learnable AI: player should be able to plan actions based on predictability of AI.
Intuitable AI: player learns the reactions of the AI to set up interesting scenarios.
Ex: if you know an AI will dive to avoid a bomb, you can throw a bomb to get them out of hiding

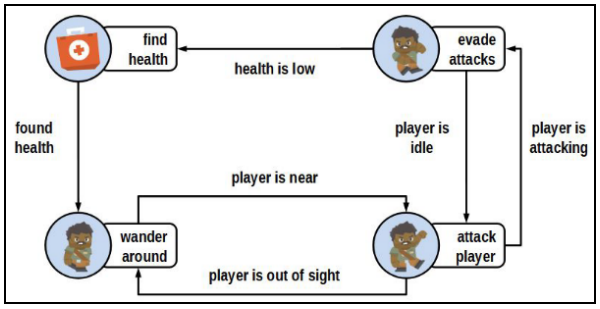
Video: https://www.youtube.com/watch?v=9bbhJi0NBkk

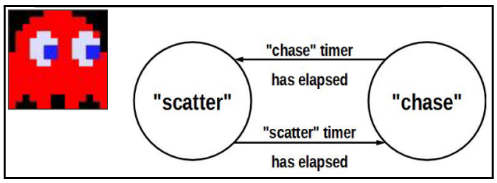
- Cheating (AI is part of game loop)
  - Always miss the player on the first few shots
  - Don't turn around when Batman's stalking you
- Barks (lets us know what the AI is thinking)
  - AI reacts to player actions, "Did I leave that door open?"
  - AI says "Musta been the wind."
  - Showing vision cones
  - Showing walking lines
- Tracking on a Blackboard
  - A global dictionary of state the AI can look at, instead of really "seeing" it

Finite-State Machines



A FSM is always "in" a specific state, from a limited set of predefined states.
Given some event, it transitions to another state.





How do we implement this in code?
You can just swap your update function from the game loop with different update functions based on state.
Typically, an FSM update() function looks like:

1. Look at some game variables
2. Update my game data
3. Select my next state (incl. myself)

Resource: https://www.gdquest.com/tutorial/godot/design-patterns/finite-state-machine/

Finite state machines are the basis for many AI techniques, including GOAP that we mentioned before.

Ex: RocketsRocketsRockets has such a simple AI, which players thought was really great. They researched the “best” AI to use for a while, and ended up with the following:

1. If wall on left, turn right
2. If wall on right, turn left
3. If player in front, shoot

Fun AI tends to be predictable, in order to set up interesting situations for the player to engage in and formulate plans.

So, start simple with AI.

Resource: https://www.youtube.com/watch?v=WbHMxo11HcU
Director AI: maintains flow state by changing variables of each level

- Weak: hide enemies, strong: add enemies
- Changing drop rates
- Resident Evil also uses this to maintain a sense of dread and limitation
- Ex: bullets despawning from level when you have enough in Half Life Alyx

Lecture 15
Mar 7 2023

Final project resource: https://letsmakeagame.net/game-idea-generator/

19 - Numerical Relationships

This will be on the quiz, so make sure to go through practice problems. Quiz 2 will be multiple choice.
Games rely heavily on math to run well.

Balance

Remember: we need to maintain a flow state.

Ex: Cookie Clicker (concerned about **progression**)

- How much should buildings cost to release them with an interesting pattern?
- How long should a session be?

Ex: Dicey Dungeons (concerned about **expected damage**)

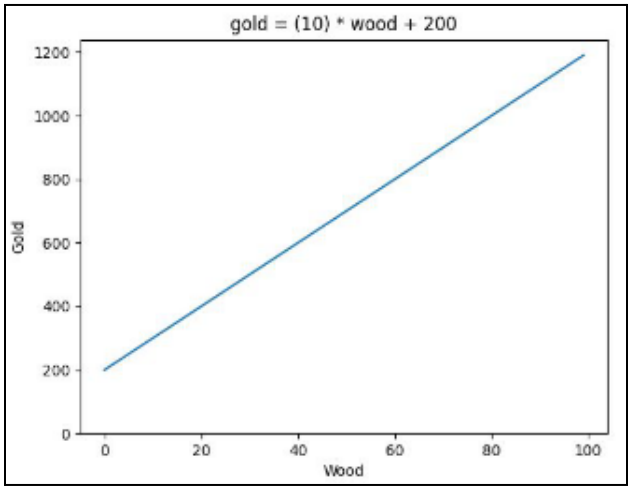
- Given 3 dice, how much damage do we expect to deal on a turn?

Numerical Relationships

We define the relationship between two values as an **algebraic expression**.

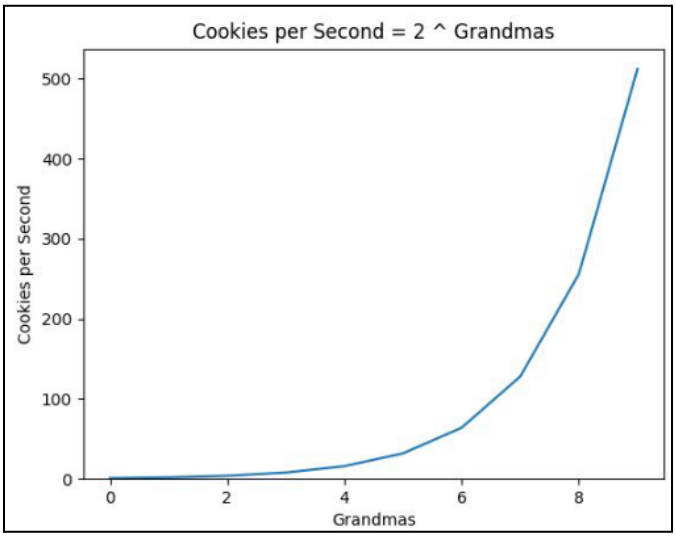
Linear relationship: increasing one variable increases the other variable. May have a base cost. In the form $y = mx + b$.

- Using linear relationships is not exciting.
    - Recall our economy discussions: we typically want to avoid obviously exchangeable resources.
    - Recall our pacing discussions: we're more interested by fluctuation than consistency.



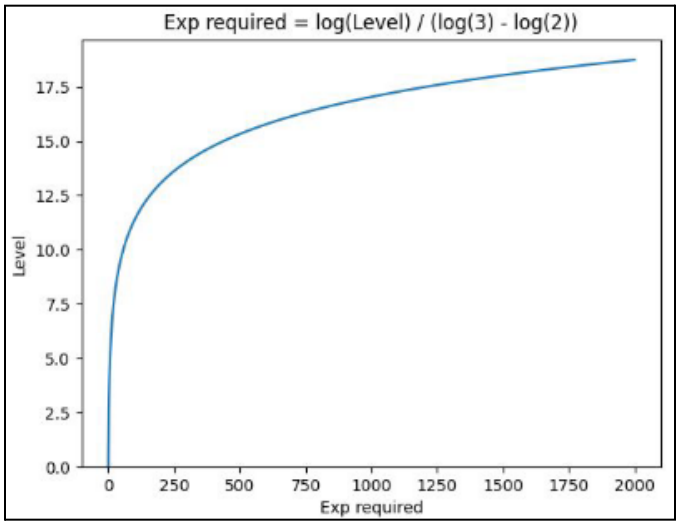
Exponential relationship: rapid growth from the beginning, which does not slow down. In the form $y = b^x$.

- This feels exhilarating, and can create a fiero-rich positive feedback loop.



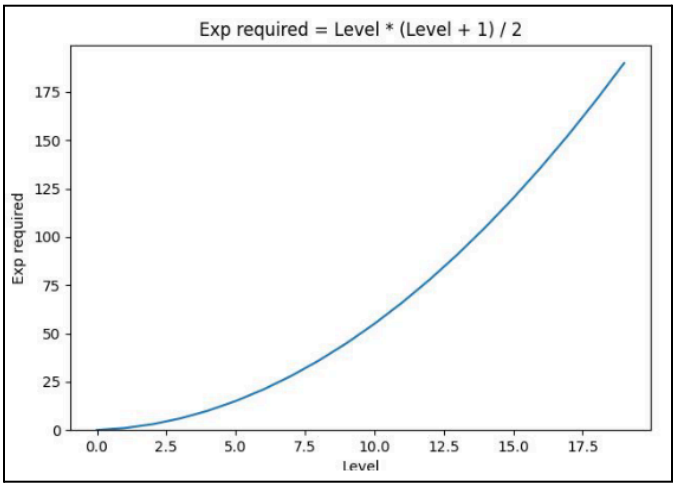
The logarithmic curve: rapid growth at the beginning, then slows down dramatically. In the form $y = log(x)$.

- Games that have “diminishing returns” use log curves.



Triangular Relationship: rapid growth but more controlled than exponential relationships. In the form $y = \frac{x(x+1)}{2}$. The pattern is: $1 + 2 + 3 + 4 + 5 +$…

- Easy for players to understand progression; it just increases by 1 each time.
- Help us to incentivize or dissuade actions by increasing or decreasing cost over time.
    - Such as preventing players from buying items in bulk.



We can **combine** relationships.
Ex: Want a rapid tutorial, some exciting growth, and some long end game?

Level 1-10 = Exponential leveling (log exp requirements)
Level 10-30 = triangular leveling
Level 30-40 = Log leveling

Understanding these numerical relationships is understanding our intended dynamics.

- How do we want the player to feel?
- What curve approximates that?
- What variables can I work with to get there?

Practice
Try going to game wikis and investigating the numerical progression for resources, levelling, building. Try to work out what relationships they follow, and how they contribute to the dynamic.

20 - Probability for Games

Probability: a measurement of how likely it is for some event to occur.

- Described as a number between 0 and 1.
  - 0: impossible
  - 1: certain

The probability of event A is written as $P(A)$.

$$P(A) = \frac{number\ of\ events\ where\ A\ happens}{total\ number\ of\ events}$$

Eg. probability that a normal six-sided die lands on an odd number is:

(Name of the event happening)

$$P(\text{odd}) = \frac{|\{1, 3, 5\}|}{|\{1,2,3,4,5,6\}|}$$

|Cardinality or # of Elements or Size|

{Unordered set of events}

$= \frac{3}{6} = 0.5 = 50\%$

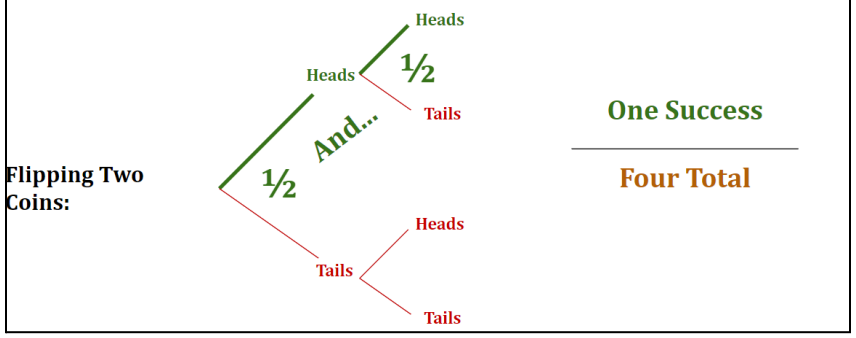
Two events A and B are **independent** if whether event A occurs does not affect whether event B occurs. **Dependent** events are the opposite of independent events.

If events A and B are independent, the probability of both of these happening is $P(A \cap B) = P(A) \times P(B)$.

Example: Probability of both coins landing on "Heads" is:

$$P(A \text{ and } B) = P(A \cap B) = P(A)P(B) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

We can consider a simple probability tree to test this:

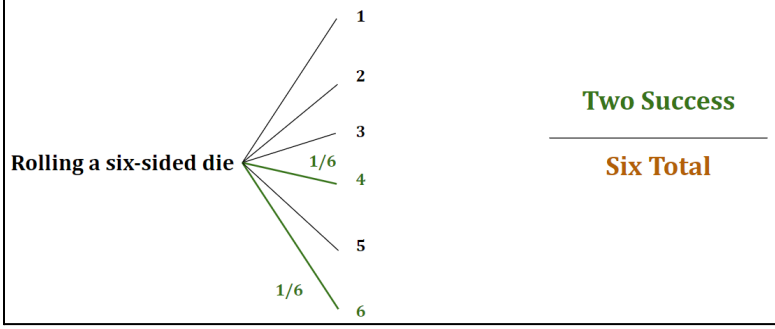


Events A and B are **mutually exclusive** if both cannot occur together.
The chance of either event occurring is: $P(A \cup B) = P(A) + P(B)$

Eg. Rolling either a 4 or 5 on a six-sided die.

$$P(A \cup B) = P(A) + P(B) = \frac{1}{6} + \frac{1}{6} = 2/6 = \frac{1}{3}$$

We can consider a simple probability tree to test this:



Combat and Probability
Some questions you may encounter...

1. If a Large Greatsword does 3-18 Points of Damage, What is the Average Amount of Damage it can do?
    - To consider: are we skewing RNG toward 18, or is each option equally likely?
2. Is a Shield that Absorbs 6 Points of Damage Better than a Shield that Absorbs 3-12 Points of Damage?
    - To consider: is it (3,4,5,6) and a 50% chance of doubling?

A typical set of a combat mechanics uses several random components:

Attacker has Likelihood of **Successfully Attacking**
Defender has Likelihood of **Successfully Evading**
Attacker's Weapon Inflicts **Random Damage Value**
Defender's Armor Absorbs **Random Damage Value**

Note that not all of these have to be used in a game.

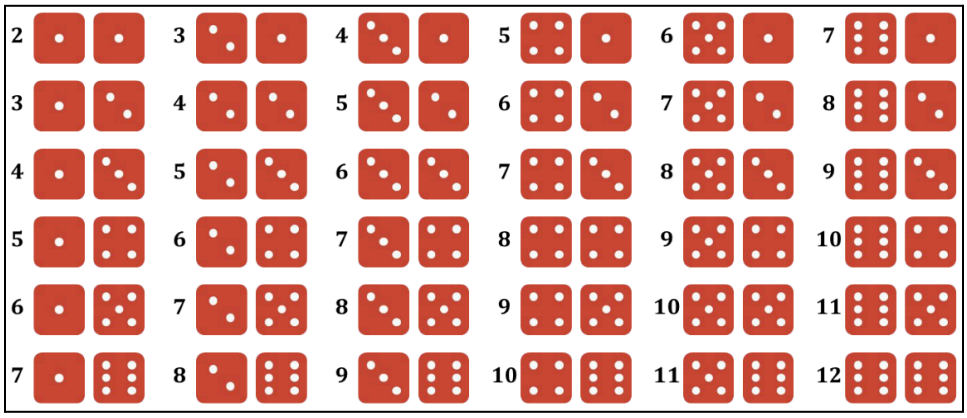
Ex: Let's say evasion is the chance of successfully avoiding an already successful hit. Assume a uniform chance of success given hit% and evade%. What is the likelihood the defender takes damage? What we're asking is: What is the probability that BOTH the attacker successfully hit AND we did not evade?
→ If the chance that you don't take damage on a successful hit = evade%, the chance that you do take damage is 1-evade%.
→ The chance that both the attacker hits AND the defender fails to evade = hit%×(1-evade%).

We can also simply calculate damage done as… weapon_roll - armor_roll
(...but usually armor_roll is constant)

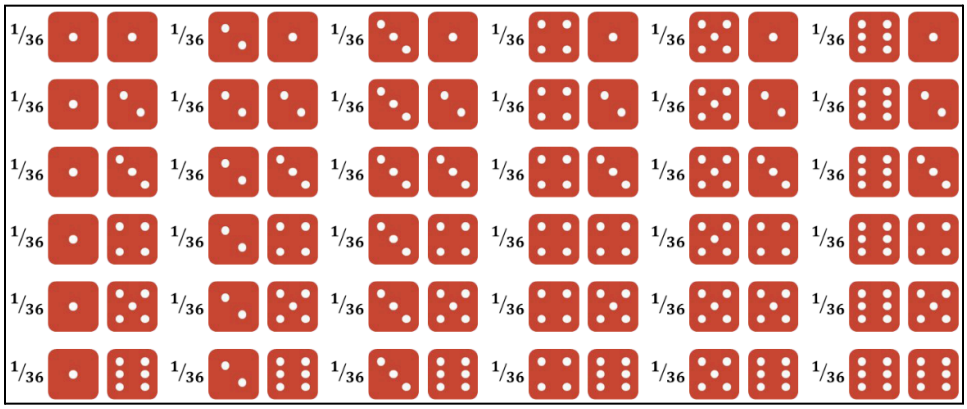
We often want to know the **expected value** of probability. That is, if we give a random value to a variable many, many, many times what's the average result?

Suppose a Large Spear does 2d6 damage (i.e. the result of rolling two six-sided dice)
What's the expected value for the damage? 7. Why? ↓

We know the events are independent.
What are the possible outcomes of rolling 2d6?



What is the probability of each outcome?



Expected value is found using a **weighted average**.

Expected Value for a Discrete Random Variable x is

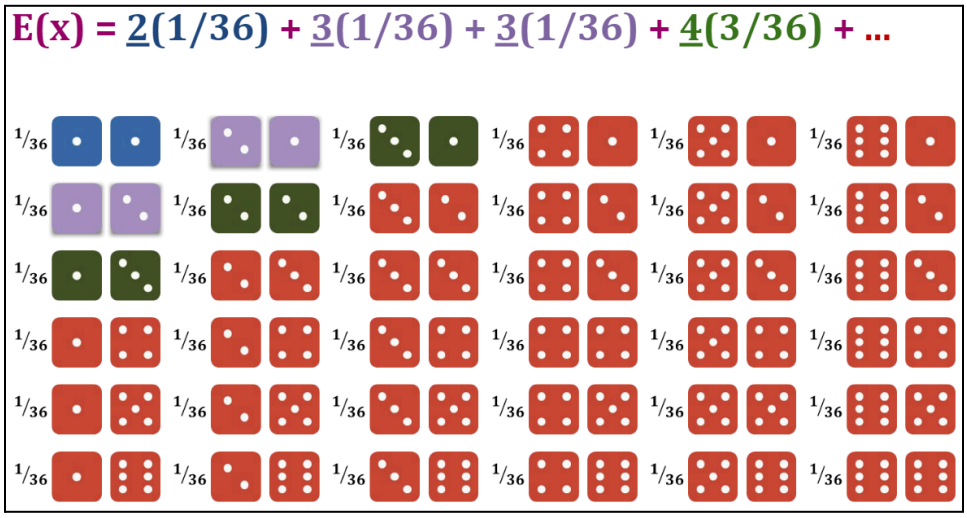
$$E(x) = \sum x_i \, p_i$$

where:

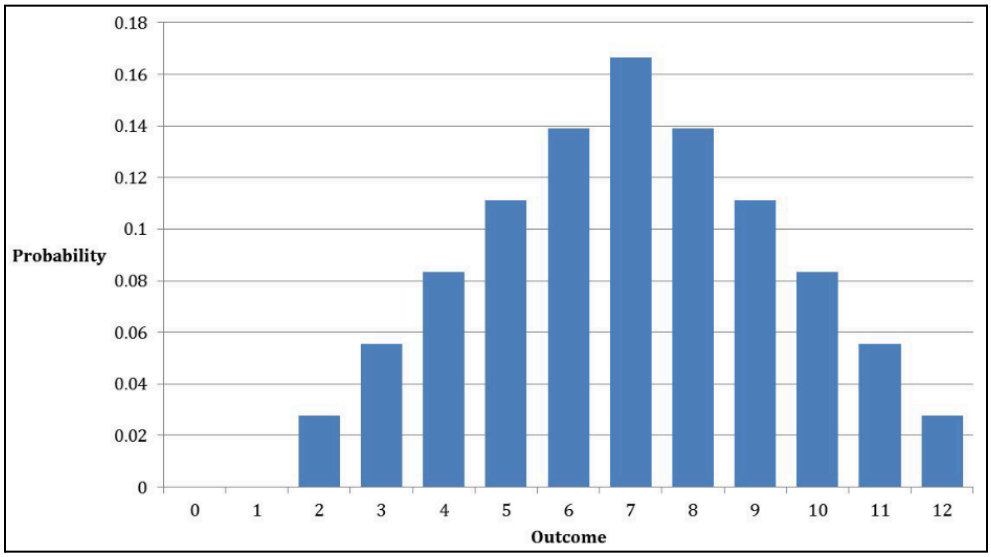
x_i **is the Value Attached to the** i^{th} **Outcome**

p_i **is the Probability of the** i^{th} **Outcome**

Calculate the expected average:



We can represent this data in a graph. You can see that 7 is the most likely to occur.



This visualization also helps us build intuition about our dynamics!
What if we know that we need an 8+ to win? How does that change how you view the game dynamics?

Great dice probability visualization tool: https://anydice.com/

Ex:

Suppose 1d20 is used for the attacker's attack roll and that the defender has no evasion roll and has sufficient armor that a roll of 16 or higher is necessary for the attacker to hit and suppose also that the attacker's weapon uses 1d6 for the damage roll except that, when the attacker's attack roll results in a 20, the damage inflicted on the defender will be doubled (i.e., 1d6 * 2).

What is the Expected Damage?

Methods of calculating expected value:

1. Spreadsheet
2. Monte Carlo Simulation
3. Calculating!

1. Spreadsheet

H2 fx =sum(G2:G)

	A	B	C	D	E	F	G	H	I
1	1d20	1d6	Hit?	Crit?	Damage?	Probability?	Product E, F	Expected Value	
2	1	1	FALSE	FALSE	0	0.008333333333	0	1.05	
3	2	1	FALSE	FALSE	0	0.008333333333	0		
4	3	1	FALSE	FALSE	0	0.008333333333	0		
5	4	1	FALSE	FALSE	0	0.008333333333	0		
6	5	1	FALSE	FALSE	0	0.008333333333	0		
7	6	1	FALSE	FALSE	0	0.008333333333	0		
8	7	1	FALSE	FALSE	0	0.008333333333	0		
9	8	1	FALSE	FALSE	0	0.008333333333	0		
10	9	1	FALSE	FALSE	0	0.008333333333	0		
11	10	1	FALSE	FALSE	0	0.008333333333	0		
12	11	1	FALSE	FALSE	0	0.008333333333	0		
13	12	1	FALSE	FALSE	0	0.008333333333	0		
14	13	1	FALSE	FALSE	0	0.008333333333	0		
15	14	1	FALSE	FALSE	0	0.008333333333	0		
16	15	1	FALSE	FALSE	0	0.008333333333	0		
17	16	1	TRUE	FALSE	1	0.008333333333	0.008333333333		
18	17	1	TRUE	FALSE	1	0.008333333333	0.008333333333		
19	18	1	TRUE	FALSE	1	0.008333333333	0.008333333333		
20	19	1	TRUE	FALSE	1	0.008333333333	0.008333333333		
21	20	1	TRUE	TRUE	2	0.008333333333	0.01666666667		
22	1	2	FALSE	FALSE	0	0.008333333333	0		
23	2	2	FALSE	FALSE	0	0.008333333333	0		
24	3	2	FALSE	FALSE	0	0.008333333333	0		
25	4	2	FALSE	FALSE	0	0.008333333333	0		
26	5	2	FALSE	FALSE	0	0.008333333333	0		
27	6	2	FALSE	FALSE	0	0.008333333333	0		
28	7	2	FALSE	FALSE	0	0.008333333333	0		

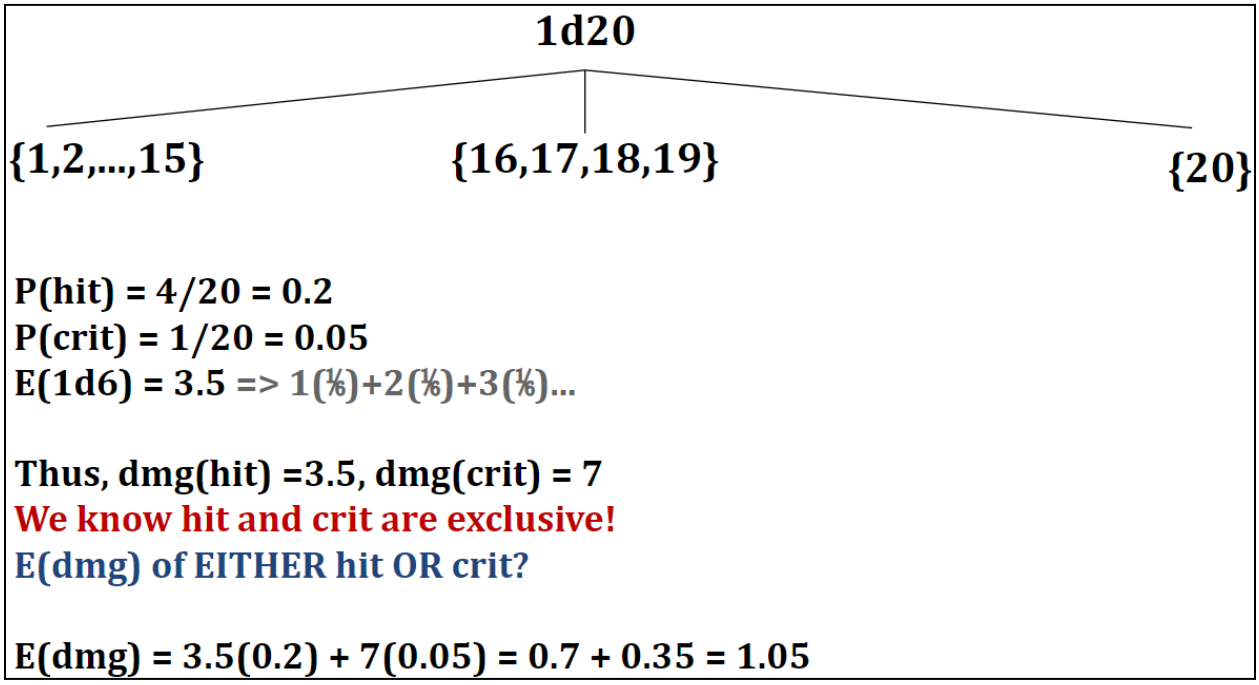
2. **Monte Carlo Simulation (Code)**

```python
def main():
    damage_so_far = 0

    # Run a ton of trials!
    for trial_index in range(NUM_TRIALS):
        damage_so_far += simulate_attack()

        # Print out average damage over time to see how it evolves with more trials
        # We add one to trial index, because at this point, we've completed the trial
        if trial_index % 50000 == 0:
            print(f'After {trial_index+1:6d} trials, the average damage so far is: {damage_so_f

if __name__ == "__main__":
    main()
```

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

After 1050001 trials, the average damage so far is: 1.0470180504589996
After 1100001 trials, the average damage so far is: 1.047220866162849
After 1150001 trials, the average damage so far is: 1.0473860457512645
After 1200001 trials, the average damage so far is: 1.0475216270653107
After 1250001 trials, the average damage so far is: 1.0475807619353905
After 1300001 trials, the average damage so far is: 1.048129193746774
After 1350001 trials, the average damage so far is: 1.0477458905585997
After 1400001 trials, the average damage so far is: 1.0480221085556367
```

3. **Calculating**

don't hit↓ hit↓ crit↓



= dmg(hit)×P(hit) + dmg(crit)×P(crit)

These 3 methods all give us the same expected value of 1.05.

Lecture 16
Mar 9 2023

Lecture 15 lecture addendum

1. Probability is always between 0 and 1.
    a. Use this to check your work.
    b. If you went over 1, perhaps events that you thought were mutually exclusive were not.
2. $\frac{1}{2} = 0.5 = 50\%$
    a. Fractions give us information about the probability space.
    b. We use decimals in our code.
    c. We use percentages when communicating verbally.
3. When in doubt, simplify the problem using a visual aid.
    a. Building a probability tree like shown in lecture 15 can highlight patterns to simplify the calculation or provide some intuition.
4. "Either" and "or" usually means add, and "both" and "and" usually mean multiply.
    a. Mutually exclusive ≠ independent events.

21-Combinatorics for Games

Combinatorics is the branch of math where we count the number of different, valid combinations of things. We use combinatorics to find the **quotient** of the probability of an event, which is the number of possible outcomes. This is useful for calculating probability and game balance.

Ex: In Diablo 2, the Cryptic Sword is customizable with up to 4 sockets that could each be equipped with a gem. There are 7 types of gems and each gem is 1 of the 5 gem grades.

7 types of gems: Amethysts, Diamonds, Emeralds, Rubies, Sapphires, Topazes, Skulls
5 gem grades: Chipped, Flawed, Normal, Flawless, Perfect

Q: How Many Different Cryptic Swords Are Possible?
→ There are 35 different possible gems to equip since $7\ types \times 5\ grades = 35$.
→ You can also leave a socket empty.
→ There are 36 options for what to equip in each of the 4 sockets (35 gems, 1 empty).
→ There are $36 \times 36 \times 36 \times 36 = 1,679,616$ combinations.

When designing our games, we need to watch out for **information overload**. Consider player dynamics; maybe they will only ever insert the perfect grade gems, or they will always use the right gems for the occasion. We can factor these into our combinatorics calculations to balance the game.

The successful application of combinatorics is an **amplifier** to the design work. These additional choices provide opportunities for fun (notably expression, challenge, submission, and narrative). For example, some players may want to do a gemless run.

When choosing options (e.g. Gems, Skills), it is important to answer the following questions:

1. Can an Option be Repeated?
    a. Ex: If the first socket is Ruby, can the 2nd socket be Ruby too?
2. Does the Order Matter?
    a. Ex: Does Ruby + Skull have the same effect as Skull + Ruby?

    If YES: Use **permutations**
    If NO: Use **combinations**

Note that both permutations and combinations can be computed with and without repetition.

The **factorial** of a positive integer n is the product of that integer and all the positive integers smaller than it.
$n! = n \times (n-1) \times \ldots \times 1$
Ex: $4! = 4 \times 3 \times 2 \times 1 = 24$

Permutations (Order matters)

When r things are being chosen from a collection of n things, the number of different permutations is: ${}_nP_r = \frac{n!}{(n-r)!}$.

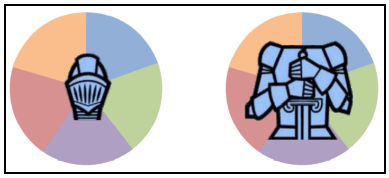
Combinations (Order doesn't matter)

When r things are being chosen from a collection of n things, the number of different combinations is: $\binom{n}{r} = {}_nC_r = \frac{n!}{r!\,(n-r)!}$.

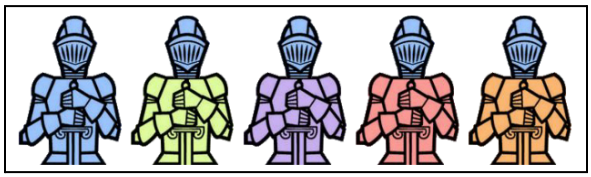
No repetition

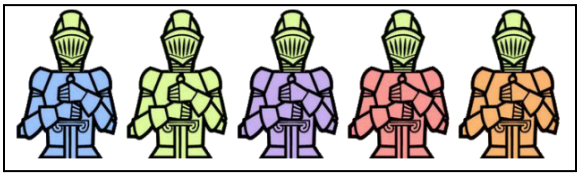
When repetition is not permitted, the number of choices is reduced after each choice.
e.g., If you Draw an Ace from a Deck of Cards, you Reduce the Number of Aces Remaining in the Deck

Ex 1: Options can be repeated and order matters. There are 5 different colours (blue, green, purple, red, orange) that can be used to customize a player's 2 armour slots (helmet, body). How many permutations are there?



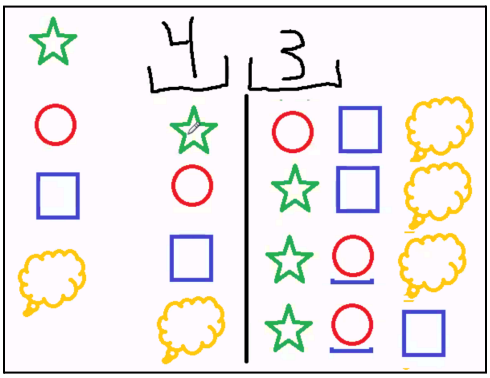
Pair blue helmet with each body color… (5 options)
Pair green helmet with each body color… (5 options)
...etc.
→ 5 × 5 = 25 options





		Helmet colour (first)				
		Blue	Green	Purple	Red	Orange
Body colour (second)	Blue	B, B	G, B	P, B	R, B	O, B
	Green	B, G	G, G	P, G	R, G	O, G
	Purple	B, P	G, P	P, P	R, P	O, P
	Red	B, R	G, R	P, R	R, R	O, R
	Orange	B, O	G, O	P, O	R, O	OO

Ex 2 (a): There are 4 shapes (star, circle, square, bubble). How many combinations of 2 shapes can be made if order matters?



$$_nP_r = \frac{n!}{(n-r)!}$$

$$_4P_2 = \frac{4!}{(4-2)!}$$

$$= \frac{4\times3\times2\times1}{2\times1}$$

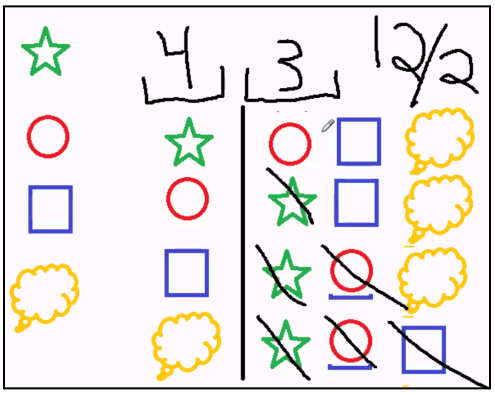
$$= 4 \times 3$$

$$= 12\ combinations$$

		First shape			
		Star	Circle	Square	Bubble
Second shape	Star	No repetition	circle, star	square, star	bubble, star
	Circle	star, circle	No repetition	square, circle	bubble, circle
	Square	star, square	circle, square	No repetition	bubble, square
	Bubble	star, bubble	circle, bubble	square, bubble	No repetition

Ex. 2 (b): There are 4 shapes (star, circle, square, bubble). How many combinations of 2 shapes can be made if order does not matter?

→ Need to eliminate duplicates.



$$_nC_r = \frac{n!}{r!\,(n-r)!}$$

$$_4C_2 = \frac{4!}{2!\,(4-2)!}$$

$$= \frac{4\times3\times2\times1}{(2\times1)(2\times1)}$$

$$= \frac{24}{4}$$

$$= 6\ combinations$$

		First shape			
		Star	Circle	Square	Bubble
Second shape	Star	No repetition	Duplicate	Duplicate	Duplicate
	Circle	star & circle	No repetition	Duplicate	Duplicate
	Square	star & square	circle & square	No repetition	Duplicate
	Bubble	star & bubble	circle & bubble	square & bubble	No repetition

When options can be repeated but the order does not matter, you must be careful not to overcount by counting duplicates.

Ex: In Titan Quest (2006) you have the option of selecting 2 of the 8 masteries (storm, earth, warfare, spirit, defense, nature, hunting, rogue).

Order does not matter.
You cannot repeat masteries, but you can leave the second mastery empty.
→ combination

(a) How many ways are there to pick 1 mastery?

First mastery							
Storm	Earth	Warfare	Spirit	Defense	Nature	Hunting	Rogue

$${}_8C_1 = \frac{8!}{1!\,(8-1)!} = \frac{8!}{7!} = 8\ ways$$

(b) How many ways are there to pick exactly two masteries (second mastery not empty)?

		First mastery							
		Storm	Earth	Warfare	Spirit	Defense	Nature	Hunting	Rogue
Second mastery	Storm	No repetition	Duplicate	Duplicate	Duplicate	Duplicate	Duplicate	Duplicate	Duplicate
	Earth	Storm & Earth	No repetition	Duplicate	Duplicate	Duplicate	Duplicate	Duplicate	Duplicate
	Warfare	Storm & Warfare	Earth & Warfare	No repetition	Duplicate	Duplicate	Duplicate	Duplicate	Duplicate
	Spirit	Storm & Spirit	Earth & Spirit	Warfare & Spirit	No repetition	Duplicate	Duplicate	Duplicate	Duplicate
	Defense	Storm & Defense	Earth & Defense	Warfare & Defense	Spirit & Defense	No repetition	Duplicate	Duplicate	Duplicate
	Nature	Storm & Nature	Earth & Nature	Warfare & Nature	Spirit & Nature	Defense & Nature	No repetition	Duplicate	Duplicate
	Hunting	Storm & Hunting	Earth & Hunting	Warfare & Hunting	Spirit & Hunting	Defense & Hunting	Nature & Hunting	No repetition	Duplicate
	Rogue	Storm & Rogue	Earth & Rogue	Warfare & Rogue	Spirit & Rogue	Defense & Rogue	Nature & Rogue	Hunting & Rogue	No repetition

$${}_8C_2 = \frac{8!}{2!\,(8-2)!} = \frac{8!}{2!\times 6!} = \frac{8\times7\times6\times5\times4\times3\times2\times1}{(2\times1)(6\times5\times4\times3\times2\times1)} = \frac{8\times7}{2\times1} = \frac{56}{2} = 28\ ways$$

(c) If each player had to select exactly three masteries, how many different options exist?
For the 1st mastery, there are 8 choices,
For the 2nd mastery, there are only 7 choices (since you can't choose your first mastery again),
For the 3rd mastery, there are only 6 choices,
etc.
Also, there cannot be any repeats which drastically reduces the number of combinations.

$$_8C_3 = \frac{8!}{3!\,(8-3)!} = \frac{8!}{3!\times 5!} = \frac{8\times7\times6\times5\times4\times3\times2\times1}{(3\times2\times1)(5\times4\times3\times2\times1)} = \frac{8\times7\times6}{3\times2\times1} = \frac{8\times7}{1} = 56\ ways$$

"Every shuffled deck of cards is unique"

Ex: Suppose that we define shuffling a deck of 52 cards as choosing a card for the 1st position, then choosing a card for the 2nd position, then choosing a card for the 3rd position, etc…
How many different ways are there to shuffle a deck of 52 cards?
→ no repeats
→ order matters
→ permutation

$$_{52}P_{52} = \frac{52!}{(52-52)!} = \frac{52!}{1!} = 52! = 80,658,175,170,943,878,571,660,636,856,403,766,975,289,505,440,883,277,824,000,000,000,000$$

Ex: How many different ways are there to choose 2 cards from a deck of 52 cards?
→ no repeats
→ order matters
→ permutation

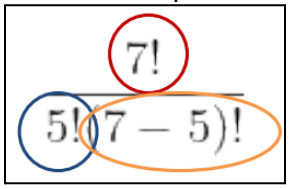
$$_{52}P_2 = \frac{52!}{(52-2)!} = \frac{52!}{50!} = 52 \times 51 = 2652\ ways$$

Ex: A full party in Planescape: Torment (1999) consists of 6 of the 8 different characters in the game, one of which must be the Nameless One. There are 7 other characters, so how many different parties of size 6 are possible?
→ no repeats
→ order does not matter
→ combination

The Nameless One is always in the party, so we need to calculate how many combinations of 5 people are possible out of the 7 characters remaining.

$$_7C_5 = \frac{7!}{5!\,(7-5)!} = \frac{7!}{5!\times 2!} = \frac{7\times6\times5\times4\times3\times2\times1}{(5\times4\times3\times2\times1)(2\times1)} = \frac{7\times6}{2\times1} = \frac{42}{2} = 21\ ways$$

A more in-depth look at the formula:



7!: total number of unique sequences of all characters
5!: reduce by how many ways the characters could be in order (since order does not matter)
$(7-5)!$: removing all of the sequences that we aren't choosing

Summary of permutations and combinations

Permutations (order matters)		Combinations (order does not matter)	
Without repetition	With repetition	Without repetition	With repetition
Multiply n r times, reducing n by 1 each time. $nPr = \frac{n!}{(n-r)!}$	Multiply n r times. n^r	Get the number of combinations with repetitions and divide out the repeating elements. $nCr = \frac{n!}{r!\,(n-r)!}$	Get the number of sequences and divide out the ones you don't need. $\frac{(r+n-1)!}{r!(n-1)!}$
eg. number of ways to draw 3 cards from a 52-card deck $52C3 = \frac{52!}{(52-3)!}$ $= \frac{52!}{49!}$ $= 52 \times 51 \times 50$ $= 132600\ ways$	eg. number of ways to customize 4 armour slots with 16 colours $16 \times 16 \times 16 \times 16$ $= 16^4$ $= 65536\ ways$	eg. number of ways to pick 3 out of 5 balls out of a bag with no replacement. $5C3 = \frac{5!}{3!\,(5-3)!}$ $= \frac{5!}{3!\times 2!}$ $= \frac{5\times4\times3\times2\times1}{(3\times2\times1)(2\times1)} = \frac{5\times4}{2\times1}$ $= \frac{20}{2} = 10\ ways$	eg. number of ways to pick out 3 out of 5 balls out of a bag, putting the balls back after each draw. $\frac{(3+5-1)!}{3!\,(5-1)!} = \frac{7!}{3!\times4!}$ $= \frac{7\times6\times5\times4\times3\times2\times1}{(3\times2\times1)(4\times3\times2\times1)}$ $= \frac{7\times6\times5}{3\times2\times1} = 35\ ways$

Resource: https://www.mathsisfun.com/combinatorics/combinations-permutations.html

22-Applications Combat Encounters

Ex: Fighting Fantasy Gamebooks

Primary Statistics

1. Roll one six-sided die and add 6 to the number rolled to compute your initial SKILL.
2. Roll two six-sided dice and add 12 to the number rolled to compute your initial STAMINA.

During your adventure you will often encounter hostile creatures which will attack you. In most cases you will have to resolve battles with the algorithm described below.

Combat Mechanics

1. Roll two six-sided dice for your opponent. Add its SKILL score to the total rolled, to find its Attack Strength.
2. Roll two six-sided dice for yourself, then add your current SKILL score to find your Attack Strength.
3. If your Attack Strength is higher than or equal to your opponent's, you have wounded it, so subtract 2 points from its STAMINA score. If not, it has wounded you, so subtract 2 points from your STAMINA score.
4. Begin the next Attack Round, starting again at step 1. This sequence continues until the STAMINA score of either you or your opponent reaches zero, which means death.

Q: If a player character with a SKILL score of 9 was engaged in combat with an opponent with a SKILL score of 6, which entity is more likely to win the next combat round?
Aka: What is the actual probability that the player character is going to win the next combat round?

Player skill: 9
Player attack strength: 2d6+9

Opponent skill: 6
Player attack strength: 2d6+6

Player wins the next combat round if their attack strength is higher than or equal to the opponent's attack strength.
Aka: What is the probability that $2d6 + 9 \geq 2d6 + 6$?

Let’s start by simplifying the problem..

1. Roll ONE SIX-SIDED DIE for your opponent. Add its SKILL score to the total rolled, to find its Attack Strength.
2. Roll ONE SIX-SIDED DIE for yourself, then add your current SKILL score to find your Attack Strength.
3. If your Attack Strength is higher than or equal to your opponent's, you have wounded it, so subtract 2 points from its STAMINA score. If not, it has wounded you, so subtract 2 points from your STAMINA score.
4. Begin the next Attack Round, starting again at step 1. This sequence continues until the STAMINA score of either you or your opponent reaches zero, which means death.

Q: How often is 1d6 ≥ 1d6?

Lecture 17
Mar 14 2023

22 - Linear Algebra and Physics in Games

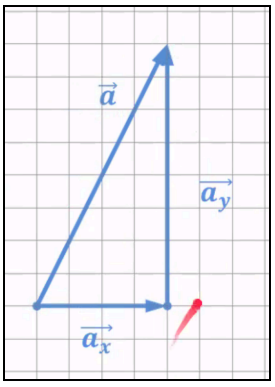
Goal: understand different concepts we use in game development, recognize some applications, perform some calculations yourself.

KEY TAKEAWAYS
- Calculate the distance between two points
- Normalize and multiply a vector
    - And the applications
- Formula to calculate dot + cross product
    - And Godot functions
- Applications of the dot + cross product

Linear Algebra

All about linear sets of equations like $2x - 3y = 1$ and $4x + 1y = 9$.
We also consider vectors and vector spaces.

Vectors are often described as:
1. A point in space
2. A direction and magnitude (line formed from (0, 0), same as #1)
3. A list of numbers (computer science)
4. A useful math construct (not very useful for us)



A **two-dimensional vector** has two values.
Ex: [1, 2]

A **three dimensional vector** has three values.
Ex: [1, 2, 3]

Typically we consider these in a **vector space**, like “x, y” or “x, y, z”.

We can break a vector a into its x and y components to create a right angle triangle.

Q: **How far apart are two characters?** → Pythagorean Theorem
It’s very common in math to consider vectors as a number for the “length” and an angle in degrees.
Ex: 10 long vector at 45 degrees
However, in game development, we represent vectors as components (length, x-axis, and y-axis).

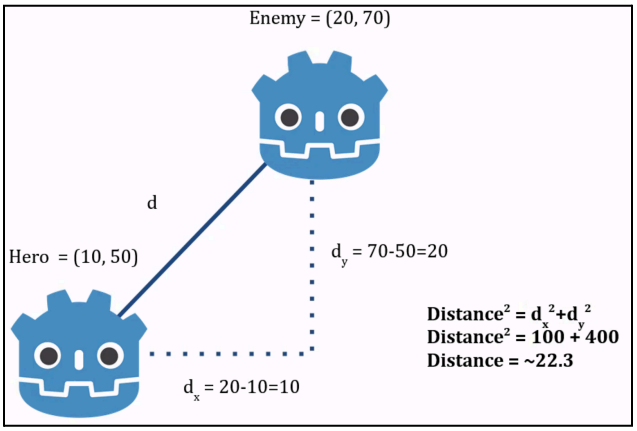
What if we have the components x and y, and need to find the length? We use the Pythagorean Theorem.

$$hypotenuse^2 = opposite^2 + adjacent^2$$

$$hypotenuse = \sqrt{opposite^2 + adjacent^2}$$

If we have the length d but want to find d_x or d_y, we pretend the origin is 0, 0.

What is the distance between the hero and the enemy? $\sim 22.3\ units$

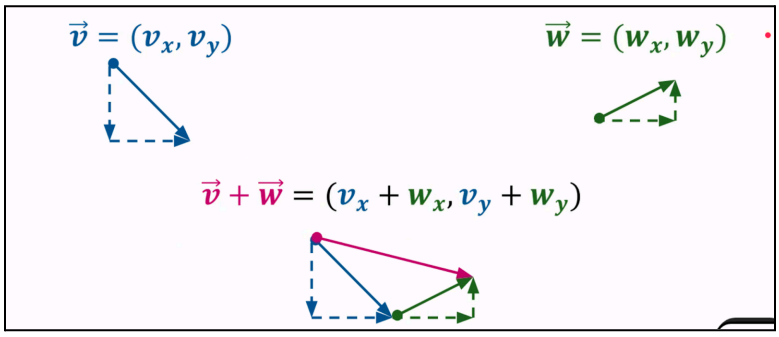


Square root is an expensive math application, so we often use $distance^2$ instead of $distance$ when comparing distances.

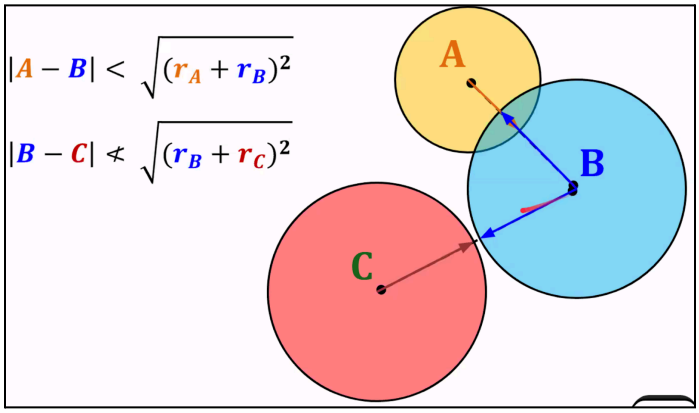
Beware: “length” sometimes refers to the number of elements, not magnitude.
In Godot, we can get the length of a vector with my_vector.length().
And the distance between two vectors with a.distance_to(b)

The length, or **magnitude**, is often represented as lines, eg. |A| or ||A|| means the magnitude of vector A.
“What is the magnitude/length of a vector?” == “From [0,0], how long is the vector?” == “How far apart are two points?”

We can **add vectors** of the same dimension together by adding their components.
[1, 2] + [2, 3] = [3, 5]
This can be done with multiple vectors at a time, and works as expected with subtraction.



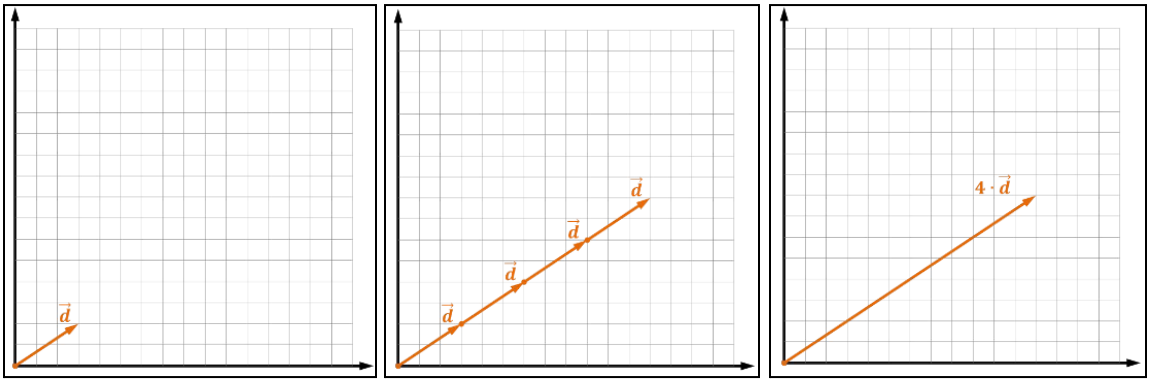
Knowing the distance between two vectors is useful when **detecting collisions between circles**.



We can see that A and B are colliding.

We can **multiply vectors** by scalars. This is the same as adding the vector to itself multiple times.

Ex: $2 \times [1, 5] = [2(1), 2(5)] = [2, 10]$



Q: How do we make sure characters move at a constant speed in all directions?

→ normalize vector

Consider the following:

- **Hero is at (0, 0)**
- **Hero's speed is 10 pixels per second**
- **Make a velocity vector and add it to our location?**
  - **... Going down?**
    - **(0,0) + (0, 10)**
      - **sqrt((0 - 0)^2 + (10 - 0)^2) = 10 pixels travelled**
  - **... Going right?**
    - **(0, 0) + (10, 0)**
      - **= sqrt((0 - 0)^2 + (10 - 0)^2) = 10 pixels travelled**
  - **... Going up-right?**
    - **(0, 0) + (10, 10)**
      - **d = sqrt((10 - 0)^2 + (10 - 0)^2) = sqrt(200) = ~14 pixels!**

We can see here that moving diagonally is faster than moving up/down/left/right.

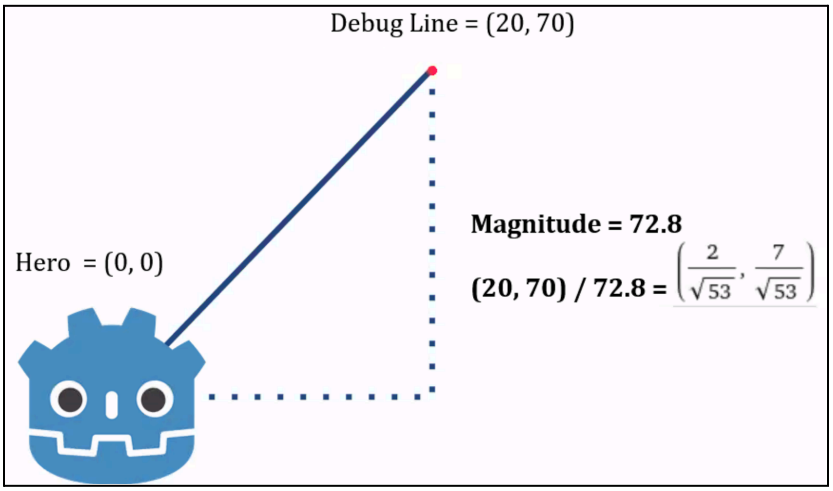
Steps to correct this ("normalize"):

1. Find what direction we are going.
2. Make that vector magnitude = 1.0.
3. Multiply the result by our intended speed.

A useful concept in game development is "**unit vectors**". A unit vector is a vector that has a magnitude of 1. We think of the unit vector as just a direction. We can multiply the unit vector by a scalar to provide it with a magnitude.

We normalize a vector by dividing it by its magnitude.

Ex: What is the unit vector? → $(\frac{2}{\sqrt{53}}, \frac{7}{\sqrt{53}})$



We get a vector in the same direction, but one that only points out 1 unit so that left/right/up/down movement is the same speed as diagonal movement.

In Godot, we can get a normalized vector by calling

```
var a = Vector2(0, 100)
print(a.normalized())
```

Q: **How do I know if I'm facing an enemy?** → Dot product

We can multiply vectors in two different ways:

1. **Cross product**: produces a vector
2. **Dot product**: produces a scalar

The dot product is related to the angle between two vectors and can help us find it.
We have two formulas to calculate the dot product:

① If we have coordinates and want the dot product:

$$\vec{a} \cdot \vec{b} = a_x \cdot b_x + a_y \cdot b_y$$

② If we have (or want) the angle:

$$\vec{a} \cdot \vec{b} = |\vec{a}|\,|\vec{b}|\,cos\,\theta$$

$$cos\,\theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}|\,|\vec{b}|}$$

$$\theta = cos^{-1}\left(\frac{\vec{a} \cdot \vec{b}}{|\vec{a}|\,|\vec{b}|}\right)$$

$$\theta = arccos\left(\frac{\vec{a} \cdot \vec{b}}{|\vec{a}|\,|\vec{b}|}\right)$$

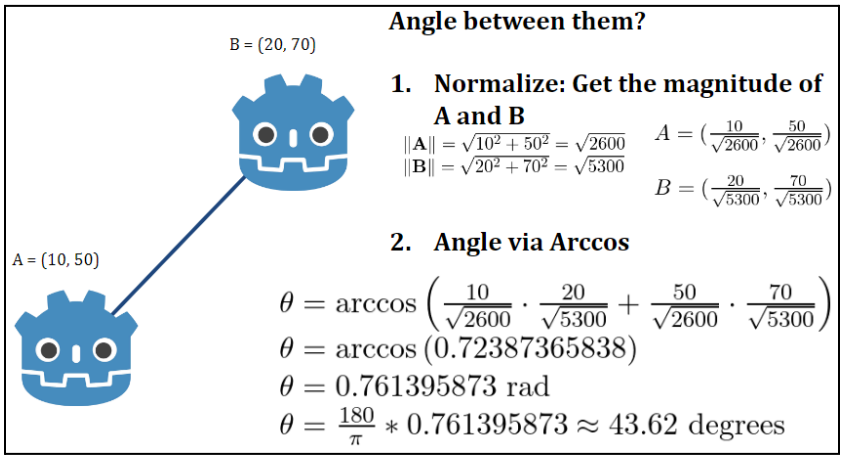
And if we normalize A and B to have a magnitude of 1…

$$\theta = arccos\left(\frac{\vec{a} \cdot \vec{b}}{1 \cdot 1}\right)$$

$$\theta = arccos(\vec{a} \cdot \vec{b})$$

$$\theta = arccos(a_x \cdot b_x + a_y \cdot b_y)$$

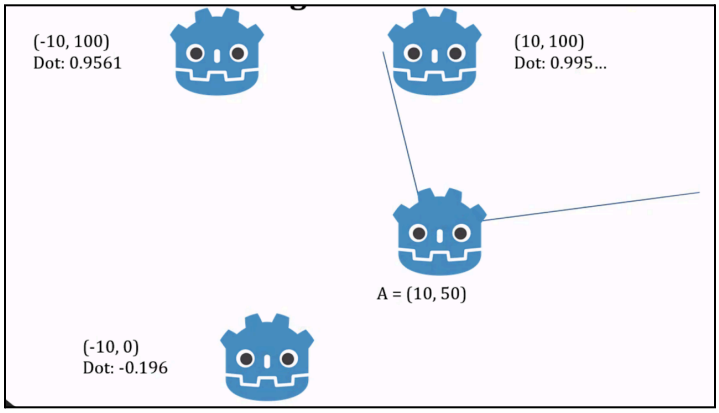
What is the angle between these two icons? (assuming (0, 0) at bottom left)



How does the dot product of normalized vectors change at different locations?

→ dot product is positive when enemy is in front of player

→ dot product is negative when enemy is behind player



The **dot product** tells us if the enemy is within the player's range of vision. We can create "vision cones" and detect if one entity is facing another without worrying about the final angle.
Eg: If the dot product is within 0.8, then the guard can see the player

Simulation: https://kidscancode.org/godot_recipes/3.x/math/dot_cross_product/

Q: **If I need to rotate from one angle to another, do I go clockwise or counterclockwise?**
→ Cross Product

Dot product tells us if the enemy is behind us or not. Cross product tells us which direction to turn in order to face the enemy.

How does the **cross product** of normalized vectors change at different locations?
→ cross product is positive when enemy is to the left of player (turn counterclockwise)
→ cross product is negative when enemy is to the right of player (turn clockwise)

Cross product is normally calculated for **3D**, and is calculated as follows:

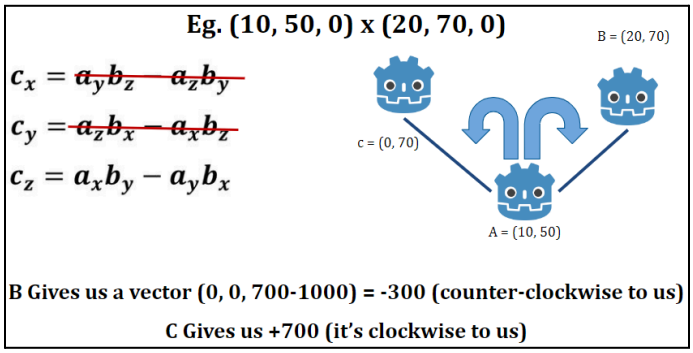
$$\vec{a} \times \vec{b} = (a_y b_z - a_z b_y,\ a_z b_x - a_x b_z,\ a_x b_y - a_y b_x)$$

Our output vector C is:

$$c_x = a_y b_z - a_z b_y$$
$$c_y = a_z b_x - a_x b_z$$
$$c_z = a_x b_y - a_y b_x$$

In **2D**, we sub 0 for z and take a scalar result.
This means that $c_z = a_x b_y - a_y b_x$ is the only vector component we need to calculate.

Ex:



Result of the cross product of A and B is negative, so the player turns right to face B.

Result of the cross product of A and C is positive, so the player turns left to face C.
In Godot, we use a.dot(b) and a.cross(b) to find the dot and cross product, respectively.

Dot and Cross Applications Review

Both are most useful when vectors are normalized.

Dot Product (·):

- Returns a scalar representing how wide the angle is between two vectors.
- Tells us if something's in front of or behind a vector.
- Easily rearranged to find the angle between.
- Parallel in the same direction ⇒ dot product of 1.
- Parallel in the opposite direction ⇒ dot product of -1.

Cross Product (×):

- Returns a vector in 3D, but a scalar representing magnitude in 2D.
- Tells us if something's left or right of a vector.
- 90 degrees to the left ⇒ cross product of 1.
- 90 degrees to the right ⇒ cross product of -1.

Additional Resources

- Dot and Cross Product in Godot: https://kidscancode.org/godot_recipes/3.x/math/dot_cross_product/
- 3B1B Essence of Linear Algebra: https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab
- Immersive Linear Algebra: http://immersivemath.com/
- Khan Academy Linear Algebra: https://www.khanacademy.org/math/linear-algebra
- Essential Math for Game Developers: https://www.youtube.com/watch?v=DPfxjQ6sqrc

Practice

Create some problems! Sub out values and draw.

If Guard is at (x, y) and facing toward (w, z), and Player is at (a, b), is the guard facing the player?

a. **If the guard can see M pixels out, can the guard see the player?**
b. **If guard wants to face the player, should guard rotate clockwise or counter clockwise?**
c. **If guard wants to run toward the player at speed N, what should their movement vector be?**
d. **If both players are are circles of radius R, when can we tell when the guard has reached the player?**

How to solve:
G: (10, 50); dG: (30, 100) (or 30 degrees); P(20, 70); N: 10; R: 5

Given this example, solve a, b, c, d.
Guard: $G = (10, 50)$
Player's distance from guard: $dG = (30, 100)$ aka $30°$
Player: $P = (20, 70)$
Speed: $N = 10$
Radius: $R = 5$

a. **If the guard can see M pixels out, can the guard see the player?** → Dot product
→ Get direction that guard is facing
$dG - G = (30, 100) - (10, 50) = (30 - 10, 100 - 50) = (20, 50)$
Guard is facing (20, 50).

→ Calculate dot product of where guard is facing and where P is facing
(example is unfinished so try it out yourself)
Positive: facing player
Negative: not facing player
(can give window ike 45 deg window)

→ If guard has a max vision distance, can the guard see the player?
If G can see 20 pixels out, is the distance between G and P less than 20?
Around 14 pixels between (from earlier)
(square distance)

b. **If the guard wants to face the player, should the guard rotate clockwise or counter clockwise?** → Cross product
Positive: counter clockwise (in Godot)
Negative: clockwise (in Godot)

c. **If the guard wants to run toward the player at speed N, what should their movement vector be?**

1. Take direction
2. Get magnitude with pythagorean theorem
3. Divide direction by magnitude and run that this speed

d. **If both players are circles of radius R, when can we tell when the guard has reached the player?**
If radii added together is less than distance of two players, then they are colliding

---

Physics

We will be looking into **classical mechanics** (things moving as the result of forces).

Kinematics is a subfield of classical physics, and focuses on movement without considering the forces involved.
This is why in Godot, KinematicBodies do not take other forces into account (which results in precise control and predictability). Instead, we consider kinematics with respect to time.

Scalar: just magnitude
Vector: magnitude and direction

Name, unit	Definition	Calculation
Position	Vector representing a point in space, position is described relative to (0, 0) or (0, 0, 0).	Vector looks like $\begin{bmatrix} x \\ y \end{bmatrix}$ or (x, y)
Speed, ms^{-1}	Scalar representing change in position over time in a single axis	$speed = \frac{\Delta position}{\Delta time}$
Velocity, ms^{-1}	Vector representing rate of change in position over time across all axes	$velocity = \frac{\Delta position}{\Delta time}$
Acceleration, ms^{-2}	Vector representing how velocity changes over time	$acceleration = \frac{\triangle velocity}{\triangle time}$
Mass, kg	Scalar relating to the “weight” of an object, affects forces	$mass = \frac{force}{acceleration}$
Force, N, aka $kg \cdot ms^{-2}$	Vectors adding together to affect position	$force = mass \times acceleration$
Time, s	Some idea of the passage of time	eg. delta or ticks

Kinematics is all about unit conversions and rearranging formulas.

Ex: **We want to go 10 pixels / second? Delta is 0.1 seconds?**

$\frac{10\,p}{1\,s} = \frac{x\,p}{0.1\,s}$

$(0.1s)(\frac{10\,p}{1\,s}) = x\,p$

$1\,p = x\,p$

∴ 1 pixel per delta is equivalent to 10 pixels per second

Velocity can be represented as:

- m/s
- $km/hour$
- $pixels/tick$

The most well-known form of **acceleration** is gravity, which is $\frac{+9.8\,ms^{-1}}{1\,s} = +\ 9.8\,ms^{-2}$.

Let's consider our time frame in evenly-spaced **ticks**. Let's call position x, velocity v, and acceleration a.

$x_{final} = x_{initial} + v$

$v_{final} = v_{initial} + a$

$a = \frac{force}{mass}$

If we're dealing with **delta** times, remember to account for partial seconds travelled as time isn't constant.

The method above is derived with integration and makes the most sense to use for our game loops. However, there are **kinematics equations** to solve position for any arbitrary time.

$position = v_i t + \frac{1}{2} a t^2$

$position = \frac{(v_f + v_i)t}{2}$

What forces are there? Let's take a look at some of the physics properties of Rigidbodies and PhysicsMaterials in Godot.

- **Gravity**: A constant, downward force; primary source of acceleration.
- **Bounce**: How "springy" a body is
- **Friction**: A force opposing the direction of movement, negative acceleration
- **Inertia**: Resistance toward rotation
- **Linear Damping**: A negative force constantly reducing velocity
- **Beyond!**: You can always add_force() to a RigidBody
- **Absorbent**: Reduces the forces of colliding objects

If we do use RigidBodies, it's really common for us to **fake forces**, rather than trying to simulate real ones. Maybe forces redirect a player's direction away from edges, or keep characters upright, or nudge a projectile back on course.

Impulses are one such force we might apply - a one time, immediate applied force to a body.

In Godot, we can call apply_impulse() or apply_torque_impulse() to nudge a RigidBody, or to apply a “rotational” nudge to it.

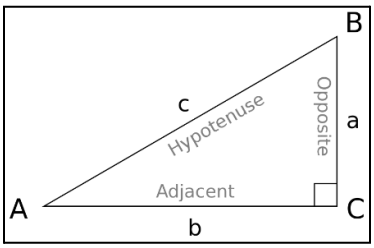
Physics is a big application of linear algebra!

Lecture 18
Mar 16 2023

23-Continued Applications in Math (not assessed)

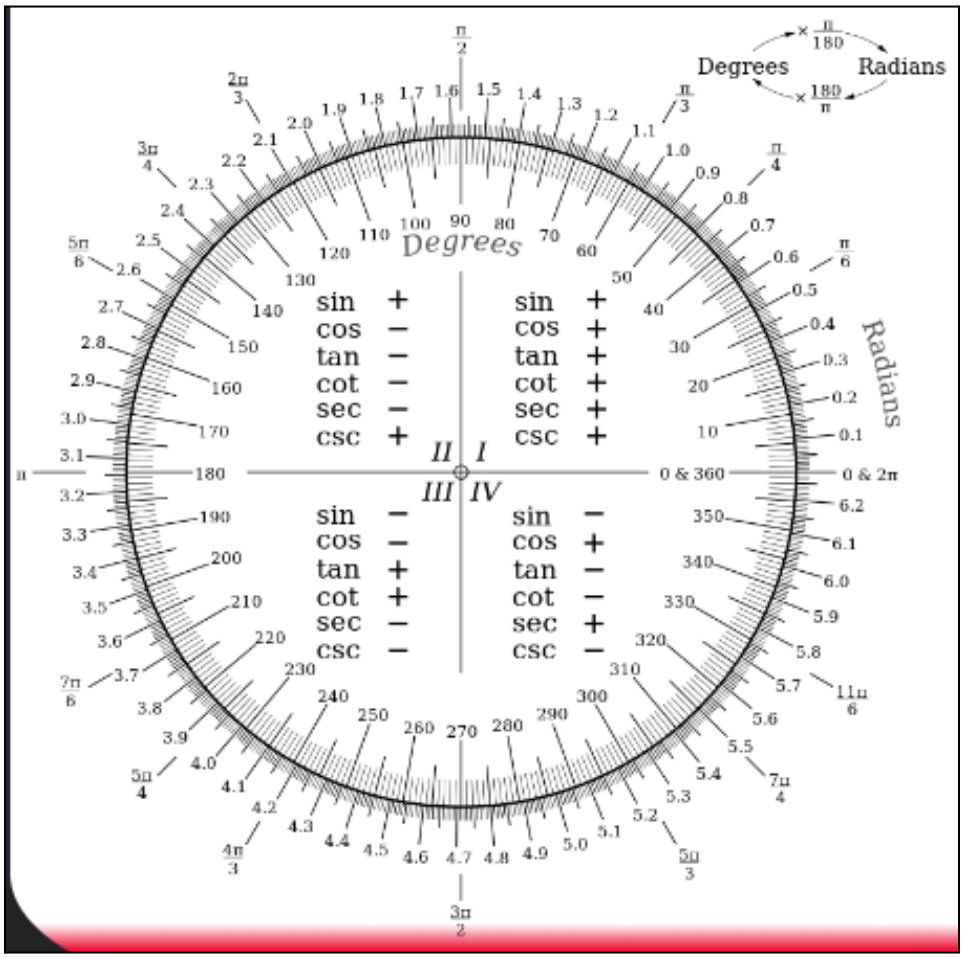
Trigonometry - Study of Angles in Triangles

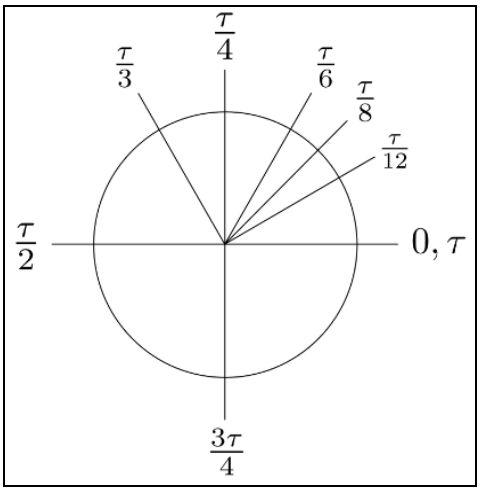
$$\sin A = \frac{opposite}{hypotenuse} = \frac{a}{c}$$
$$\cos A = \frac{adjacent}{hypotenuse} = \frac{b}{c}$$
$$\tan A = \frac{opposite}{adjacent} = \frac{a}{b} = \frac{a/c}{b/c} = \frac{\sin A}{\cos A}$$



Converting between Degrees and Radians

- Every function in Godot handling angles uses radians.
- We can convert using the global functions deg2rad() and rad2deg().
- $360° = 2\pi = 1\tau$ (“tau”)





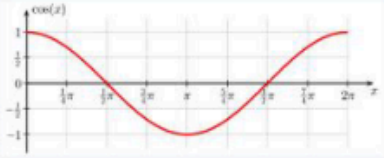
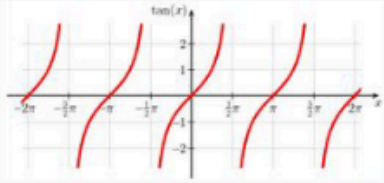
Applying Trigonometry

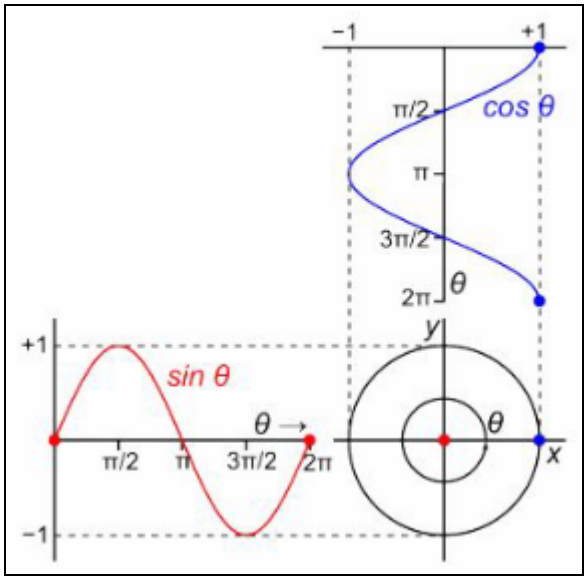
When we know an angle and radius and we want the x and y components, $x = radius * cos(angle)$ and $y = radius * sin(angle)$.

Moving a projectile at a certain angle? $(x, y) \mathrel{+}= (v_x * cos(angle), v_y * sin(angle))$

Periodic Functions

Periodic (Trigonometric) Functions

Function	Period	Domain	Range	Graph
sine	2π	$(-\infty, \infty)$	$[-1, 1]$	
cosine	2π	$(-\infty, \infty)$	$[-1, 1]$	
tangent	π	$x \neq \pi/2 + n\pi$	$(-\infty, \infty)$	



Here you can see the relationship between the unit circle and sin/cos waves.
You put in an angle and get back a value between -1 and 1, inclusive.

Periodic Function Transformations

Try it out yourself! https://www.desmos.com/calculator

- We can stretch periodic functions larger by multiplying them by a. $a\ sin(x)$
  - Ex: sin(x) vs 3sin(x)
- We can shift periodic functions to the right by b by subtracting b. $sin(x - b)$
  - Ex: sin(x) vs sin(x-3)

Application of Periodic Functions

We can put ever-increasing numbers into functions and get a result bouncing between -1 and 1.
We can apply this to get:

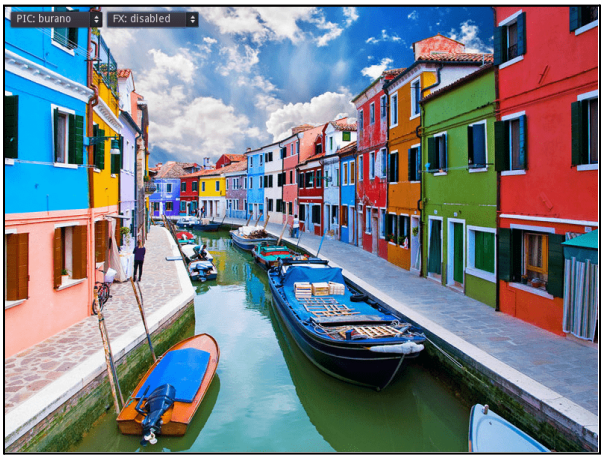
- Floaty powerups moving up and down
- Volume tweaking to give a weird feeling
- Fading between colours
- … and everything that would normally require some loop logic.

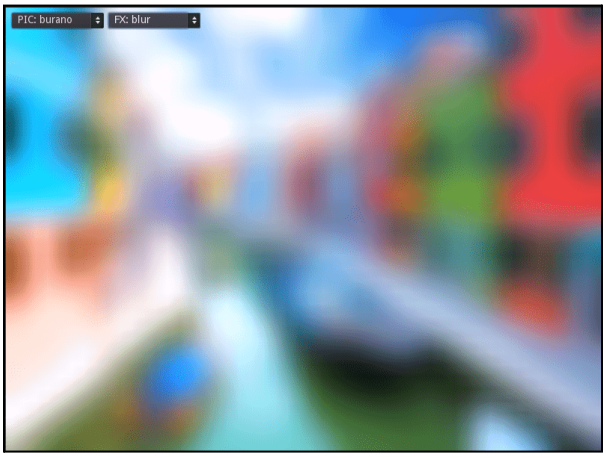
Another application is shader development.

Shaders - Pixel-by-Pixel

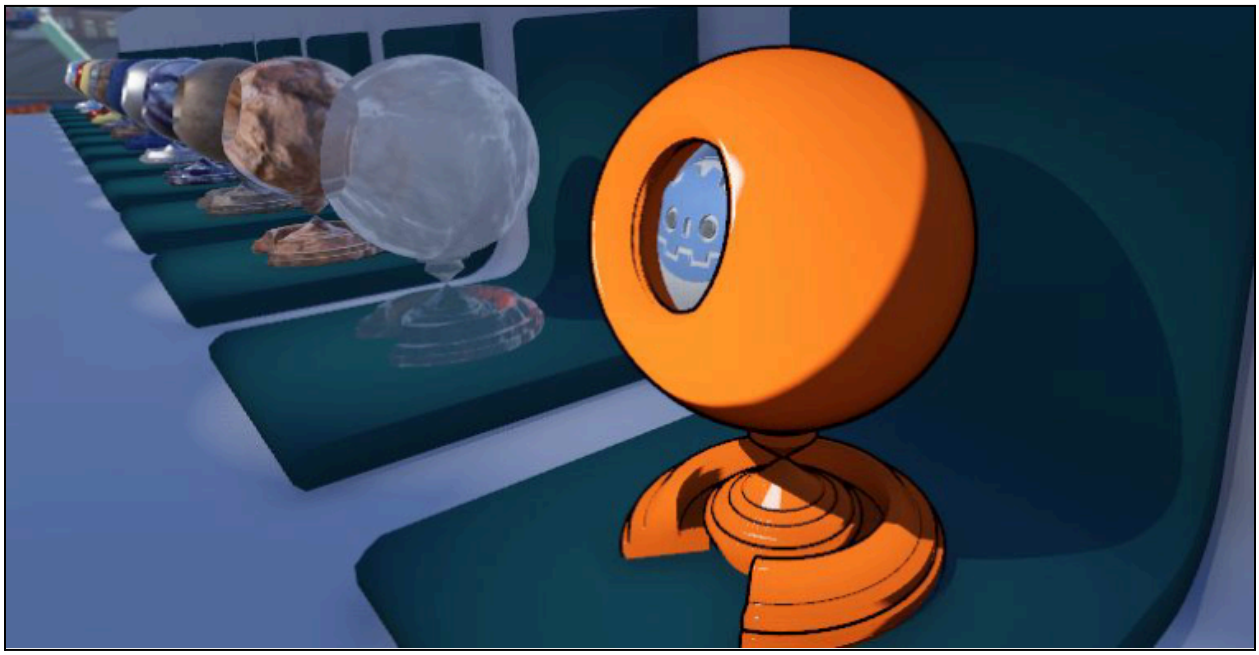
- Shaders are the code your graphics card runs to draw things to the screen based on source data.
- When working in engines at an entry-level, we mostly think about shaders as code we run after all the regular graphics work is done.
- They work on each pixel simultaneously! This extreme parallel capability is why crypto mining and data science are so good on graphics cards.
- **Vertex shaders** affect the position of vertices in a 3D model.
- **Pixel shaders** affect the colour/appearance of pixels.

- We can apply these shaders to the entire screen.
    - https://godotengine.org/asset-library/asset/122

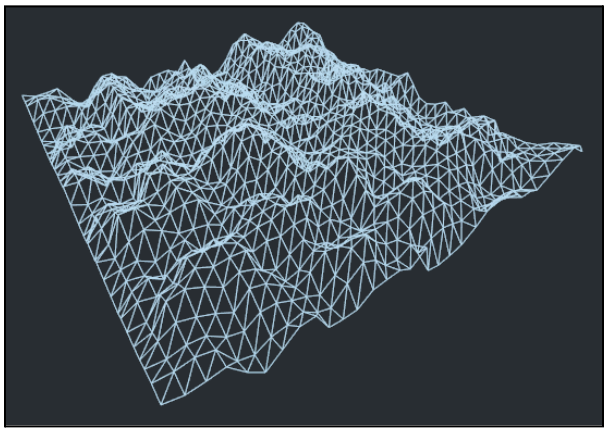




- We can apply them as a material on an object.



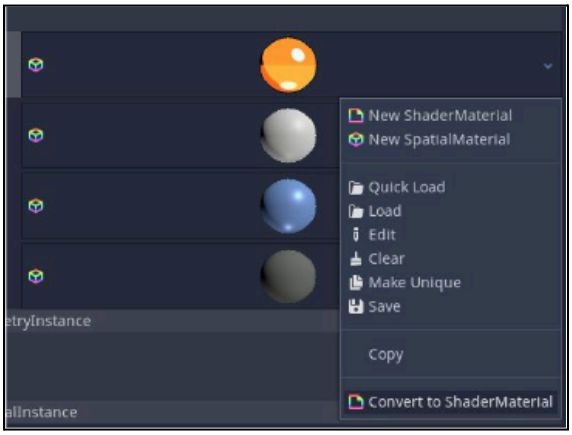
- We can use it to mess with the mesh of a model, eg. creating procedural terrain
    - https://docs.godotengine.org/en/stable/tutorials/shaders/your_first_shader/your_first_3d_shader.html

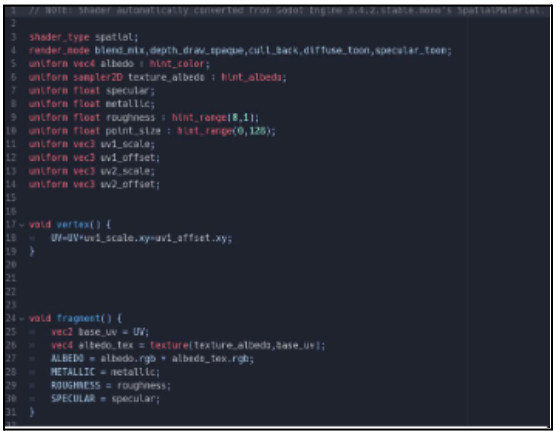


- Because shaders run every frame, on every pixel, they have to run really fast and be able to work without knowing what the surrounding pixels are like.
- As such, we tend to use periodic functions like sin and cosine as functions of time.

Abstraction of the Shader

- Game engines usually hide a lot of shader code from you.
- In Godot, we make “SpatialMaterials”, a default shader with lots of parameters we can tweak.
- We can convert these to “ShaderMaterials” to expose the default code and make changes.





Writing Shaders

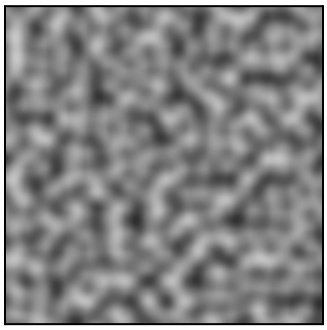
There’s a few different languages to write shaders in.

- Godot has its own, but it’s far from standard.
- Most people learn **glsl** or some form of it.

The Book of Shaders is a great resource for learning primitive shader concepts in an interactive way. https://thebookofshaders.com/

Noise Functions

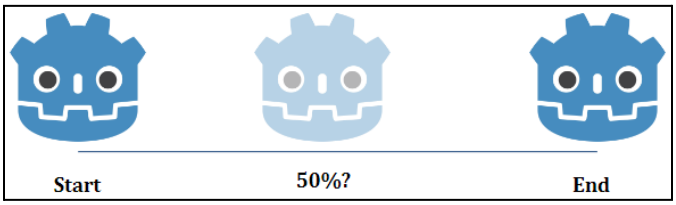
- Noise functions are functions which generate interesting sequences of numbers.
- An example of this is **Perlin Noise** which is used for all kinds of procedural generations.
    - https://en.wikipedia.org/wiki/Perlin_noise



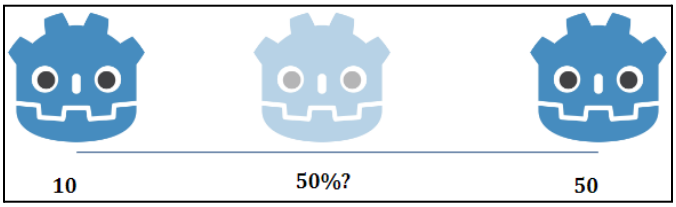

Linear Interpolations - LERP from Here to There

- We use lerp *all the time* in game dev.

If you have two values, what the value at $x\%$ along the way between them?



1. What is the range? $50 - 10 = 40$
2. What is $x\%$ of that range? 50% of $40 = 20$
3. What value is $x\%$ along the way? At 50%, we should be at $10 + 20 = 30$

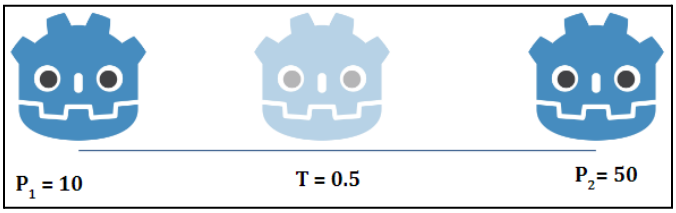


Or, we can view this in terms of position p.

$$p_t = p_1 + t(p_2 - p_1)$$

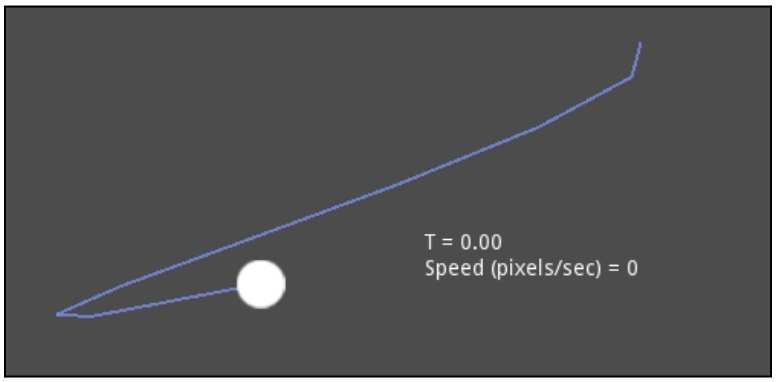
$$p_t = 10 + 0.5(50 - 10)$$

$$p_t = 30$$



LERPing in Godot

- Godot provides lots of tools for interpolation.
    - https://docs.godotengine.org/en/stable/tutorials/math/interpolation.html
- We can lerp with a vector, a colour, a curve…
    - We can interpolate position from a path
        - https://docs.godotengine.org/en/stable/tutorials/math/beziers_and_curves.html



    - We can interpolate curves and colours

```
6  export(Curve) var my_curve
7  export(Color) var one
8  export(Color) var two
9  var t = 0
10
11 func _process(delta):
12     t+=delta
13     $Icon.position.y = my_curve.interpolate(abs(sin(t)))*100
14     $Icon.modulate = one.linear_interpolate(two, abs(sin(t)))
15
```

- Curves are quite a useful interpolation!
    - Remember all of our talk on using curves for loot drops, exp, jump speeds, enemy spawn rates…

Matrix Transformations

Using matrix operations, we can handle all of the below at once, really quickly:

- **Scale an image** by multiplying each coordinate.

$$T = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}, \vec{p} = \begin{bmatrix} x \\ y \end{bmatrix}, \text{then } T\vec{p} = \begin{bmatrix} ax \\ by \end{bmatrix}$$

- Rotate an image by applying our trig functions.

$$T = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \vec{p} = \begin{bmatrix} x \\ y \end{bmatrix}, \text{then} T\vec{p} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

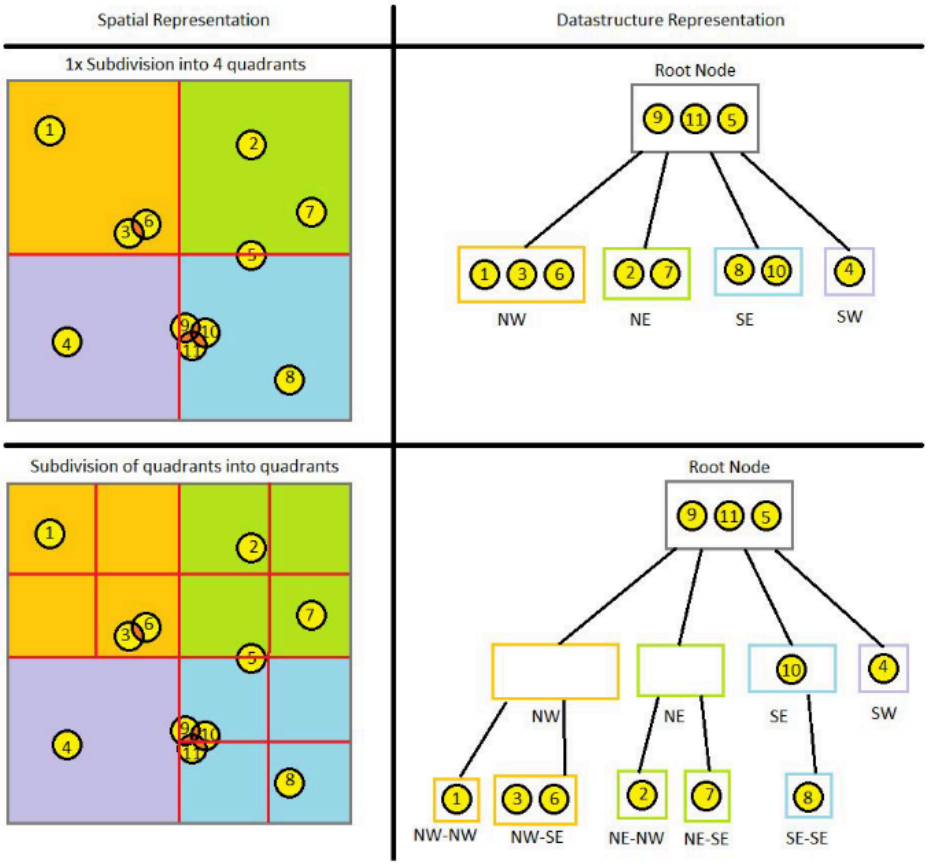
- Translate an image (not as easy, will not go into detail).

Discrete Math in Games - Data Structures and Algorithms

Ever wondered how we can process so many collisions at once, efficiently?
Or how a game like Minecraft can know how to load things smoothly? Discrete math!!

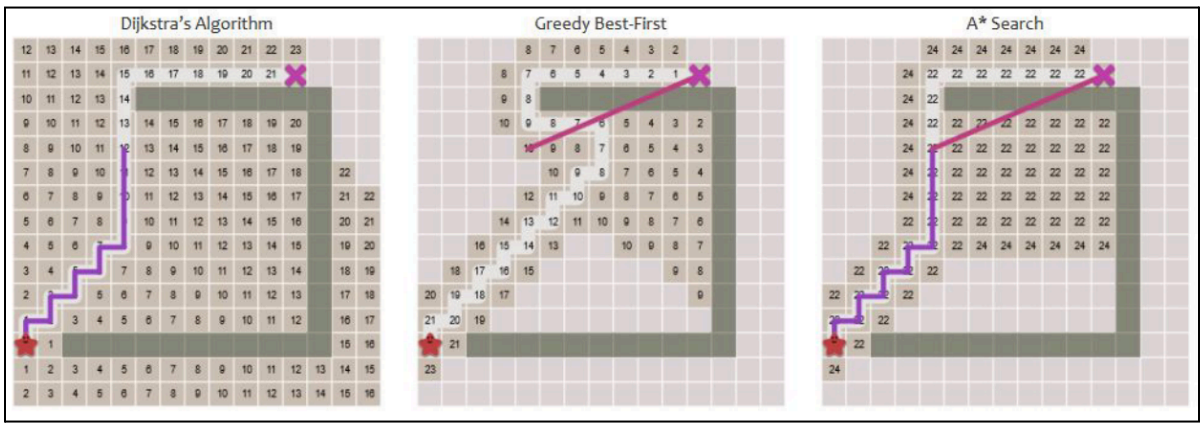
Octrees

- https://www.gamedev.net/tutorials/programming/general-and-gameplay-programming/introduction-to-octrees-r3529/
- We use “Octrees” or something similar to subdivide out space.



Pathfinding

- https://www.redblobgames.com/pathfinding/a-star/introduction.html
- We use trees to represent our space to perform operations like path finding, eg. the A* algorithm



RedBlobGames is a great resource for game algorithms and data structures.
https://www.redblobgames.com/

Closing Notes

- Recall that game engines are good at making what the game engine is good at making.
- An experimental game – a game requiring specific optimizations or a game just outside the space of an engine – will require custom code.
- Just about every established company working with a consumer engine is still writing lots of custom code to make it work for their game.
- Don't get tired down to a single tool. You'll end up fighting the engine to make a game it's not good for, or avoiding game ideas because your engine doesn't allow it.

Lecture 19
Mar 21 2023

Quiz opens Wednesday 5pm and closes Friday 5pm
About 14 questions, multiple choice, primarily covering math.

- Numerical relationships and sequences
- Probability for games
    - Expected value
    - Probability of event occurring
    - Probability trees
    - When to multiply and when to add probability of events
    - Critical hit?
    - What if the opponent can defend themselves?
    - Are you likely to win an encounter?
    - If you win one encounter are you likely to win the next?
- Combinatorics
    - Number of ways things can happen
- Linear Algebra
- No physics (i think)

60 minutes, can write all the way until the end since there's a grace period to submit at the end.
Will have access to course slides during the test.

24-World Building and Non-Linear Narratives

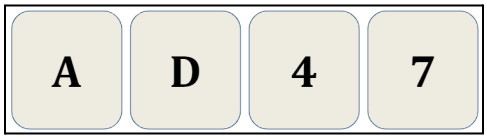
Recall from last discussion about narratives

Context

If the game loop relies on us knowing the player's mental model to provide feedback

- How can we set proper expectations?
- How can we help build a strong initial model?
- Will our game make sense?

We discussed the 4-card problem. It's harder to solve problems with less context.



Affordances from theming

Intuitive affordances: asks for an interaction

- don't jump on spike
- Pull on the big hand pulls

Symbolic affordance: affordance through culture

- Gas canister blows up
- Red cross symbol (don't actually use this symbol in your games lol)

Story as a Malleable Mechanic

Story World Economy Action Underlying Fantasy

(Most flexible) ←——----------→ (Least flexible)

Why?

Underlying Fantasy

- The core promises and pillars of the game
- Not much can be changed about this

Action

- Actions are programmed in and are part of the core game loop
- Not much can be changed about this

Economy

- Also involves a loop

World

- Flexible, but this must be established before the story can be built

Story

- Story is flexible up until the publishing of the game.
- Ex: have an annoying bug like green fog which will not go away? Just change the story of the game so that it's set in space.

How do we tell stories in games?

Stories aren't just text.

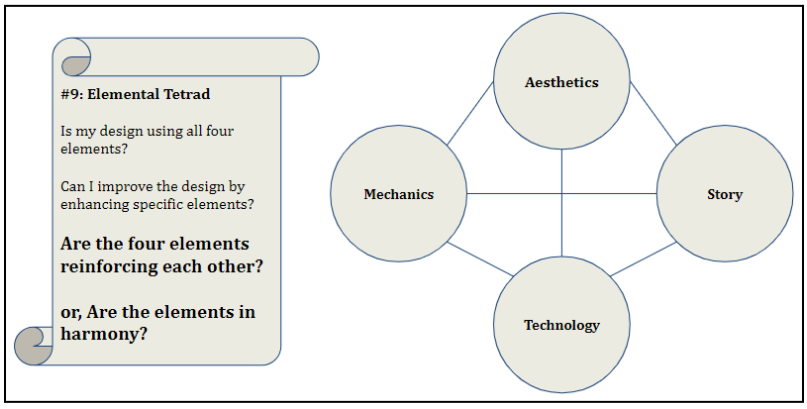
Concept: Time out of gameplay

- Too much dialogue
- Too many books
- Game is just cutscenes

Story in Games

- Many people "hate games with stories". Remain open. You might just dislike poorly implemented stories.
- Stories, themes, narratives - just like our other game tools, can be used well and be used poorly.

Harmonizing your Elements (The Elemental Tetrad)



Motivating Your Demographic

Narrative: Games as drama

Rules:

- Avoid considering game elements "always good" or "always bad".
- Must consider context, implementation, harmony.
- Does this element support the gameplay? Does this element motivate our players?

Challenges and Affordances of Interactive Stories

Ingredients for a Good Story

1. A story is just a character with a goal and obstacles preventing them from reaching it.
    - Problem solving? Surprising conflicts? Perfect for games!
2. Obstacles should be mechanically meaningful.
    - Relate the obstacles back to the action of the game.

Challenges of Interactive Fiction

- Good stories have an underlying sense of unity. All elements work together, build off of each other, and are enjoyable.
- Interaction can get in the way; which choices are actually good? How can we curate this?
    - Which choices do we want to give the player?

Our **mission** as designers is to provide the player with a fun, engaging experience. To engage, we have verbs.

- **Games**: run, shoot, jump, climb, throw, punch
- **Film**: talk, ask, negotiate, convince, argue, shout

→ Not usually in games, but if they are, they appear in the form of cutscenes.

Multiple paths

- Achievers and explorers will want to play multiple times… but is it really still surprising the second time?
- Just too many outcomes. If we give the player 3 choices, and want a unique outcome for each, that's manageable but might not be that exciting.
    - With 10 choices - we have 88, 573 choices.
    - With 20, we have - 5, 230, 176, 601 outcomes!

Ex: Detroit Become Human - many people dislike the endings.

Essential Experience

We need to remember that all parts of the game come together to present an experience, and that central to that experience is interaction. Story is more than text and exposition.

Mechanics as Metaphor: Story Through Mechanics

Content warning: examples in this section cover topics such as domestic abuse, death, war, the holocaust, slavery. While these are heavy examples, it's purely because these demonstrate very clearly how we become emotionally connected to stories.

Human Perception of Apparent Behaviour

Studies show just how much we empathize and project with otherwise abstract things.

Recall: the game "Loneliness", what's the story?

- Square tries to meet other squares and the other squares keep running away from you.
- This is a great example of "mechanics as metaphor".

We see interesting "mechanics" being used to tell stories in film - our brains are wired well for storytelling. Much of what we know about story in games began in film techniques.

One of the early film "mechanics" showed how juxtaposing images changes our emotional interpretation: https://www.youtube.com/watch?v=_gGl3LJ7vHc.

- People were asked "What did you think about the actor's emotions/ability to act?"
- People praised the "subtle acting" but it was the same clip each time.

We apply emotions to non-human things: https://www.youtube.com/watch?v=n9TWwG4SFWQ.

Games and Experience

In games, we have a very delicate power - the ability to seat the player into an experience and have them walk in someone else's shoes. We must handle serious topics with respect.

Ex: Brenda Romer's game "*Train*"

Playing cards to load "Passengers" onto a train, rolling dice, and accelerate or decelerate yours or your opponent's train.

"The game is over when it ends."

At the end, it's revealed that you've (maybe) been following orders, written on an SS typewriter, to send Jewish prisoners to a concentration camp.

- Train was designed in response to Brenda's child being confused after class about the conditions of slaves being forcibly transported away from their homes.
- Simply hearing about, or reading about these things didn't connect well enough.

https://www.wsj.com/video/designer-discusses-train-a-holocaust-game/EA433B7B-E8D2-44CF-ABA0-93191F886BC8.html

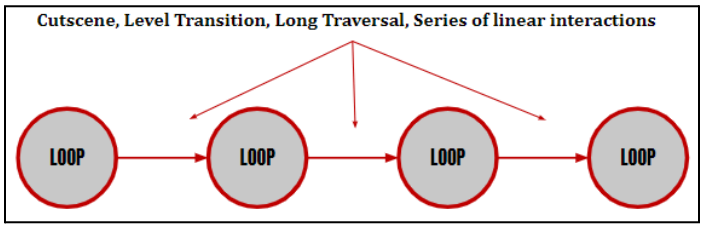
Mechanics as Metaphor
A key insight is that you are building the boundaries of experience through mechanics, but we still need to understand what we're trying to accomplish with our story.

Going forward, recognize that when we're talking about storytelling in games, we aren't just talking text. Consider how each of these concepts can be done through the medium of games, interactions, and experience design.

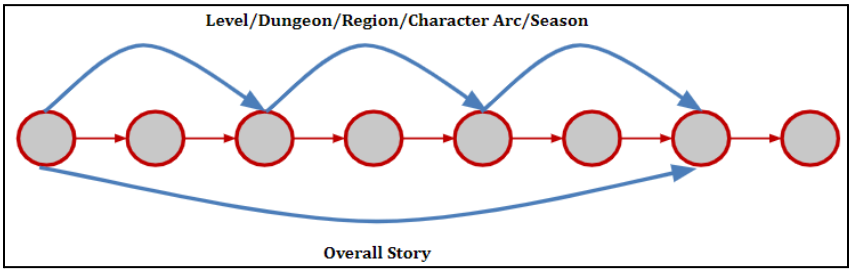
Narrative Structures: How have games told stories?

Idea: String of Pearls

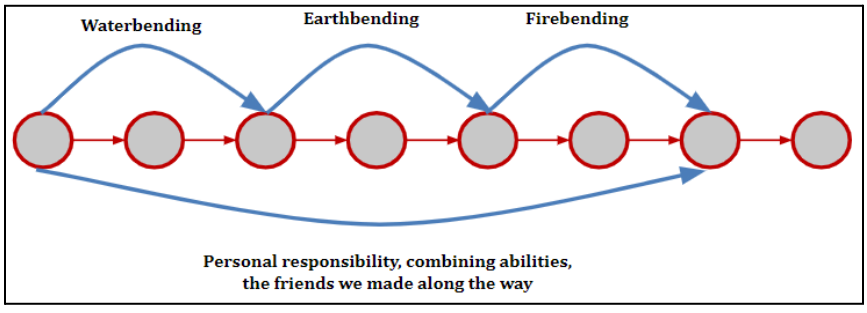
- Relates to our discussion on arcs and loops
- The player gets sections of non-linear gameplay, tied together with linear story progression.



Arcs keep players engaged. Like storylines in TV shows.

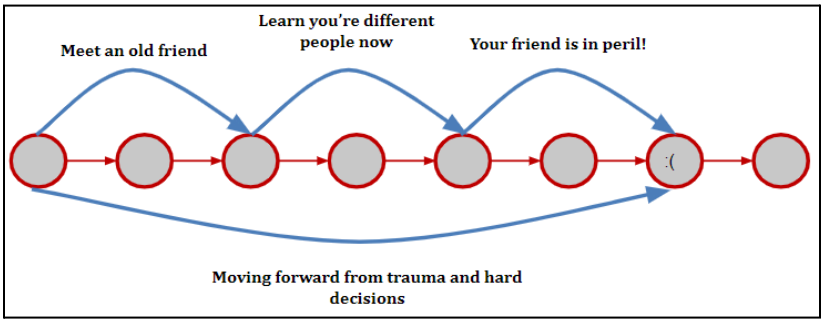


Each string should feel like a **reward**. You can gain **aesthetics** such as powers.
Example below: Avatar: The Last Airbender.

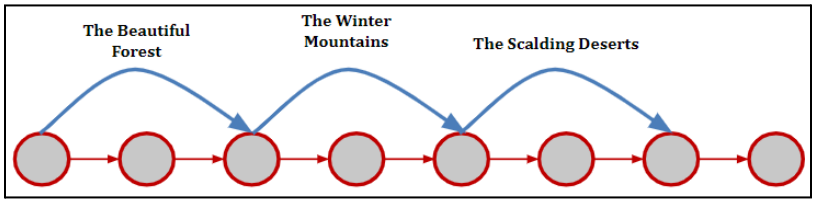


Relationships can be **rewarding**.

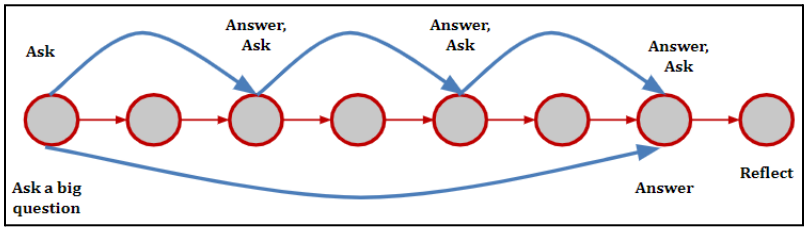
Example below: Life is Strange



Sensation rewards!



Asking and answering questions (over and over) with a reflection at the end.



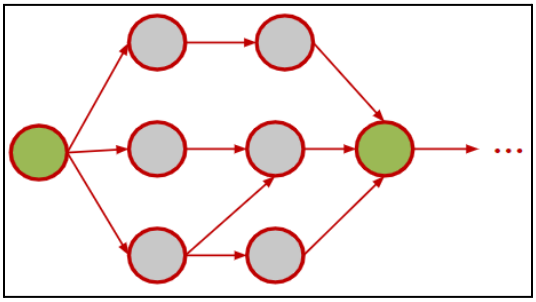
Non-Linearity in String of Pearls

- Three techniques: Sub-branching, Artificial Choice, “Blending”.
- Most often used <u>all at the same time</u>.

Sub-branching

Each pearl is a branching narrative with a fixed start/end.

- Your actions do not affect the ending, just some story in between the start and end.

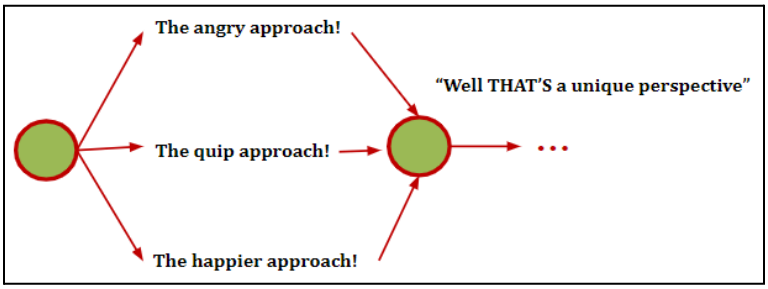


Artificial Choice

The choices are linear, but each pearl is designed such that any choice makes logical sense for the outcome.

Ex: Same voice line for each reaction, but it makes sense in each scenario

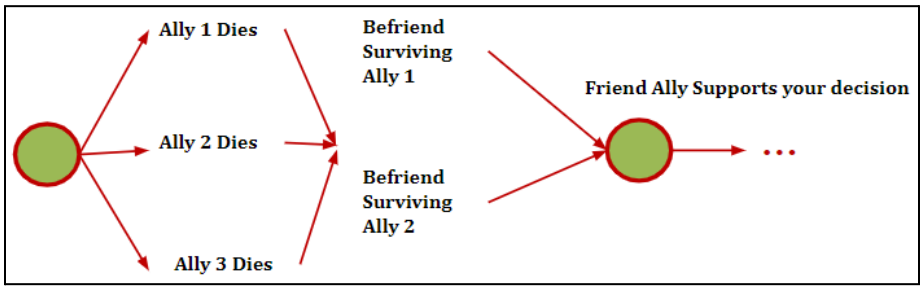
Ex: the Kuleshov Effect (video from above where people interpreted subtle expressions when there were no changes in expressions)



Blending

Each pearl affects *state*, but not transitions.

- Achieved on our end easily, usually by just updating a variable.



Call-Backs

Not really a technique, but it's often suitable to simply refer back to a decision that was made without having to change anything.

- Can appear in the form of a character saying something, or a flag appearing as a different colour based on previous actions.
- Achieved on our end using a boolean which determines which action will be taken based on the player's previous actions.

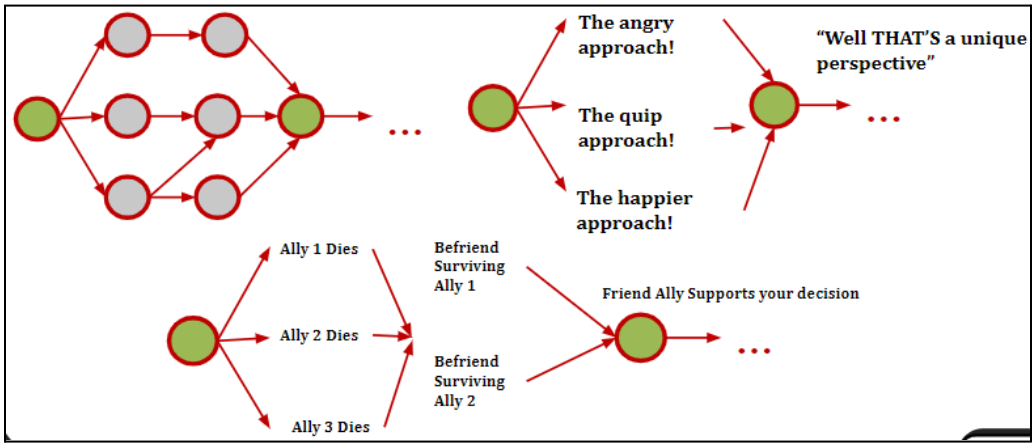
Mix & Match (will come back to this)

By mixing and matching, you can develop very immersive experiences that seem to react to your choices! We can (and should) design these with **interest curves** in mind.

- You can make high-level decisions on how your game transitions across beads: New planet? New power? New biome? New look? New ally? New goal?

False Branching Narratives

There are many approaches to achieve “fake” branching narratives.



Is False-Branching LYING?

- There’s usually pushback talking about these structures. “Ugh, I hate games that don’t actually branch.”
- Spoiler: You haven’t played a game that truly branches on every decision.
- Think about how this branching could’ve been improved.
    - You aren’t supposed to notice the false branching. “If you do your job right, nobody will notice you did it at all”.
    - If you noticed it, it was probably a bad implementation.

The Story Machine: Making Experiences that Make Stories

We’ve talked in the past about **meta-narrative aesthetics**

- Where the story is told by the player after the game, not the game telling them.
- This is quite a new concept to try and design for!
- We see more and more of this through social games, sandbox games, and let’s plays.

Keeping in Mind the Hardships

When designing more open experiences, we need to remember how things go wrong:

- Explicit branches lead to massive branching.
- Extreme breadth is hard to keep interesting.
- Most stories aren’t interesting on their own.

Story Machine

Some great examples of games as “story machines”:

- **Minecraft**: Simple rules, powerful tools, emergent conflicts, short-term and long-term goals, minimal direction, passing knowledge from one player to the next
- **DayZ**: High tension, simple downtime with emergent conflicts, unpredictable infrequent encounters with humans, “campfire stories” forum to discuss stories
- **Among Us**: Chaotic interactions with people, mandatory group storytelling, simple gameplay, focus on paying attention

Questions to consider:

- How can you add more choices to achieve goals?
- How can more *types* of conflict arise in-game?
- Can characters personalize their story somehow?
- Do the rules (and feedback loops) promote good interest curves?
- Who can your players tell the story to? Why will they care?

Game Story Breakdown: Star Wars: The Old Republic

About SW:TOR

- MMORPG set in the Star Wars universe.
- Lots of interesting narrative shapes used together!

Overview

- You play one of two factions (Empire or Republic)
- You play with one of two morality (Light side, Dark side)
- You play one of 8 classes (Bounty Hunter, Trooper, Sith Warrior, Jedi Knight, Smuggler, etc.)
- You start on one of four planets (two planets per factions, two classes per planet)
- There are many planets you progress through, one after another

About SW:TOR

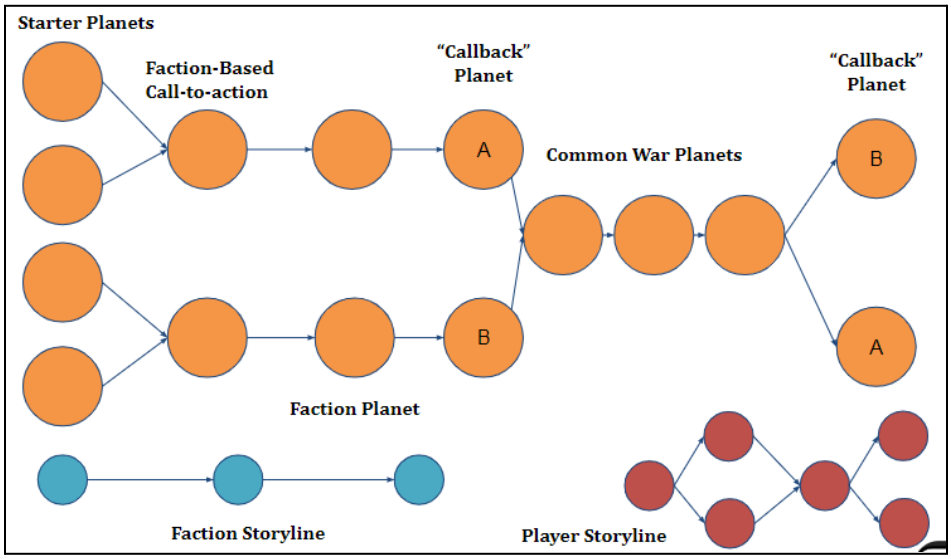
To keep the stories feeling fresh, they've layered the pearls in different ways.

- Faction Story: Empire or Republic
    - Linear pearl-string, branching is only within each pearl
    - The Empire-Republic branch MERGES ~75% of the way through the game
- Planet Story
    - Each planet has a perl-string with some local branches, but ultimately
- Class Story
    - Each character class (Jedi Knight, Bounty Hunter) has a story just to them - high branching!
- Ally Story
    - Each ally has a simple, zero branching storyline

Branching in SW:TOR

- Lots of room for explosive growth!
- They keep it constrained with pearl-strings.
- Ultimately, only the player's story is affected by their decisions and the rest is handled through callbacks.
- Decisions in the other stories reward growth in the personal story.

SW:TOR Overview



On a second playthrough, a new faction provides ~5 unique planets of a total 26, but new stories on many more, a whole new player storyline, and whole new faction storyline for **~50% of the game**.

Publishers who are offered narrative games will ask about its "**seen content**".

- **Seen content** is the percentage of content the average play will see on a single playthrough.
- The goal with these structures tends to be maximizing seen content.

Feeling Alive

- These varied, layered structures help make the narrative feel more alive.
- They even tackled the replayability issue by offering bonuses to replays to skip common content (i.e. content the players already say in the first playthrough)!

Starting at slide 67 (worldbuilding) next time.

Lecture 20
Mar 28

Open to showcase final project game? → fill out consent form

24-World Building and Non-Linear Narratives

The Space of the Story: Introduction to Worldbuilding

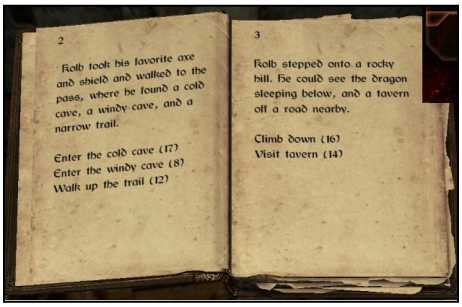
Game world is composed of:

- **Setting**: Where, when ← this lecture
- **Actions**: What, how
- **Characters**: who ← next lecture
- **Plot**: why

What is Worldbuilding?
Worldbuilding emphasizes **where** and **when**, the **lore** of the world.

The **concept** of "worldbuilding" originated from J.R.R. Tolkien's *The Lord of the Rings*. Tolkien designed languages, and made sure each name had a history, and frequently referenced distant lands to make the world larger and more alive than it is.
→ "**Distant Mountains**" method: lots of effort naming distant places, people, and events that are never encountered by the main character

Where do we see this in games?
→ **Explicit lore** (lore books, codex entries, item descriptions)



→ **Implicit lore** / **environmental storytelling** (skeletons on couch, notes pasted to a wall)

Environmental Storytelling

Setting Diversity

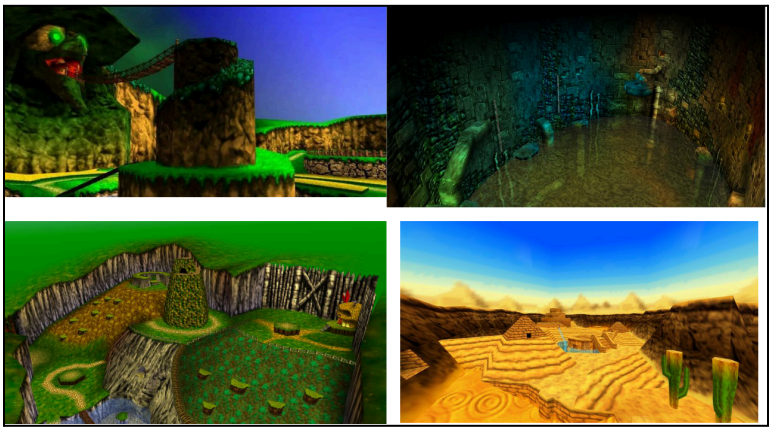
Have you ever felt like the game was progressing overall but there was little progress? And all the levels feel the same? And you sometimes forget where you are?

Games are in motion. Players need to be able to recognize things at a glance. As such, the environmental theme should change as the game progresses. Diversity reminds the player of their progression.

Orienting the Player

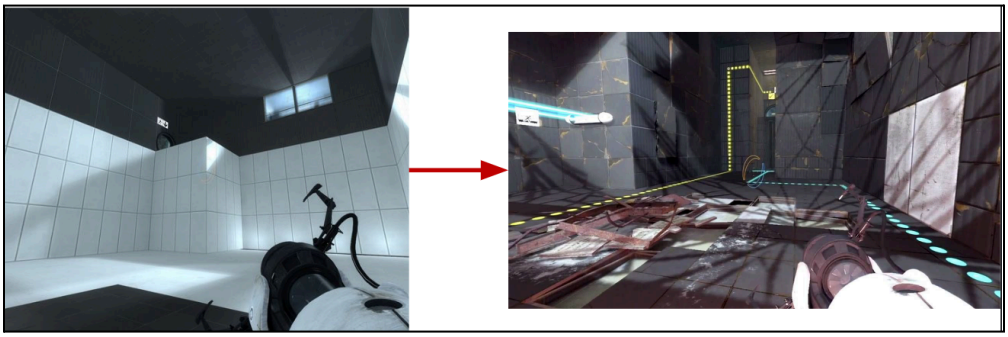
- Reinforce the player's progression by changing the environmental theme.
    - New enemy styles
    - New dynamics
    - New music
    - New colour palette
- The theme can tell a story.
    - The world-trotting diversity
    - Moving into a broken world

Example: Banjo Kazooie (1998)



Example: Skyward Sword (2011)

Meaningful Thematic Diversity
Example: Portal 2 (2011)



Pristine testing room → destroyed testing room

Picking a setting

You should be picking settings related to your interests and whatever promotes the experience you're designing.

However, there's a **quick rule** to help **identify a great setting** for the player:

1. It's simpler than the real world.
2. The player is more powerful than in the real world.

What makes the setting simpler? What makes the player more powerful?

Actions in the World - Making it Interactive

Characters

- Any kind of dialogue spoken by a character should tell you more about that character.
    - Tone
    - Word choice

Fictional Elements: Action

- Actions are the primary concern of the player. "I am the player, so what can I do?"
- Actions must make sense for the environment, the player's motivations, and say something about the experience.
- Actions are also a way to reward players (we have touched on this before).
    - **Rewards of Facility** allow the player to do more.
        - Offer permanent abilities players didn't have before.
        - May allow for a new way of moving between spaces.
        - Ex: Rolling into a ball in Metroid.
    - **Rewards of Access** allow the player to go to new areas.
        - Allow the player access to new locations or resources.
        - Used once.
        - Not useful to player once used.
        - Ex: keys, lockpicks, passwords.
    - More reward types here: https://gamestudies.org/1101/articles/gazzard_alison

Diversity of Actions

Keeping these two rewards in mind, we can design diverse experiences and try to theme our rewards to speak about the world.

Example: Hammerwatch (2013)
Trailer: https://www.youtube.com/watch?v=n9aGEzxogqY

Rewards of Access:

1. Keys: choose access
    a. Bronze keys (frequent)
    b. Silver keys
    c. Gold keys (very rare)
2. Puzzle
    - Performative
    - Deliberate
3. Secret entrance
    - Observant
4. Button: single-access key
    - Opens one door
    - Find the door it opened

Example: Link's Awakening Remake (2019)
Trailer: https://www.youtube.com/watch?v=ZROB4TnYH_I

Rewards of Facility:

1. Each item promotes new gameplay and new areas
    - Recall: String of Pearls
2. Puzzle unlocks one-time use keys

Rewards and Motivations

- Remember, there must be a balance.
- Rewards should be theoretically interesting, but make sure the rewards match the player's motivations.
    - Ex: Narrative regards for highly-competitive actions?
    - Ex: Rewards of cooperation for successful solo-play?

Going Beyond

- While environments can carry a lot of narrative weight, people relate heavily with characters.
- We will look into designing our environment to support impactful characters.

25 - Plot Structures and Characters

Recall: Game world is composed of:

- **Setting**: Where, when ← last lecture
- **Actions**: What, how
- **Characters**: who ← this lecture
- **Plot**: why

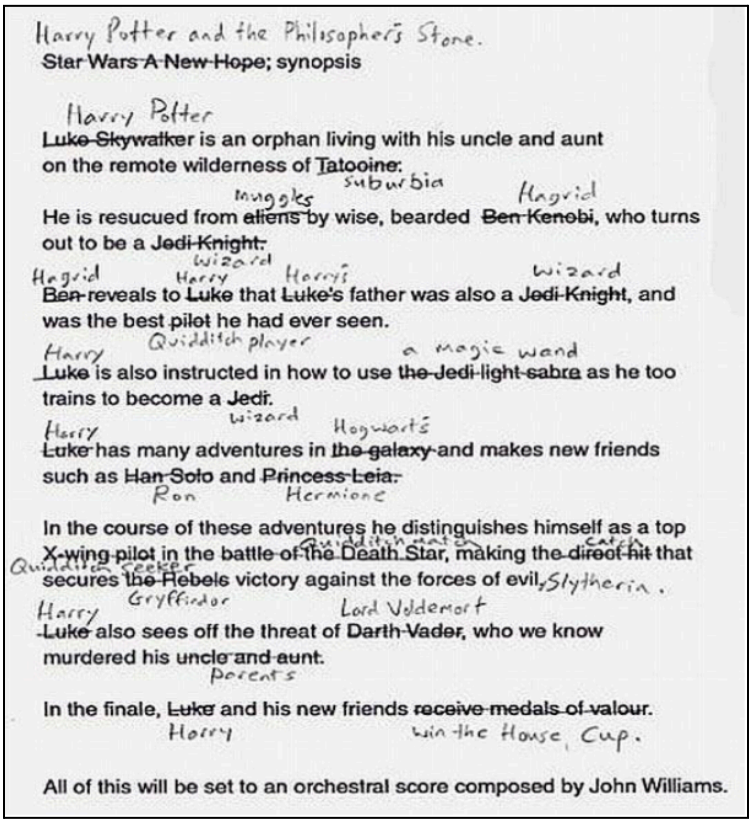
Types of Fun

- **Fantasy** is associated with "**make believe**".
- **Narrative** is associated with "**unfolding story**".

While they are different types of fun, narrative is often the vehicle to deliver fantasy.

How to write a best-seller?

First, let's take a look at the similarities between Harry Potter and Star Wars…



Harry Potter and the Philosopher's Stone.

~~Star Wars A New Hope~~; synopsis

Harry Potter

~~Luke Skywalker~~ is an orphan living with his uncle and aunt on the remote wilderness of ~~Tatooine~~. muggles suburbia Hagrid

He is resucued from ~~aliens~~ by wise, bearded ~~Ben Kenobi~~, who turns out to be a ~~Jedi Knight~~. Hagrid wizard Harry Harry's wizard

~~Ben~~ reveals to ~~Luke~~ that ~~Luke's~~ father was also a ~~Jedi Knight~~, and was the best ~~pilot~~ he had ever seen. Harry Quidditch player a magic wand

~~Luke~~ is also instructed in how to use ~~the Jedi light sabre~~ as he too trains to become a ~~Jedi~~. wizard Harry Hogwarts

~~Luke~~ has many adventures in ~~the galaxy~~ and makes new friends such as ~~Han Solo~~ and ~~Princess Leia~~. Ron Hermione

In the course of these adventures he distinguishes himself as a top ~~X-wing pilot~~ in the battle of ~~the Death Star~~, making ~~the direct hit~~ that secures ~~the Rebels~~ victory against the forces of evil, Slytherin. Quidditch seeker Quidditch keeper earth Harry Gryffindor Lord Voldemort

~~Luke~~ also sees off the threat of ~~Darth Vader~~, who we know murdered his ~~uncle and aunt~~. parents

In the finale, ~~Luke~~ and his new friends ~~receive medals of valour~~. Harry win the House Cup.

All of this will be set to an orchestral score composed by John Williams.

Both of these stories follow the "Hero's Journey".

Joseph Campbell's "Hero's Journey"

An attempt to find the common pattern that is followed by most adventure stories. There are 17 stages, split up into 3 sub-groups:

1. **Departure**: future hero is called into the unknown
2. **Initiation**: future hero is transformed into the hero
3. **Return**: hero returns in triumph

The Departure

1. The future hero lives in a mundane world
2. The future hero refuses the call to adventure
3. A second call is accepted
4. Supernatural being provides assistance
5. Hero sets out, the point of no return
6. Hero enters a dangerous and unknown world

The Initiation

1. Encountering trials and challenges
2. Bonding with a strong figure of the opposite gender
3. Encountering temptation
4. Overcoming a persuading authority
5. Apotheosis: transformation into the hero
6. Successful achievement of the goal

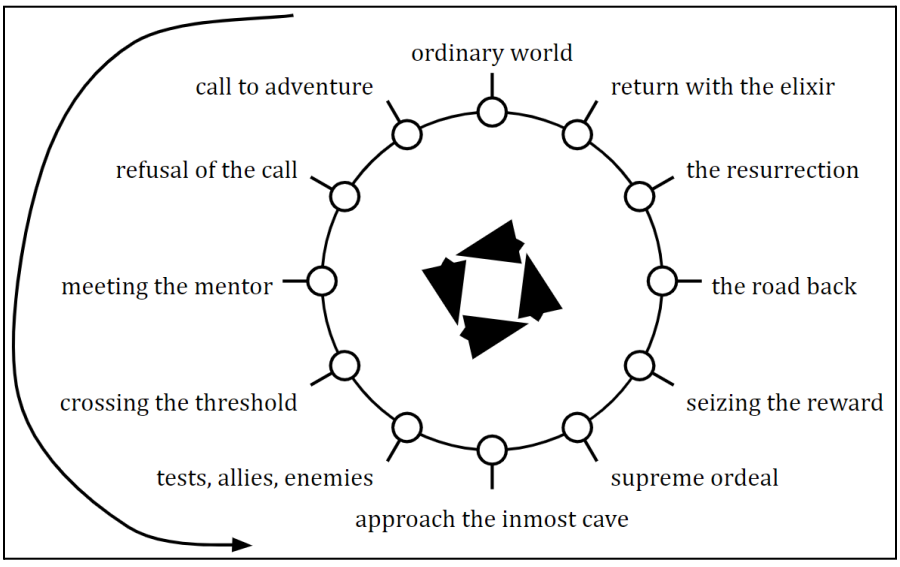
The Return

1. The hero does not wish to return
2. An urgent race home
3. A rescue from an unexpected source
4. Crossing the threshold (again) home
5. Hero has mastered both “sides” of the threshold
6. Hero has earned the right to choose

It’s important to note that the Hero’s Journey is not the starting point for your story, it’s a **lens (lens #76)**.

Vogler’s Simplified Interpretation

While Campbell’s work is extensive and formative, Christopher Vogler developed a simplified interpretation with only 12 stages for screenwriters which is more commonly used today.



On Using the Hero's Journey
We can examine a story through the lens of the Hero's Journey to identify potentially problematic areas - areas we could expand, things to insert.

Our goal is to make the player **experience** the hero's journey, not simply have it told to them.

- Make the player mechanically refuse the call.
- Provide supernatural aid which tell us more about the characters, instead of just… being.
- Let the player experience the divinity of apotheosis.

The Humanity of the Hero's Journey
At its core, it's the story of growing up. Refusing the call, experiencing loss, receiving help when needed, facing the trials of life, and returning a more grounded human to help those that are starting *their* journey.

Alternatives to the Hero's Journey
There are a lot of other story structures out there! Explore other narrative patterns.

Propp's Functions (not assessed)
Vladimir Propp developed a narrative pattern for Russian fairy tales.

1. **Absentation**: Leaves the home.
2. **Interdiction**: Forbidding act passed to the hero.
3. **Violation of Interdiction**: Violate the rule.
4. **Reconnaissance**: Villain tries to attain knowledge.
5. **Delivery**: Villa succeeds and gains a lead on the victim.
6. **Trickery**: Villain deceives the victim to acquire value.
7. **Complicity**: Victim is fooled or forced to concede and helps the villain.
8. **Villainy or Lacking**: Villain harms a family member.
9. **Mediation**: Something negative above comes to the attention of the hero.
10. **Beginning Counteraction**: Hero considers resolution, seeking items/rescuing, etc.
11. **Departure**: Leaves the home environment with a sense of purpose.

The adventure begins.

12. **First Function of the Donor**: Encounters a helper and is tested.
13. **Hero's Reaction**: Responds to actions of the future donor.
14. **Receipt of a Magical Agent**: Acquires a magical agent as consequence of good actions.
15. **Guidance**: Transferred or delivered into a vital location.
16. **Struggle**: Hero and villain meet and engage in direct conflict.
17. **Branding**: The hero is marked in some manner, a distinctive scar or item.
18. **Victory**: Villain is defeated by the hero.
19. **Liquidation**: Earlier misfortunes are resolved.
20. **Return**: Travels back home.
21. **Pursuit**: Pursued by threatening adversary.
22. **Rescue**: Hero is rescued from a chase; may be saved by another.

23. **Unrecognized Arrival**: Hero arrives, along their journey or in their destination, and is unrecognized/unacknowledged.
24. **Unfounded Claims**: …
25. ...

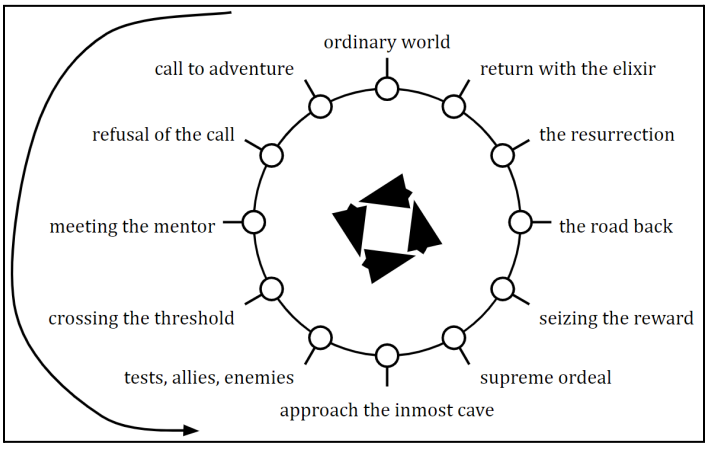
Full list here: https://tvtropes.org/pmwiki/pmwiki.php/Main/ProppsFunctionsOfFolktales

Defeating Writer's Block

Once you recognize the kind of structure you're writing, when stuck, use the lens to conquer writer's block and improve upon your story.

Oftentimes, events and characters correspond to specific stages of the Hero's Journey.

- Which Events and Characters in George Lucas's "Star Wars" (i.e., IV) would Correspond to Each of Christopher Vogler's 12 Stages?
- Which Events and Characters in J. R. R. Tolkien's "The Lord of the Rings" would Correspond to Each of Christopher Vogler's 12 Stages?



Characters in the Journey - Archetypes to Consider

Character archetypes

- Proposed by Campbell, Vogler, Propp, Schell, and many more…
- Recurring roles that are useful for character design.
- Each character receives a function.

Schell proposes these simple character archetypes for games:

1. Hero
2. Mentor
3. Assistant
4. Tutor
5. Final Boss
6. Minions
7. Three Bosses
8. Hostage

For this course, we will focus on Campbell's archetypes:

1. **Hero**: protagonist and the subject of the journey
2. **Mentor**: the teacher (often provides a gift)
3. **(Threshold) Guardian(s)**: present barriers/obstacles to the Hero
4. **Herald**: the one to issue the call to adventure
5. **Shapeshifter**: not as they appear, may shift allegiance
6. **Shadow**: the antagonist/villain in the story
7. **Trickster**: a source of comic relief (and complications)
8. **Ally**: the companion and a source of assistance

Using Archetypes

- First, you design your cast of characters.
- If you are struggling with motivations or understanding their place in the world, you can examine them through the lens of character function (lens #86).

Character Development

Which Characters in George Lucas's "Star Wars" (i.e., IV) would Correspond to Each of these Character Archetypes?

- **Hero**: Luke
- **Mentor**: Obi Wan Kenobi
- **Guardian**: stormtroopers
- **Herald**: R2D2
- **Shapeshifter**: Han Solo
- **Shadow**: Darth Vader
- **Trickster**: R2D2, C3P0, Han Solo
- **Ally**: R2D2, C3P0, Han Solo, Chewbacca

Hero's Journey, Summarized

- The hero's journey, simplified by Vogler, has 12 stages.
- We use the journey as a lens to examine our story (not as a starting point).
- In games, we strive to have the player *experience* the journey, not just be *told* it.
- To create compelling characters, we can design characters and see if they align with archetypes to determine motivations.

Writing Strong Characters

Why care about characters?

Let's remember what a game story is all about:

1. Somebody wants something, badly.
2. They're having a hard time getting it.

The avatar vs. the character

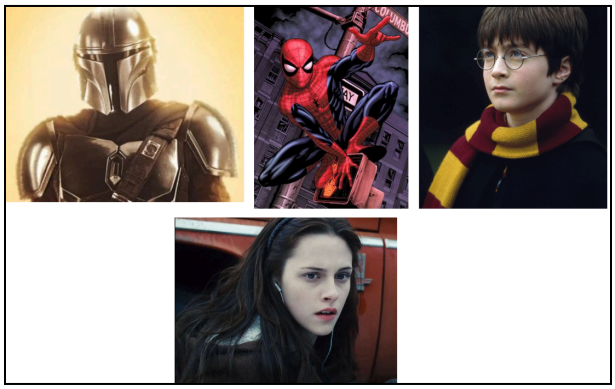
- In games, an **avatar** is the “character” which the player controls.
- While character refers to the beings in your story, it’s more accurate to say that **characters** represent those that want things badly.
- An avatar can absolutely be a character.

Avatars and the “blank state”

- Oftentimes in novels, films, and games, the main character/avatar is… kind of bland.
- Why? We are trying to project ourselves onto the avatar.
    - If we attempt to project and we meet inconsistencies with our projection, we lose engagement.
- Often, the character is some kind of “other” to the world they’re entering, so that they must learn about things as we learn about them.

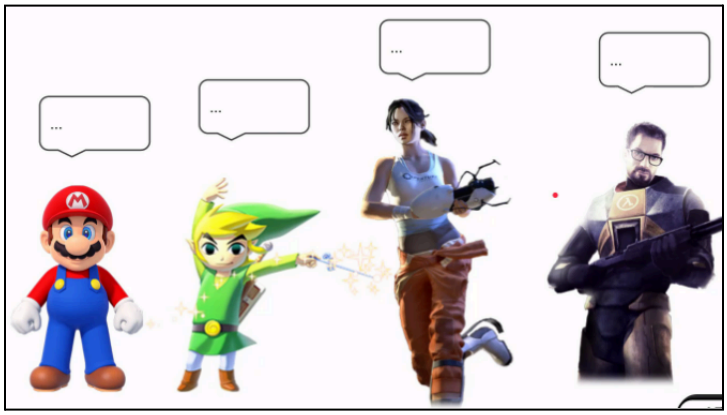
Have you ever played a game and been immediately pulled out of immersion with a line like…
“ALL RIGHT, HOTSHOT! IF YOU’RE REALLY THE GREATEST WARRIOR IN THE NINE DIMENSIONS, GO SHOOT SOME BOTTLES”
… that has made you go ”what is happening to this character?”

Examples of blank slate characters:



Talk on creating memorable characters: https://www.youtube.com/watch?v=4mgK2hL33Vw

Avatars and the Silent Protagonist



Characters and context
You don't know these character's motivations.

- The **avatar** is meant to provide **context to mechanics**. "Why can we do what we do?"
    - Ex: our hero is a plumber, so he can go into pipes.
    - Ex: our hero is a reincarnated hero of time.
- The **characters** are meant to provide **emotional context**.

But… why?
Do we NEED to have silent protagonists?
Let's keep in mind our goal: the player should be in sync with our avatar.

What is a character?
A character is someone who wants something badly.

- Protagonist wants to free the kingdom
- Minion wants their home back
- Antagonist wants control
- Love interest wants support
- Innkeeper wants to hear stories
- …

Elements of character (Who they appear to be)

1. What does the character want? → purpose
2. What do they do to get it? → action
3. Who do they seem to be? → trait

Lecture 21
Mar 30 2023

25 - Plot Structures and Characters

Being "In-Sync" with a Character

To be "in-sync" with a character doesn't imply that they must be bland - it means that we must unify the player and the character.

1. If the purpose is inconsistent, the player doesn't care.
2. If the actions don't make sense, the player won't play.
3. If the traits are out of line, things may not be as impactful as they could be.

Purpose - Emotional Context

- Usually, the designer chooses the purpose.
    - Exceptions in games like Minecraft, where the avatar is a godlike figure with the ability to shape the terrain, etc, who chooses the purpose.
- Share with the player!
    - Share a laugh between the player and character.
    - Share a secret with the character.
    - Share a heartfelt moment.
    - Share a thought.
- Make them likeable.
    - Ex: a funny, noble, sympathetic underdog
- Keep their purpose simple.
    - Escape.
    - Acquire X.
    - Don't die.

Action - Mechanical Context

- Don't make the players do something dumb.
    - You never want the player to think "why would I EVER do X when Y is obviously the best choice".
- Make sure it follows the purpose.
    - Is the action frivolous, or does it move them toward their goal?
- Enable gameplay that makes sense.
    - Ex: the best silent protagonist characters (Gone Home, Portal) tend to be games with no other characters.
    - From GDC talk (27:31), re: Portal 2:
        - "Am I the only one, who really at one point wanted to say to GLaDOS, 'you know what? You're a french fry and I have a portal gun, I'm gonna take my chances.' But you can't!"

Traits - Investment

- With a silent protagonist, we have intentionally few traits.
    - Typically just to motivate necessary story beats.
    - Ex: an engineer with card access and a crowbar with little combat experience.
- More cinematic characters will have parts of their character expressed through clothing, relationships, and dialogue.
- Player-driven characters will have character customization and backstories.
    - Hybrid is fine!

Writing Characters - Traits and Relationships

Techniques for characterization come from Schell's, Chapter 20.

Purpose of Dialogue

- Dialogue shouldn't just be about progressing the plot.
- We want the player to learn about the characters through what they say and how they say it.

From the textbook (Schell, page 378-380):

> **LESTER**: Sabu!
> **SABU**: What is it?
> **LESTER**: Someone has stolen the king's crown!
> **SABU**: Do you realize what this means?
> **LESTER**: No.
> **SABU**: It means the Dark Arrow has returned. We must stop him!

What's wrong with this dialogue? What do you know about Lester and Sabu? Nothing!!

How do we fix this dialogue? Define some traits.

- Sabu: trustworthy, short-tempered, valiant, a fiery lover
- Lester: arrogant, sarcastic, spiritual, impulsive

> **LESTER** (exploding into the room): By the Gods! Sabu, I have news! (impulsive and spiritual)
> **SABU** (covering herself): How dare you intrude on my privacy! (short-tempered)
> **LESTER**: Whatever. Maybe you don't care that the king's crown has been stolen? (arrogant and sarcastic)
> **SABU** (a faraway look in her eye): This means I must do what I promised... (trustworthy and valiant)
> **LESTER**: I pray to Vishnu this is not another story of an old flame... (Lester: spiritual and sarcastic; Sabu: fiery lover)
> **SABU**: Silence! The Dark Arrow broke my heart, and the heart of my sister—I promised her that if he ever returned, I would risk my life to destroy him. Prepare the chariot! (short-tempered, fiery lover, trustworthy, valiant)

Traits

- You can view traits similarly to game pillars.
- Start by deciding the information to convey about the plot.
    - Ex: Someone has stolen the king's crown…
- Then lean on your traits to determine delivery.
    - Ex: "Whatever… maybe you don't care that the king's crown has been stolen?"

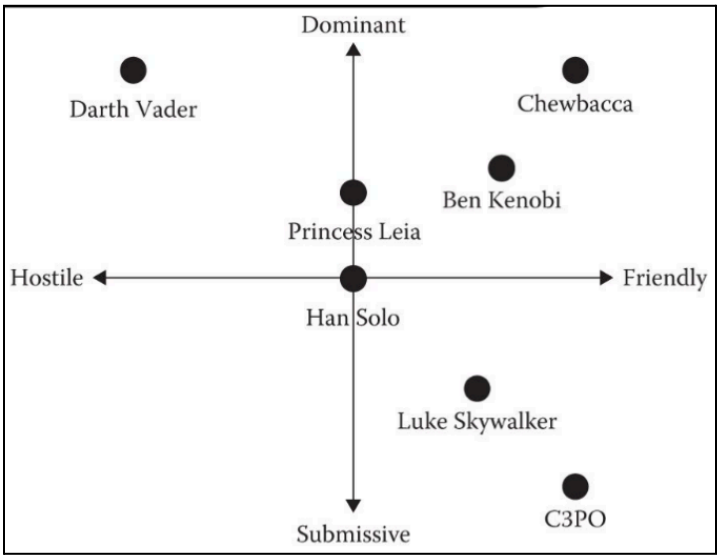
Designing Relationships - Setting the Character in Others

A character's most defining traits may be how they treat other characters.

Interpersonal Circumplex

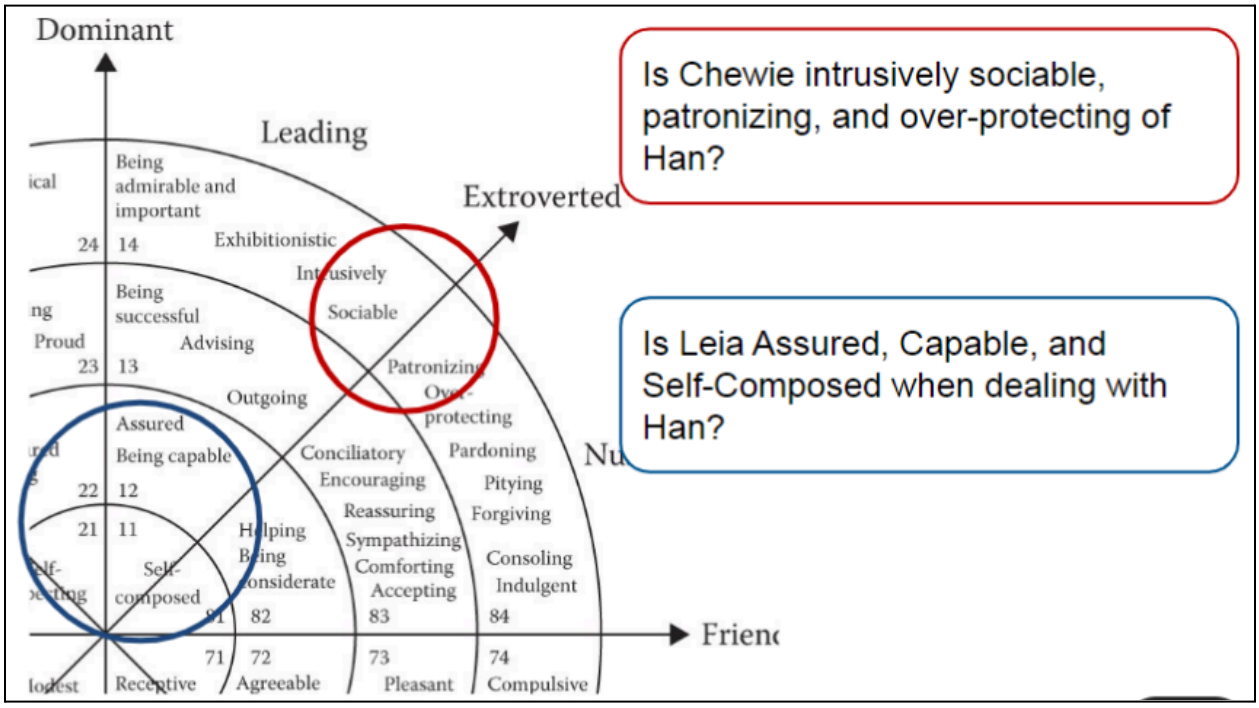
- A tool to help analyze and design character relationships.
- Brought to game design from psychology by Katherine Isbister.
- We map how a character reacts to other characters.
- Two axes: agreeableness and dominance.

Example: Interpersonal Circumplex applied to Star Wars



Using the Circumplex

- We don't always need to map relationships like this, but it can be a helpful lens into designing interactions between characters.
- The first time characters interact, figure out where they lie in relationship to one another.
- You can use the outside circle to determine the nature of their interaction.



The Character Web

- Another lens we can use.
- From the perspective of the character, write a short line about how they feel about the other characters.

Ex: Archie's Web (from the textbook)
Veronica: Archie is lured by her elegance and beauty. Though she is rich, Archie doesn't care much about that.
Betty: Archie's true love, but her insecurity constantly gives him mixed signals so he doesn't pursue her as forcefully as he might.
Reggie: Archie shouldn't trust Reggie, but he often does because Archie always tries to be a nice guy and Archie is kind of gullible.
Jughead: Archie's best friend. What they have in common is that they are both underdogs.

Character Transformation (Schell's, pg. 390)

- Strong characters don't always stay the same.
- To reflect this, we can design transformation charts.
- These charts allow us to:
    - Ground ourselves
    - View characters at a high level
    - Fill in gaps to shape characters
- We can modify these charts to also track abilities like time travel or to track how a character's choice of companions changes.

Ex: Cinderella

		Scenes				
		At Home	**The Invitation**	**The Night of Ball**	**The Next Day**	**At Last**
Characters	Cinderella	A sad, suffering servant	Hopeful, then disappointed	A beautiful princess	Suffering and sad again	Happily ever after
	Her stepsisters and stepmother	Superior and mean	Ecstatic and arrogant	Disappointed at getting no attention	Hopeful that they might fit the slipper	Disgraced and incredulous
	The Prince	Lonely	Still lonely	Fascinated with a mystery woman	Desperately searching	Happily ever after

Applying Character Tools

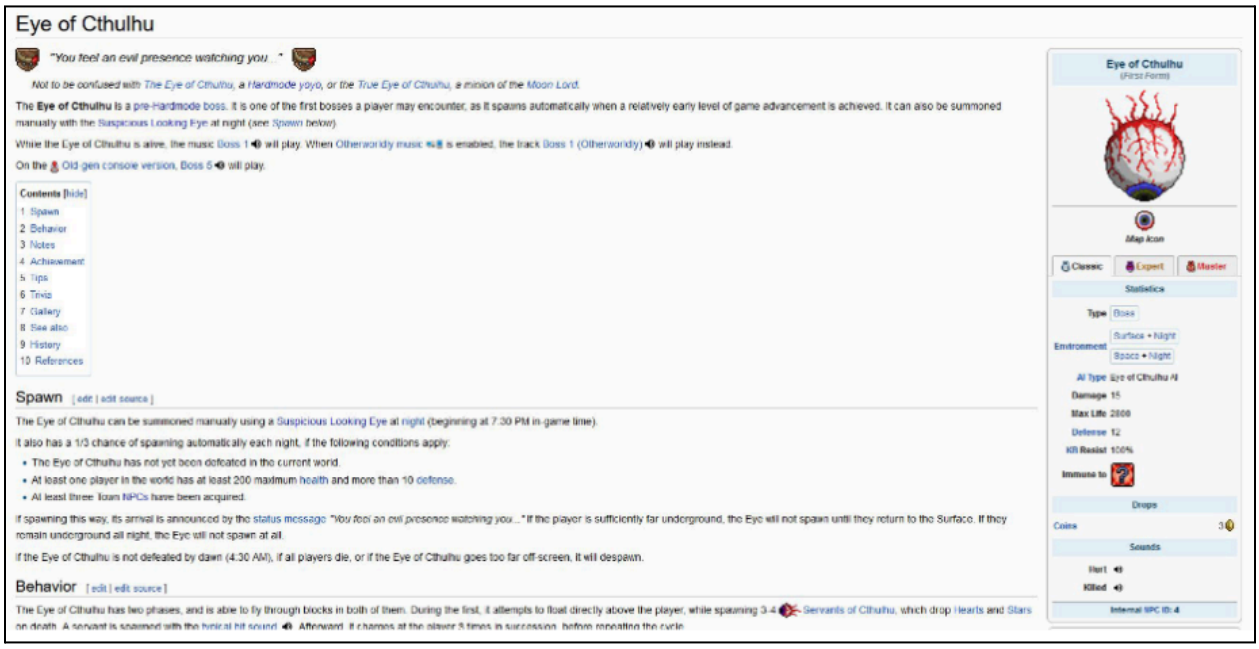
Recap of Tools and Lenses

1. Purpose, Actions, Traits
2. Trait lists (i.e. character pillars)
3. Interpersonal Circumplex
4. Character Webs
5. Transformation Charts

Note: except for purpose/actions, these WILL NOT apply to MANY games.

Wikis

- As game designers, we write a lot of documents.
- As you design and develop, expect to be maintaining spreadsheets, wikis, notebooks, project boards, etc…
- Find out which tools work for you and incorporate them into your documents!
- Think of a video game wiki, which includes all the information you need about a character.

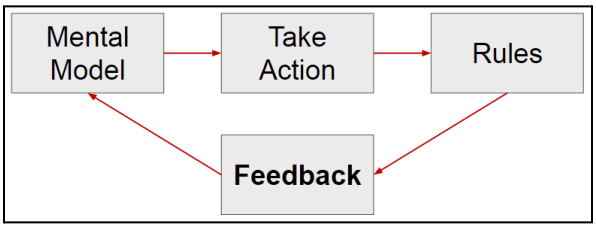


Applying Tools

- It's up to you to ask questions, write the questions down, and write down your answers.
- When it's time to answer the next question, you will have more "facts" to build off of.
- Consider relationships, stories, dynamics, actions, economies, interest factors, aesthetics, pillars.

26 - Level design: Arcs in Games

Review: Game loops



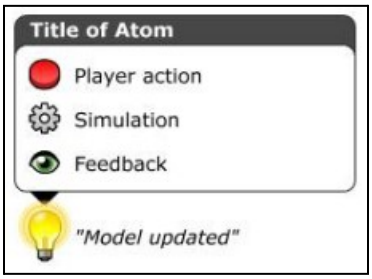
Mental Model	Goal formation Affordances Knowledge of interactions
Take Action	Physical inputs Intention, asking/answering a question Testing a skill
Rules	Game responds State is updated
Feedback	The player learns if they did well or not The game lets the player know the response

Game Loops as "Skill Atoms"

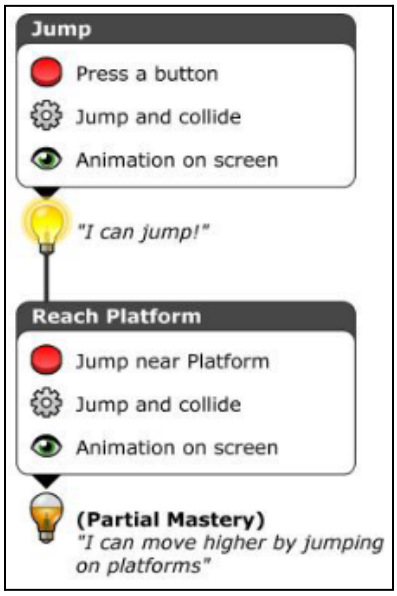
- Proposed by designer Dan Cook.
- Consider loops as practice toward mastering a mechanic.
    - Break game loop into learning one skill.
    - Feedback in the form of animation on screen.
- A "skill chain" can form where new skills require mastery of old ones.

Resource: https://www.gamedeveloper.com/design/the-chemistry-of-game-design#close-modal

General skill

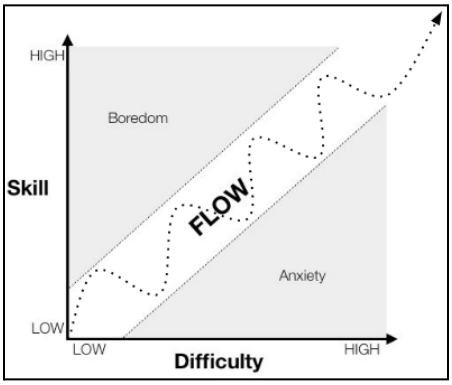


Skill chain



Recall: Flow

- The player is “in the zone” when the game isn’t too tough or too easy.
- We don’t want the player to be anxious or bored.



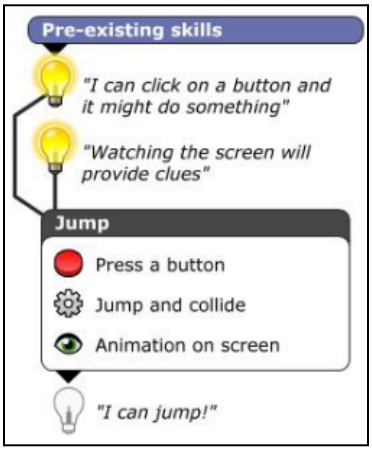
Skills, Loops, and Flow

- Throughout our game, we can keep track of which skills are being tested and where we think that player is.
- We can look out for these statuses during playtests.

- The moment of **mastery** or **partial mastery** feels great, once.
    - After this moment, we can assume the skill is **active** and available to the player.
- If we realize a skill hasn’t been used, it’s **unexercised** and should be used without practice.
- If a skill doesn’t lead to the results the player wants, it might **burnout**, leading them to avoid that action for future tests.

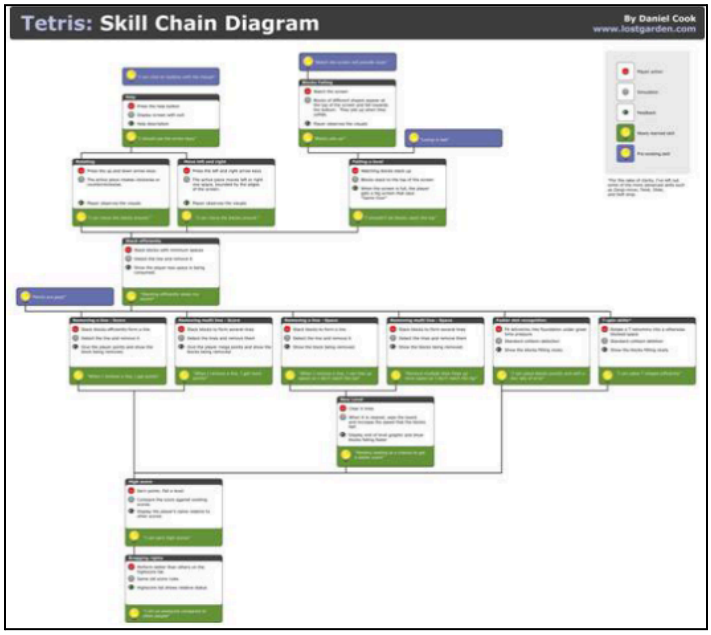
Skill Atoms and Affordances

- We can consider what the player probably has in their mind coming into a skill atom.



Planning Games with Skill Chains

- We can plan our intended skill chains and then compare them to what our game is actually doing.
- Skill chains can branch out and have many requirements.



Full size image here: https://lostgardenhome.files.wordpress.com/2021/03/tetris-skill-chain.pdf

Skill Chains and Puzzles

- Skill chains are useful for the puzzle genre.
- Puzzles are based on intuition and progression of learning arbitrary mental skills.

Ex: The Witness
Spoilers for the opening levels of The Witness.
Playthrough: https://www.youtube.com/watch?v=49mseQwFeO0

Mental model: player knows that they need to complete the line

Skills developed:
> Drag line in one direction (feedback signifying success: door opens)
> Change directions while dragging line
> Drag line to end of puzzle (visual and audio cues to signify end)
> Follow wire to find next puzzle (visual cue, wire lights up)
> Activate multiple wires connected to a puzzle by solving it two different ways
> …

This is an example of partial skill progression since players can only move on to more difficult puzzles once they master the previous skill.

This is how self-paced learning works, like Khan Academy. Games are a form of self-paced learning. Players need to UNCOVER mechanics rather than simply be TOLD mechanics.

What is Level Design?

Level design is applying everything we know about game design, but in the form of arcs.

- Ex: Chess, how do we lay out pieces on board to encourage a certain dynamic?

An arc functions similar to a *loop*:

1. The player forms a goal at the beginning.
2. The player takes actions repeatedly in the middle.
3. The player gets a big feedback dump at the end.

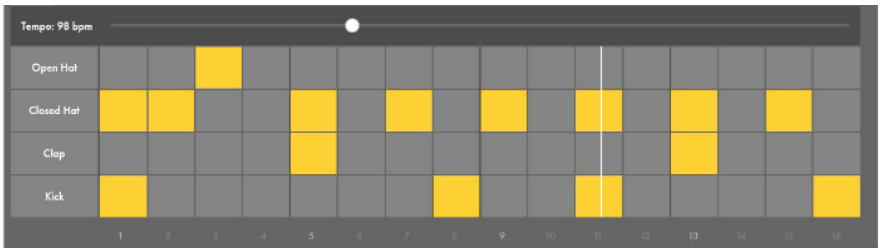
Arcs in Other Media

- Arcs are what we're most comfortable with in other media.
- Stories, movies, and songs all have a beginning, middle, and end.
- We take inspiration from other media when developing arcs because they have been so successful at keeping audiences hooked.

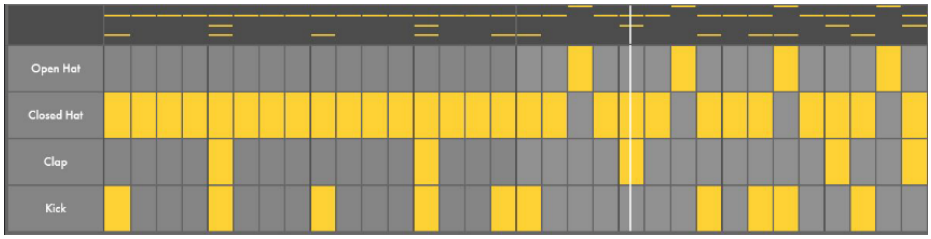
Arcs in Music

We can learn a lot about level design by considering some basic music theory.
Interactive resource: https://learningmusic.ableton.com/

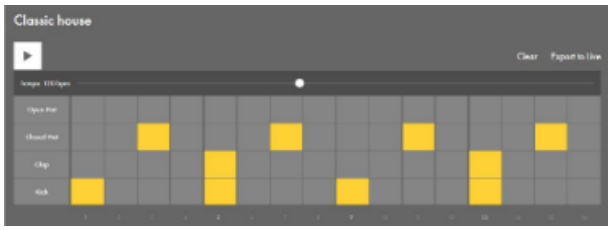
- We design beats by placing notes on a looping track.
- Each note represents an instrument.
- Each note and instrument provides some quality.



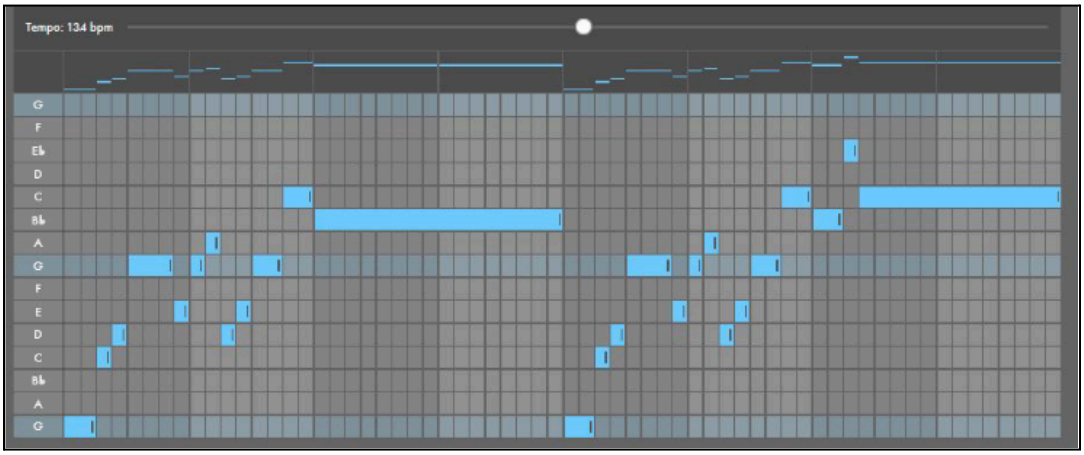
- We develop bars, repeating patterns of beats, like this two bar pattern, developing the rhythm of our song



- If we keep the bars the same but change instruments and tempo it feels completely different (different quality)



- Over time, you learn how individual notes in an instrument sound good together.
- You follow patterns for how notes progress.



- Eventually, you learn that songs follow structural patterns.
- We take smaller patterns and lay them out in repeating combinations of bars.
- The form - the particular ordering - can be common across many songs.

- We identify the structure, we consider the “ingredients” needed, and we define each ingredient as needed.
- Then, we add “flavour” to each one, and make sure it all works together.

Arcs in Sitcoms

Noah Charney has an excellent review of how arcs are patterned in sitcoms, at which time in minutes:

Article: Cracking the Sitcom Code - The Atlantic

1. **The teaser (1-3)**: A single joke, setup, introduce the protagonist and potentially the main obstacle of the episode
2. **The trouble (3-8)**: A new problem comes to the attention (Story A). They form a plan. Story B is introduced featuring a side character, which might link to Story A.
3. **The muddle (8-13)**: The plan doesn’t work, there’s another obstacle, faced in their own style that reflects the character.
4. **Triumph/Failure (13-18)**: Desperation, last resort.
5. **The Kicker (19-21)**: An “outro” showing the aftermath, possibly setting up the next episode.

Looked at another way…

1. The teaser (1-3): The Goal
2. The trouble (3-8): The Obstacle, Planning to Overcome the Obstacle
3. The muddle (8-13): Trying the plan, it goes wrong
4. Triumph/Failure (13-18): Improvisation that reflects the character
5. The Kicker (19-21): Resolution, fun and prepping for the future

Note: not to be used as a template, but often used to set up the flow/pacing.

Arcs

- We can plan arcs in advance without knowing every detail.
- The plan is the experience we're trying to achieve. Consider flow and fiero.

If music, TV, books, movies, are all based on formulaic patterns, why do we like them?

1. We like *mostly* predicting patterns.

- We're good at learning and anticipating what's next.

Tika tee, tika tee, tika tee, ___ ___?

2. We like being *surprised*.

Tika tee, tika tee, tika tee, tikatikatika

3. We like patterns at different scales.

A. Tika tee, tika tee, tika tee, tika tee
B. Tika tee, tikatika, tikatika, tika tee

A. Tika tee, tika tee, tika tee, tika tee
B. Tika tee, tikatika, tikatika, tika tee

C. Don-chi, …

Lecture 22
Apr 4 2023

26 - Level design: Arcs in Games

Arcs in Game Design

We've seen that almost all media is broken into some form of ingredients, notes, beats, and structures.

We can treat each "**level arc**" as a way to teach something new to the player.

1. Setup: easy challenges that establish what the level's about
2. Hook: show things getting *real*; suck them into the level
3. Development: increasing challenge and training the core skill
4. The Turn: the high point, the greatest challenge
5. Resolution: reflecting on your accomplishment

We typically expect some form of **reward** from arcs:

- Narrative, sensation
- New powers or skills
- New places and progression

Learning Curve

- A level helps master a skill chain.
- The "learning curve" describes how the important chains are taught to the player.
- A steep learning curve refers to a game which requires *lots* of skills to be mastered or partially mastered before progressing past early levels.

Note: while we're talking about levels, we really mean arcs.
Arcs occur at all levels of our game.

- A **room** could be an arc.
- A **level** could be an arc.
- A **checkpoint** could be an arc.
- The **game** could be an arc.

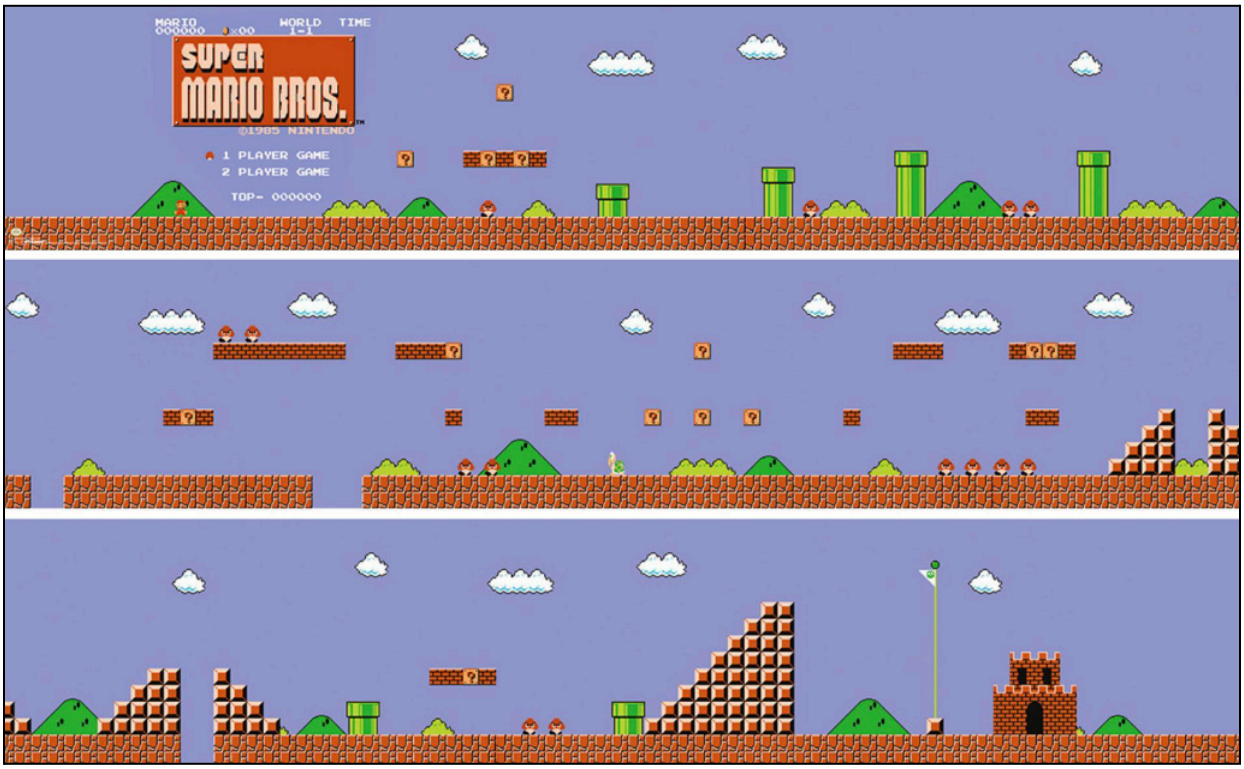
Everything we say about levels applies to all levels, from levels that are 5 seconds long to levels that are months long.

Purpose of a Level Arc

- The "experience" goal of a level is to grow the player's mastery.
- We identify what ingredients will help master those skills, and plan a progression.
- We plan small arcs and loops to get the player to our desired experience.
- Our desired experience tends to be fun and fiero.

We ask all of the same questions about arcs as we do **loops** - the knowledge is transferable, except that we also have lessons from other media.

Ex: Consider the various skills and patterns in this level.



Skills:	Progression of jumping
- Moving left and right - Jumping	- Jumping to hit a Goomba - Jumping to hit a question mark block - Jumping onto platform above - Jumping over pipe - Jumping over gap - Big jump to land on flagpole

Key Takeaway

- Levels have an experience goal.
- Experience goals are experienced through patterns.
- Patterns exist to teach and explore skill atoms.

How do we avoid burning out our player?
We can make a loop fun, but what about an arc?

Pacing in Games

Imagine this…

You've started your game and gotten playtesters but:

- Nobody bothers to go past level 1.
- Everybody quits halfway through.
- Nobody cared about your big finale.
- Your players just want to take a break.

We CAN examine this from a **flow** perspective.

- Was the level too hard?
- Was the player trained on all the right skills?
- Has the player gotten time to practice?

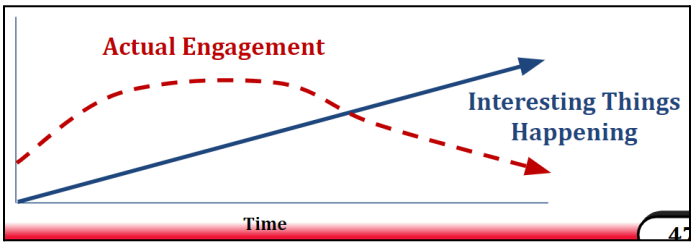
But… sometimes, it's not just the difficulty that's posing a problem.

Maintaining Interest

- Maintaining consumer interest is the main goal of entertainment.
- If we continuously throw bigger and bigger stimuli at the consumers, they become habituated to it and it loses its impact.

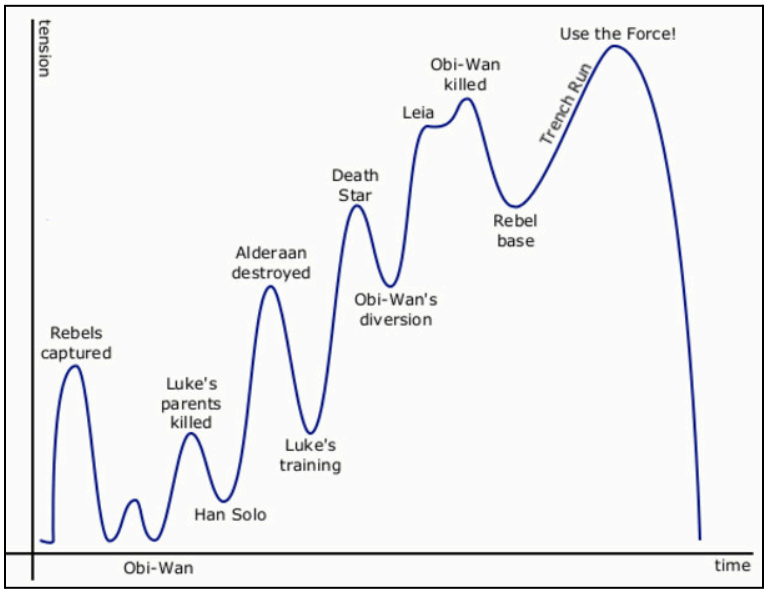
Thus, we consider a few things:

1. What makes something interesting and engaging?
2. How do we catch and hold onto interest?
3. How do we leave the consumers feeling excited?



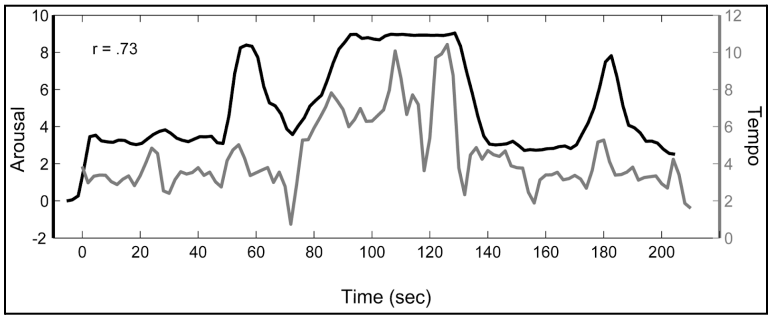
Ex: Let's take a look at the "classic" cinematic interest curve… **Star Wars**.

1. **Hook the viewer!**
2. Relax expectations, let the situation sink in.
3. **Oh no! Big event!**
4. Side excursion…
5. **BIGGER EVENT!**
6. Take a breather…
7. **HUGE EVENT**
8. Ahh, calm-
9. **AHHHHHH**
10. **AHHHHHHHHH**
11. **phew**



Ex: **Neural Arousal in Music Listening**

1. An intro to the song
2. **Big hook**
3. Calm down
4. **Longer, bigger, more exciting!**
5. Calm for a bit
6. **Big finish**
7. End higher than we started



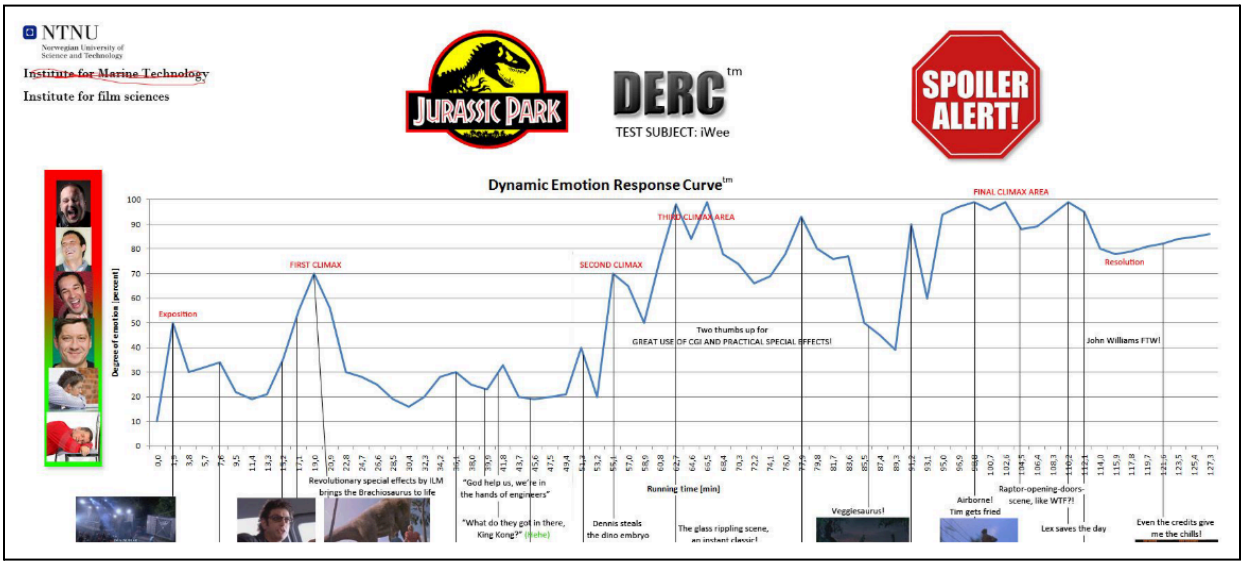
Song Structure

- Song structure and interest is usually built specifically to engage in familiarity, novelty, and intensity.
- This is why most pop songs follow identical structures.

Link: Song Structure Basics for Music Producers + Pop Song Form Examples Infographic

Neural Responses

Neural activity seems to reflect certain findings and have started to be used in playtesting.



From Chapter 16 of Schell's:

A. The interest starts BEFORE the game.
- Setting expectations.

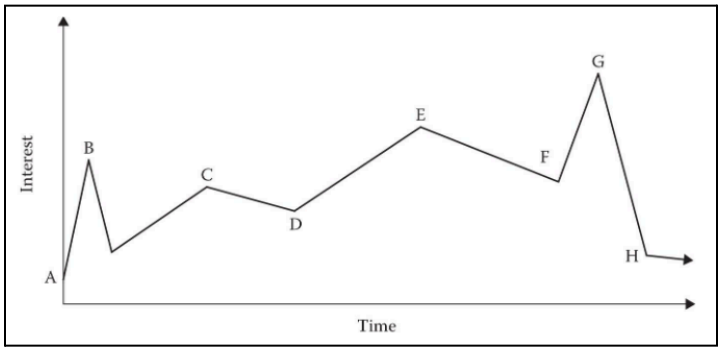
B. The hook.
- Why should they keep playing your game?

C. Gradually rising and falling interest.

…

G. Sharp increase to the climax!

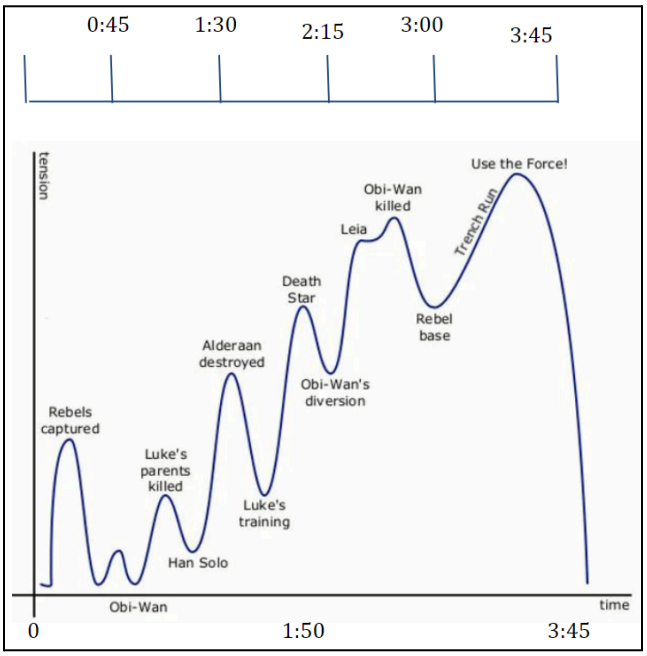
H. End more interested than when they began.



What exactly makes something interesting, though?
Can we reasonably identify when things are interesting?

Ex: **Maintaining Interest**

[STAR WARS™: The Old Republic™ - 4K ULTRA HD - 'Deceived' Cinematic Trailer]



Factors of Interest

Inherent Interest - It just… *is*.

- Some things are just more interesting than others.
- Anything appealing to “base instincts”.
    - Novelty vs. sameness
    - Difficult vs. easy
    - Risk vs. safety

Poetry of Presentation - The beauty of it

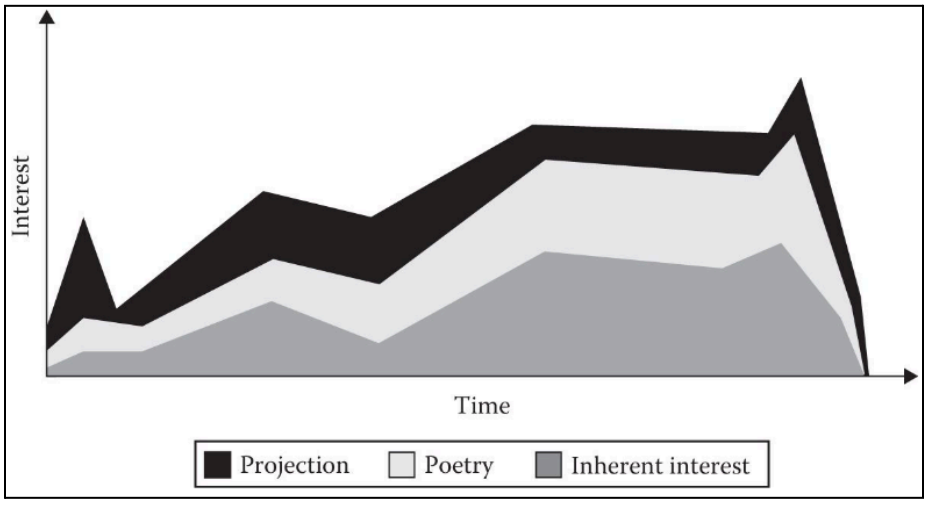
- We enjoy artistry.
- We may have some kind of beauty in the game: views, music, story.
- A poetic presentation is more interesting than something purely functional.

Projection - I vs We vs Them

- Things happening to a stranger are less interesting than things happening to you.
- Things happening to friends are more interesting, but still not as interesting as things happening to you.
- When the player’s in full control, they’re projecting.

Interest Lenses (from Schell’s Chapter 16)

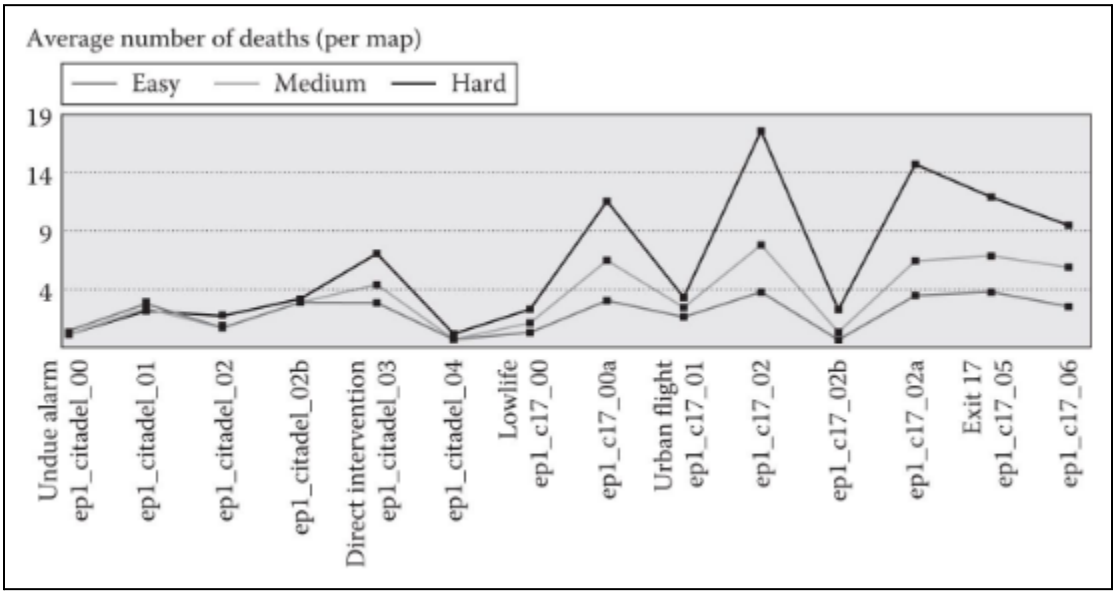
# 70: Inherent Interest	# 71: Beauty	# 72: Projection
• What aspects capture interest immediately? • Can the player do something they haven’t before? • What base instincts does my game appeal to?	• Can I make each element of my game more beautiful? • What does beauty mean within my game? • How can elements be composed together to make it beautiful?	• What can my players relate to? • What captures imagination? • Where have they always wanted to see? • Can they be who they imagine themselves to be? • Do they get to do things they want to IRL?



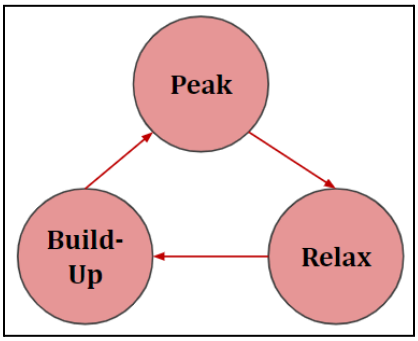
Metrics mix and match over time.

Inventing Metrics
It's up to you to invent real, numerical interest factors.

This is a graph of **Half-Life 2**'s average number of deaths per map.



In **Left 4 Dead**, Valve tracks a number of factors about the experience as "Stress Levels". This is fully exposed to PC players with debug options.



Screenshot from [Director AI for Balancing In-Game Experiences | AI 101].

The director is maintaining an interest curve as a **state machine** that attempts to generate the peaks and valleys of the interest curve in order to keep the player engaged.

What metrics does Valve look at?

- How many zombies have been killed recently?
- How far away are the zombies getting killed?
- How far away are the nearest allies?

Activity
Let's brainstorm some interest metrics for different games!

1. Role-Playing Game (Mass Effect, Fallout, Skyrim)
2. First-Person Shooter (Call of Duty, Doom)
3. Farm Simulator (Stardew Valley, My Time at Portia)
4. Puzzle-Platformer (3D Mario, A Hat in Time)

Pacing in Practice

Practical Pacing Curves

- As we've seen, some companies attempt to automate the calculation of interest on each level.
- Some approximate it based on level elements.
- Naturally, in games, many metrics tend to revolve around flow and fiero.
    - What skills are they testing?
    - When was the last time they tested that skill?
    - How difficult is the skill test?
    - How novel is the skill test?

You can actually use simple interest curves right now!

- Track your interest metrics in a spreadsheet with a count such as:
    - New mechanics learned
    - Mechanics used
    - Average number of deaths
    - Number of jumps
    - Amount of backtracking (negative interest)
- Sum them and chart for each level to approximate.
- Refine your metric over time, based on playtesting.
- Realistically, you'll begin to internalize the pacing curve and aim for it as you work without needing to consider the metrics *as* much.

Arcs are Fractal and so is Pacing

We get enjoyment from this pacing pattern at every level of our experience.

Fractal Curves - the deeper we look, the more curves we see
We can see the pacing curve in all elements, at differing scales:

1. **Game scale**: intro, rising interest level-by-level, end with a big finale
2. **Level/scene scale**: new challenges at the start, testing in more interesting ways until a final boss battle
3. **Action/challenge scale**: interesting intro, anticipation, release

Revisiting **Mario's 4-Step Design** as **Interest Fractals**
Video: Super Mario 3D World's 4 Step Level Design
Transcript: Super Mario 3D World's 4 Step Level Design

Game Scale: this is obvious

Level Scale

1. Risk-free introduction to new mechanics.
    - Red-blue switches that flip when you jump
    - Player is introduced to this mechanic over a platform so they can't fall into the void
2. Mechanics established further with no safety net, sometimes revisiting old mechanics.
    - Player navigates switches over the void
3. Twist the mechanic to challenge the player.
    - Player scales a cliff face using switches
    - Player jumps over rings of fire while navigating the switches
4. Final change to show off skills.
    - Using the new mechanic to reach the flagpole (and, in other levels, beat a boss)

Challenge Scale: kill enemy, navigate a couple panels, choice to get star, backtrack, reach ending

Action Scale:

- Running
    - A little slide and a puff of smoke
    - Mario's head throws back and pushes forward quickly then slows
    - End with a slide and a sound effect
- Jumping
    - Head down, punch the air, rise up really quickly
    - Slow the fall, room to choose where to land
    - Choose to ground pound or begin a faster fall
    - End on a big smoke cloud

Pacing and Arc Design

- Level design is just like designing the rest of your game, except that we're designing arcs instead of loops.
- Our arc has a clear goal, spends time exploring ways to achieve the goal, and makes the player feel good for achieving the goal.
- To keep attention during an arc, we follow a cinematic pacing curve, hooking the player and raising interest until an awesome finish.

This video demonstrates reducing levels down to their atoms, categorizing levels as patterns, and identifying sub-arcs: The Legend of Zelda: The Minish Cap's dungeon design | Boss Keys

Lecture 23
Apr 6 2023

27-Level Design: Practical & Rational Game Design

How can we begin to plan the difficulty curve of our game?
How can we produce plans for levels when we don't fully know the specific details?
How can we learn to see patterns in our skill atoms?

Question to consider: can you recall any examples of intentional pacing in video games?
Ex: Breath of the Wild's weapon durability system

- Your weapon will break in the middle of combat
- Makes battles more engaging
- As you get stronger the weapons break over multiple combats
- Avoids consumables problem where you never use other weapons

Level Design in Practice - The artifacts of a level

Process of Level Design

Level design is a test of all of your design skills.

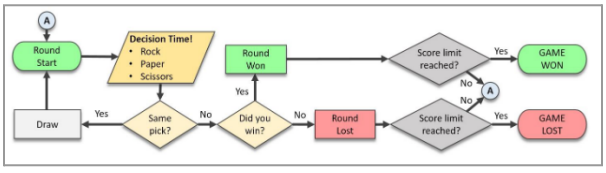
- A level motivates the mechanics.
    - It makes the mechanics make sense and contextualizes them.
- A level gives the player the feelings that we want them to feel.
- A level keeps the player engaged and coming back by giving them goals.

Levels define the path of your player experience. So, we need to plan our levels ahead of time.
… but, we don't know if something is fun until we try it.
So how do we plan a level?

The Two "Brains" of Level Design

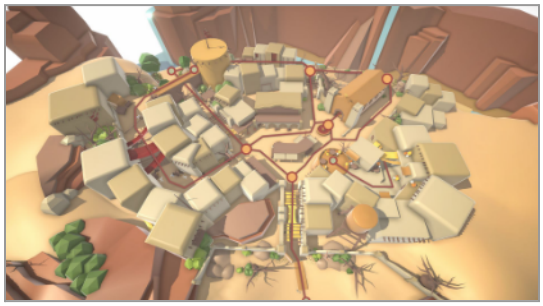
Functional Requirements

- What we need to accomplish
- What needs to get done in this level in terms of the game arc (major beats)
- What happens technically



Source: Practical Game Design, Kramarzewski, De Nucci

Aesthetic Requirements



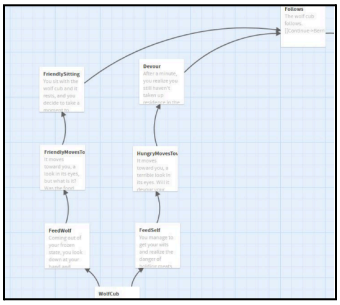
Source: https://80.lv/articles/introduction-to-level-design-for-games/

Both sides contribute to the same goal: player experience. Each side needs care.

First, we consider a **high-level plan** of our game:

1. Consider the overall progression (less flexible)
    - How are the story details being distributed between levels?
    - When will the player get access to new mechanics? What is our skill chain?
    - What major moments will occur in the game? When?
2. Consider the aesthetic themes (more flexible)
    - What feelings do we want to evoke in the player as they progress?
        - "Player" transformation charts
        - "We want the player to feel overwhelmed" vs. "We want the player to feel complete control"
    - What themes and imagery should we use as we progress?

Tools for Story Scripting → Twine (https://twinery.org/), Yarn Spinner (https://yarnspinner.dev/)



- Visual way to do minor scripting
- See story progression through flow chart
- Can use to make standard text adventure
- A great way to communicate your design to the rest of the dev team
- Use to get a high-level idea of the overall flow of the story and major experience beats.

Between skill chains and experience beats, we can start to consider the functional requirements of our levels.

Tools for Planning Functional Requirements → Notion, Google Docs/Slides

- Jot points
- Start to consider the functional requirements of our levels
- Notice patterns and conventions being established in the game → develop arc
- Continue breaking things down until you're happy with the level as a well-defined story

Individual Nodes

- Starts with short, ~10-20 second (1-2 turn encounters) which are intended to be quick, exciting, and hit at the player's overall resources
- Stronger encounter, which takes some strategy and can severely reduce resources; ~2 minutes (~10-15 turns)
- More encounters which ramp up in difficulty, ~ 3-6 turns each
- Final boss/group encounter, ~20-30 turns
- Average **10-15 minute** encounter w/ miniboss + boss (can also just be more difficult encounters)
- **Variants:**
    - Hoard of trash node (toss in occasional thoughtful)
    - Hoard of trash -> Miniboss
    - Trash->Miniboss->Thoughtful->Boss
    - The Big Boss: Boss -> Boss 2nd form -> Boss 3rd form

Individual Encounters

- Trash: 10 seconds
- Thoughtful Trash: 30-60 seconds
- Miniboss: 120-180 seconds
- Boss: 180-240 seconds

Act tl;dr

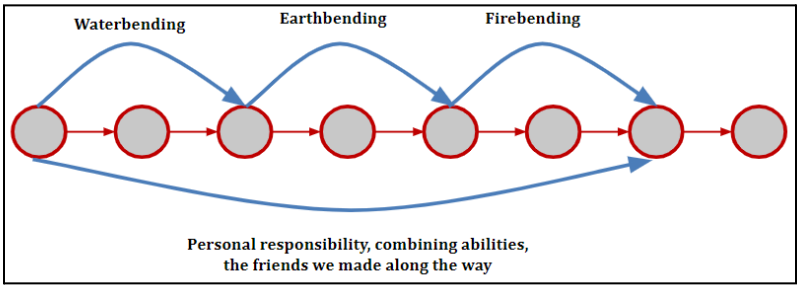
- **Prologue:** Introduce the core mechanics
- **Act 1:** Get the player familiar with the mechanics, introduce minor subversions, at full baseline power, prepared for Act 2. Primarily monster variants with a few zone effects. Primarily linear with short branches for unlocks to provide a full set of basic relics and characters.
- **Act 2:** Playground for the player to experience unique battlezones and unlock lots of things which customize their roster. Unique, dynamic changing relics introduced. Uniquely operating classes introduced. Lots of biome variation. Longest act.
- **Act 3:** Combined subversions and race to the finale
- **Epilogue:** Replay old sections with a variant (powerful relic)
    - Questionable

Prologue (30 mins)

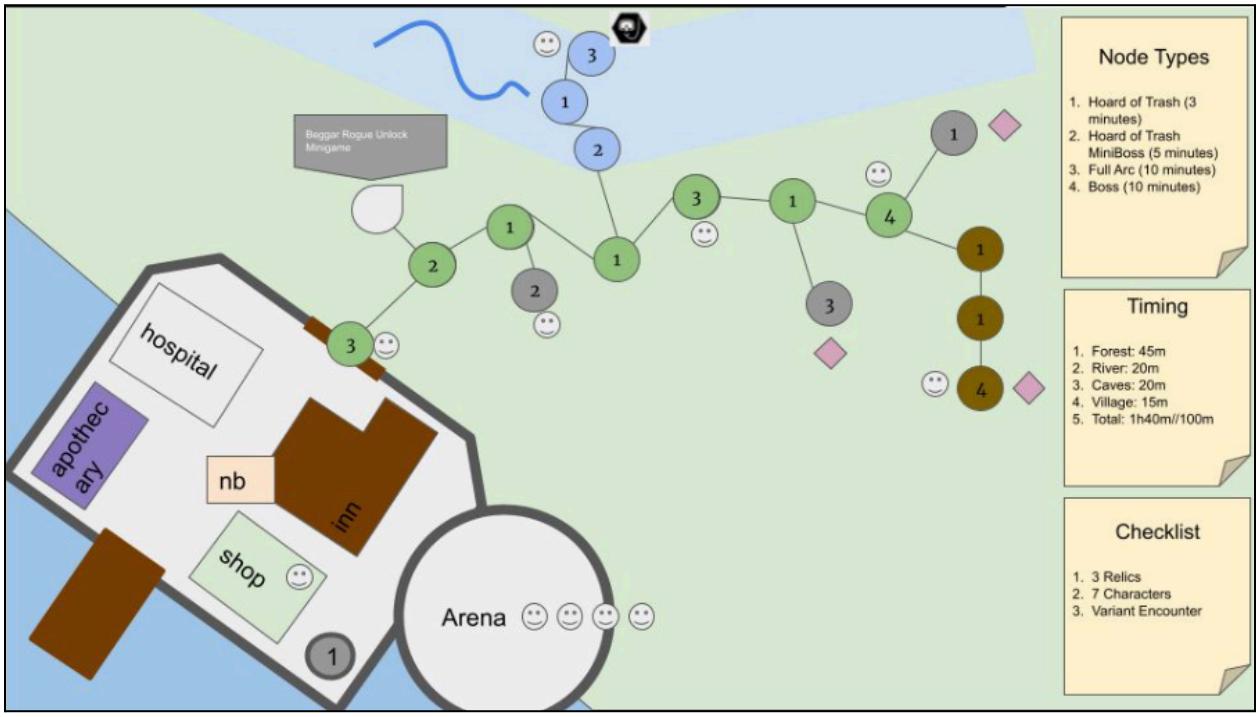
- High-Level Intentions
    - Player learns about the core systems of the game
    - Player learns about the overall theming
    - Player is motivated to start their quest and venture out
- Required Beats
    - Achieve 3 aspirations
    - Unlock base 3 character cards
    - Master basic, non-elemental combat
- Town
    - Places:
        - Keep
        - Front wall?
        - Tavern
        - Shop
        - Hospital
        - Apothecary (for cosmetics)
        - Arena
    - Encounters
        - Tavern
    - Aesthetic:
        - Ocean-side castle town
        - Distinct scene with zones to click into
        - Zoom into and reveal nodes and idle animations based on zoom level
            - Apply a hiding fog or something to fuzz the view
            - Single viewpoint locked camera views

Loops within Arcs within Arcs

Like loops, arcs are at many scales. It's helpful to go back and forth between the different arc scales to consider requirements.



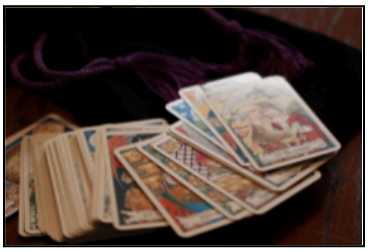
Sketch!



- Lay out loops using a sketch.
    - Where are rewards distributed?
    - How long is a session?
    - How long is each type of level/encounter expected to be?

- Consider your aesthetic requirements.
    - What perspective should we have?
    - What is our game/level's palette?
    - How do we convey key information to the player?





- How should the game feel aesthetically? (mood board!!)
    - Colours, themes, geometric inspirations
    - Game references, movies
    - Helps the team understand the feeling
    - Can use chat gpt to generate a description and put those descriptions into an image generator to get inspiration images that generate the right emotion

Big Arcs to Small Arcs

That's how we plan the entire game. How do we plan a level?

Levels and games are both arcs, so use the same tools.

Process

1. Concept
    - What is it about?
    - What's required?
    - What's the hook?
2. Sketch
    - Clearer idea
    - Understanding the scope of the level
    - Developing the technical requirements
    - What kinds of programming/art tasks will be needed for this level?
3. Greyboxing
    - Feeling the level in the game engine
    - Iterative game design
4. Art pass
    - Moving from gray boxes to art
5. Polish
    - Debug
    - Final adjustments

Concept examples from Practical Game Design (2018) (available at the library)

Left 4 Dead (Multiplayer FPS) – No Mercy Campaign

The first campaign in the game is set in New York City, shortly after the viral outbreak. Our survivors are awaiting rescue on the roof of a downtown apartment complex. A helicopter flies by and tells everyone to make their way to the Mercy Hospital to evacuate. The survivors embark on a 4 or 5-part adventure that takes them through the streets of New York, the metro system, and the sewers, culminating in a grand finale on the rooftop of Mercy Hospital.

Mario Kart (Super Nintendo, Racing) – Rainbow Road

Exciting, multi-colored paths in the depths of outer space! Loads of 90 degree turns and a few long straights. Short length and normal grip. Difficulty is added by removing all barriers and putting in Super Thwomps (falling blocks that knock out racers when touched).

Sketch to Concept → Understand the requirements

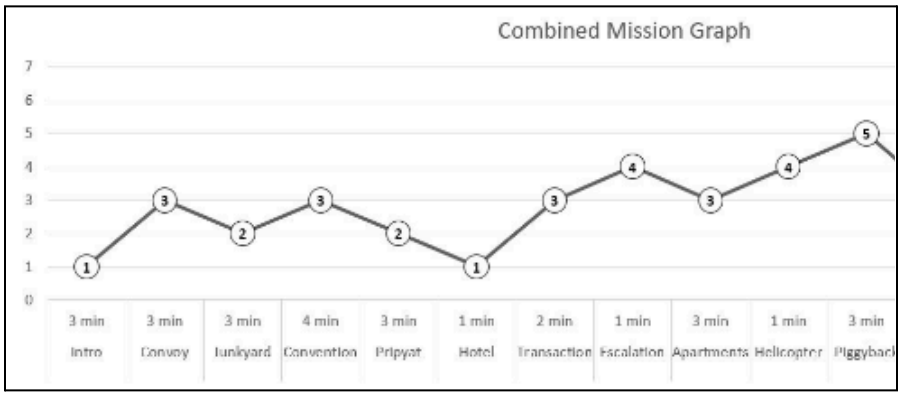
- Skills
    - Enter: the player has mastered A, B
    - Exit: the player needs to master X, Y, Z
- Goals
    - Enter: the player is motivated to find L
    - Exit: having found L, they are not trying to find Q
- Major beats
    - Meets the companion, develops a bond
    - Obtains the X item and learns how to XYZ
    - First major boss fight, huge reward
    - Feels like the quest is just beginning
- Anything else that's important to your game

Sketch → Understand the level

- Mood board, sketching, all the things talked about previously. Help everyone down the communication chain of different teams understand the level.

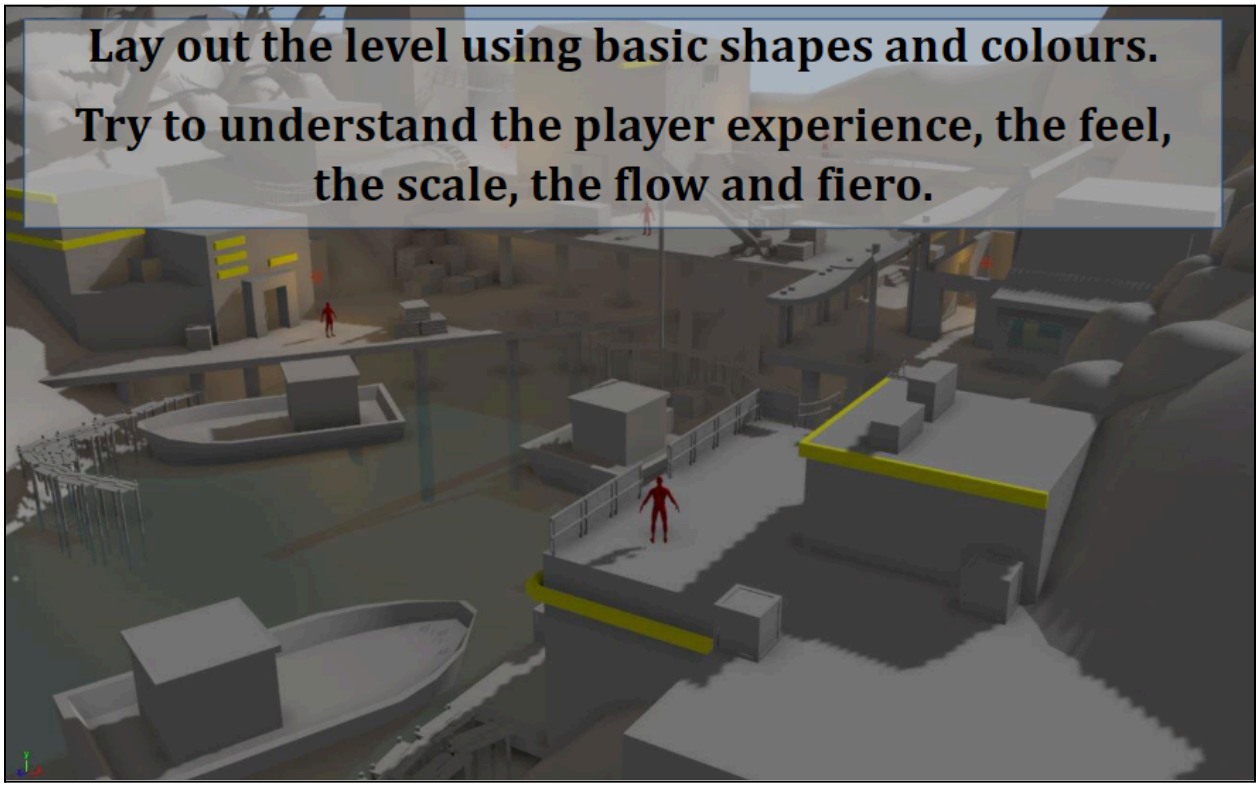
Premise: A quiet and atmospheric black-ops mission. The player needs to follow their partner, stay patient, sneak around, and time their assassinations to avoid detection. This section can be divided up as follows:

Name	Details	Duration	Intensity
Intro	Follow Captain MacMillan. Lay low and take down your enemies in a stealthy fashion. Infiltrate a nearby church.	3 min	1
Convoy	You'll need to get past an open field undetected. Enemy helicopter, armor, and infantry patrols are on the lookout.	3 min	3



Examples from Practical Game Design.

Greyboxing → Understand the experience



- See if the choices are clear. "Can I reach this ledge? Can I use this tactic?"
- "Squint test" - if we blur our vision, can we still navigate the level?
    - Emphasize sight lines and interaction points
- Why greybox? Don't rely on assets to carry your game. Might get "lost in the visuals".
- Use greyboxing to "validate" the game: "Is this worth working on?"
    - Is this feasible?
    - Is it worth the art input?
    - Did playtesters react positively?

What Comes Next?

From here, we let the artists art, the programmers program, and the designers design.

- It's possible to start these at the sketch phase, but be careful.
- Prioritize the items that are most likely to be kept.
- The design will absolutely still be tweaked and adjusted, but the more you commit to the dev/art, the more you commit to the design.

Practical tips for designing spaces

Work Out the Shape

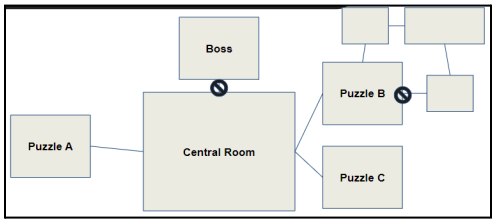
We can start with a general shape that gets the pacing across. "If we start at this point (open area), then we can move here (smaller area) where there's more of a claustrophobic feel…"

Map generated by https://donjon.bin.sh/fantasy/dungeon/

Consider Landmarks

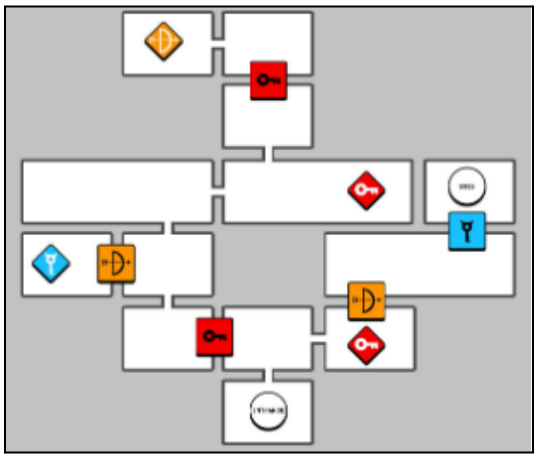
- Guides people to move towards them
- Orients people; "You are here right now, in relation to here, here, and here"
- Ex: Disney World makes use of "weenies", landmarks which guide and orient guests in the park.

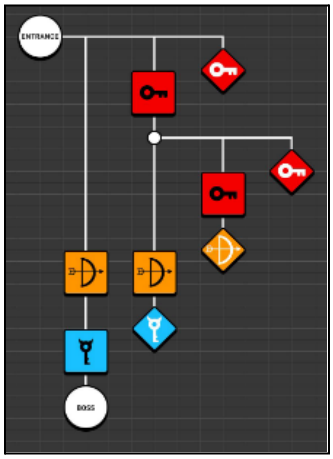
- Can see this in **"hub and spoke" model** in games →
  - Usually, one central room acts as the landmark



Meaningful Choice and Progression

- Consider lock & key gating
  - Simplify your understanding of the progression and consider the meaningful choice in your design.
  - "Need a red key to progress, need two keys to get through here…"
  - Do remember - locks aren't always opened with keys.

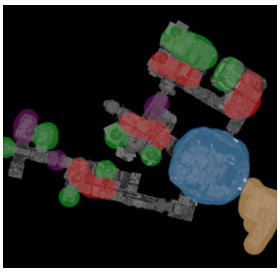




Pictures from https://www.patreon.com/posts/how-my-boss-key-13801754

Pacing

- Establish Nodes/Locals/Districts/Clusters
    - Identify the purpose of each section to get an understanding of pacing.
    - Consider how they connect and how they provide context to the player for the gameplay.



Loot: cluttered, safe, loot props
Enemies: big encounters with trash
Encounter: boss/narrative beats
Hub: verticality, visually stunning, safe

Summary

1. Shape level
2. Major levels
3. Choice progression
4. Clustering different areas "this area is a hub, this area has a lot of enemies, this area has a lot of loot"

- Come together to get clear idea of what this looks like before we've touched any code and asked for any art
- Can get clear idea of dynamic of level without touching system… until we want to validate it

Exercise! Let's design a Zelda level.

Inspiration: A Link to the Past Walkthrough - First Dungeon

Enter skills:

- Slashing enemies with a sword
- Throw boomerang to hit enemies
- Pick up and throw pots
- Press switches to open a door
    - Find switches under pots

Technical requirements:

- Find a key item
- Use the key item to defeat a new type of enemy
- Key item grants access to new rooms
- Key item used to defeat final boss

Resource: Miro (https://webwhiteboard.com/)

Rational Game Design - Let's design a level

Let's talk about Ubisoft's method of design developed for *Rayman Origins*

- Similar concept to skill chains, but formalizes this whole process
- Used to design most Ubisoft games
- Just one approach, a very specific framework/implementation/process

https://www.gamedeveloper.com/design/rational-design-the-core-of-i-rayman-origins-i-
https://www.gamedeveloper.com/design/the-rational-design-handbook-an-intro-to-rld

Problem: no ideas

- Fall back on a process to help you get started
  - Gives you structures and blanks you can fill
- Apply constraints by knowing your requirements

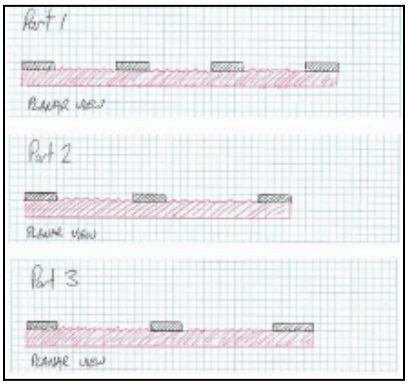
Rational Game design (RGD)

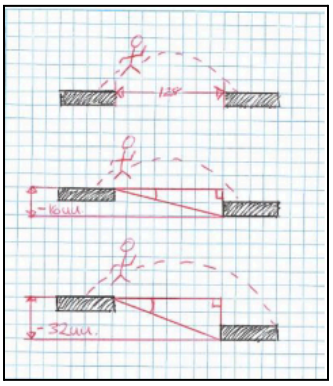
- Fixed process with a fixed order
- Helps us understand the difficulty and learning curve of our game objectively
- Data-driven and based on playtests
- Some terms are used uniquely to RGD, like *mechanic*

→ RGD Workshop : Rational Game and Level Design
→ The Rational Design Handbook: An Intro to RLD

1. What is the goal, in the clearest terms possible?
   - Ex: Get the ball from point A to point B without being intercepted.
   - Ex: Get the player to the right side of the screen without running into an enemy.
2. What skills are we asking of the player?
   - Ex: What skill is needed to jump over a small gap? …to reach platforms? …to jump on enemies?
   - Physical vs. mental vs. social skills
   - Consider the *physical inputs* for the skill.
3. Assess the difficulty.
   - Find the ways the skill can be easier/harder.
   - Playtest with first-time testers to get some values, the skill's "parameters".

	World Units	RLD Difficulty Value
Easy Jump	128	1
Medium Jump	160	2
Hard Jump	192	6





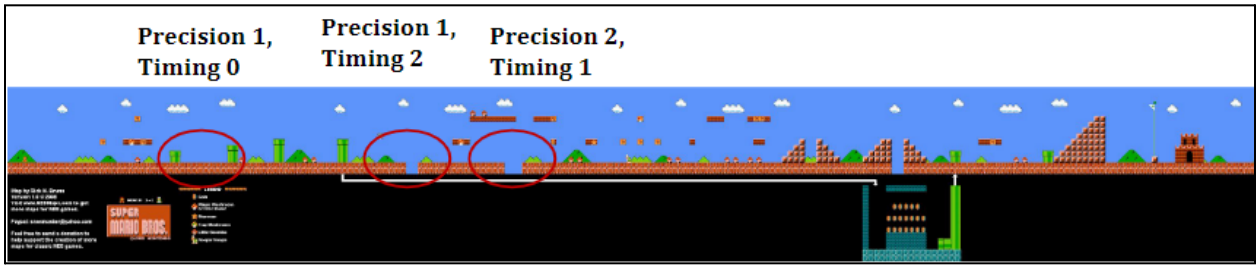
   - Ex: What are the world units? How many pixels apart to make it a difficult jump?

Quick Example of steps 1-5

- Goal: Get a ball from Point A to Point B without being intercepted.
- Required Skills: Precision, Timing, Measurement
- Expected Input (What am I physically playing on)
  - Let's say it's a mobile swipe. (Not good enough, break it down.)
    - Origin Point, End Point?
    - Speed?
    - Distance?
- Atomic Parameters:
  - WoO (Window of Opportunity)
  - Anticipation
  - Angle of Tolerance
  - Amplitude of Tolerance
- Difficulties: None, Easy, Med, Hard, Impossible

Parameter/Difficulty	None	Easy	Medium	Hard	Impossible
WoO	Any	10 sec	5 sec	2 sec	0 sec
Anticipation	None	30 sec	15 sec	<10 sec	None
Angle of Tolerance	360°	270°	180°	30°	<5°
Amplitude of Tolerance	Any	Any	Full Swipe	< Full Swipe	< Quarter of a Swipe

4. <u>Mix and match modifiers to create unique ingredients to use in our levels.</u>
   - Find new ways of presenting the same ingredients.



   - Ex: Mario level

5. <u>Ingredients become progression.</u>
   - Map our intended skill progression.
   - This lets us estimate flow and pacing.
   - These are still estimates, and may not align with playtests.
   - Actual implementations of RGD get very in-depth, very specific.
   - Check out the links above for more information!

Puzzle	Difficulty	Ingredient
1	1	Precision(1)
2	2	Precision(2)
3	4	Timing(1)
4	2	Precision(1) + Timing(1)
5	2	Timing(2)
6	3	Prevision(2) + Timing(1)

Note: you must continue testing after you change the aesthetics.

- You can vary the aesthetic of the same pattern and get drastically different results.
- Players may not realize that it's the same pattern.
- This saves us effort and time.
- More content with less resources.

Summary

- You will find your own way to design your content, but keep all of these things in mind as you go.
- Given level design is game design, consider how you can take these lessons and apply them to *other parts* of your design!
    - Pacing applies on the level scale and the game scale.
    - Levels are just arcs that each implement skill tests.

Interested in learning more?

→ GDC level design playlist (~15-30 hours) (GDC Level Design Talks)
→ Practical Game Design Book (available from the library)