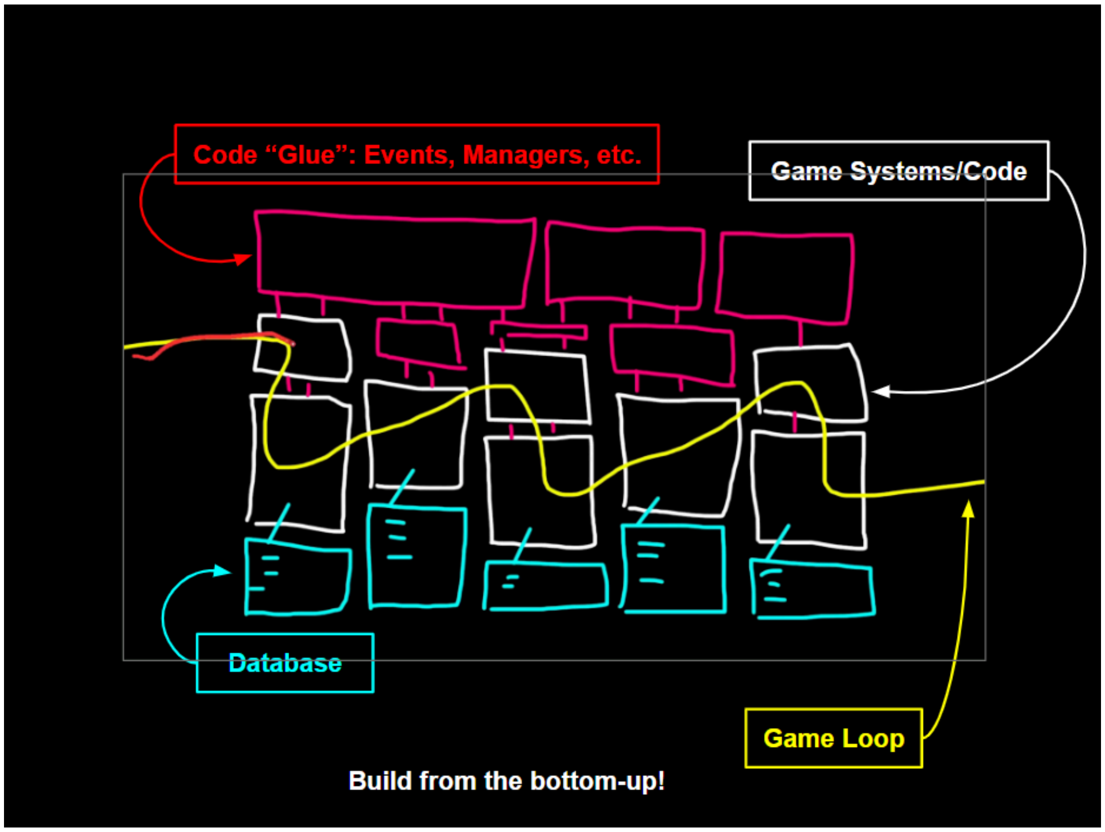
Investigation Week - Project Misfits

Table of Contents

Data-driven Codebase Design

Source: Best Code Architectures For Indie Games

- Separate your codebase into 3 distinct layers: the database, game systems, and “glue” code
    - The Database is where your immutable data should live. These should live in text files, CSVs, or other file formats which are easy to read from.
        - Examples: player/NPC parameters, dialogue, string text
        - Keep relevant data in one place; if it is *game-designed* together, store it together
        - Godot can also store data as Resources to make it easier to access in-engine.
        - Consider designing the database first, then building the game systems on top of it. This keeps your codebase data-first.
    - Game Systems should not contain any data. Instead, they should read from the Database at runtime to configure themselves and set up the game world.
        - e.g. the Player class accesses a text file with parameters for movement speed, jump height, health, etc.
    - The “Glue” Code connects the game systems and database together indirectly. “Glue” code can be events, game managers, connector scripts, or in-editor connections. It prevents data and game systems from being directly reliant on each other, thus letting the codebase stay *modular* and *scalable*.
        - However, avoid “universal glue”: for example, if you find your game manager becoming bloated with the functionality of multiple systems, you should separate the game manager into multiple smaller managers that connect with each other.
- There is one system that will always be one bit of “spaghetti code” you cannot change: time, specifically the game loop that is always calling from other game systems.
    - Manage it by tightly controlling the execution flow; make sure everything runs in the order you want it to.
- If you find two systems are difficult to keep entirely separate (e.g. health and attack), consider combining them into one shared system!


Code "Glue": Events, Managers, etc.
Game Systems/Code
Database
Game Loop
Build from the bottom-up!

Screenshot from "Best Code Architecture For Indie Games" (https://www.youtube.com/watch?v=8WqYQ1OwxJ4)

Core Game Systems

- Major Code Systems List
- Some major systems include Player, Combat, NPCs, Cutscenes, and Dialogue. Each of these have multiple smaller systems that must interact with each other. For example, the Player system will involve a movement system, health/attack system, interaction system, etc.

Shaders

Source: https://godotshaders.com/

I browsed through the website Godot Shaders to get ideas for how to best utilize shaders in our game. Godot Shaders is, as you would expect, a website to share shaders made for Godot. This makes it easy to understand how the shader was created in Godot and easily put the source code in your game.

Generally, shaders should help us generate supplemental or automated art to reduce the workload of our dedicated artists. Since our game is 2D, there are fewer opportunities to utilize shaders than in 3D games. However, I was able to find some shaders which could be applicable to our game:

- Modifying Shaders
    - These shaders modify existing art to automate animations or add additional effects
    - 2D Fire (https://godotshaders.com/shader/2d-fire-2/) creates a fire-waving effect to textures using a sine wave and noise.
    - 2D Foliage Wind Effect (https://godotshaders.com/shader/2d-foliage-wind-effect/) slightly rotates UVs around a pivot point, giving the impression of the sprite gently moving with the wind.
    - Highlight CanvasItem (https://godotshaders.com/shader/highlight-canvasitem/) gives sprites a periodic shine effect, which can be helpful to highlight interactables or collectables.
- Generation Shaders
    - These shaders generate entirely new graphics
    - Procedural Torch & Candle (https://godotshaders.com/shader/procedural-torch-candle-shader-fire-smoke-sparks/) creates a small animated flame without using textures or particles.
    - Water 2D + reflection 4.x (https://godotshaders.com/shader/water-2d-reflection-4-x/) creates a water texture which reflects what is directly above it. The water is also animated and distorts its reflection.
    - Smooth 2D Cloud (https://godotshaders.com/shader/smooth-2d-cloud/) creates a panning cloud texture using perlin and cellular noise.

Narrative Documentation

I looked into different documentation, both narrative specific and general GDDs. I found…

- Adventure Time’s TV Show pitch bible
- Hilda’s TV Show pitch bible
- General GDD Template
- GDD for a game called ‘Claw’

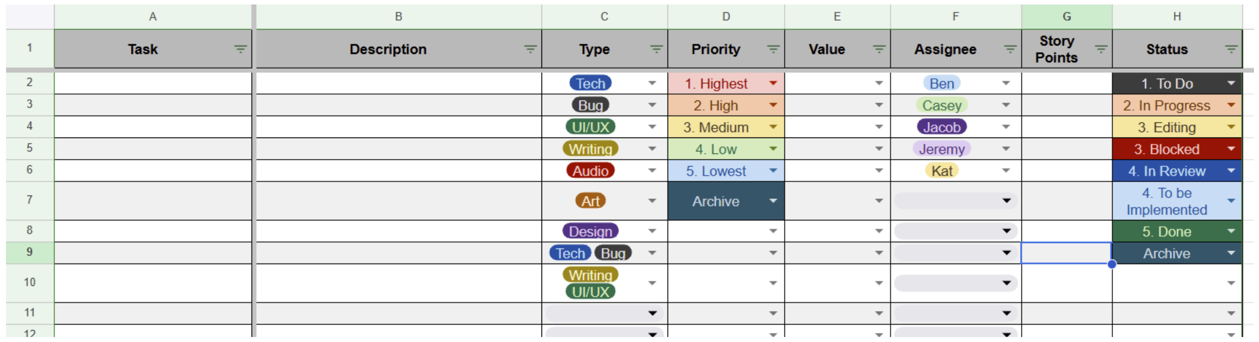
I included all of these documents in our google drive, and took notes on all of them.
Some key takeaways!

- In our GDD, we should describe our target audience as one of the first items
- Include Debug/Cheat codes in the GDD to help developers
- Spend a lot of time on fleshing out the main character, describing what makes them interesting and well-rounded.
- Spend less time on side characters. Flesh them out, and also describe them in relation to the main character (relationship? do they hold up a mirror to the MC? etc.)
- Make sure to include sections for character design philosophies, how to create a character (pipeline), creating enemies/monsters, section on Game world focusing on key features (features which should standalone, be awesome, and help sell the game)

Production Software

I looked into various production software, but basically all of them had way too much overhead and needlessly complex. I looked into Jira, Github Actions, Asana, Shortcut, and Trello.

I suggest we use Google sheets (and I made a sample). Sample Sprint Spreadsheet?



	A	B	C	D	E	F	G	H
1	Task	Description	Type	Priority	Value	Assignee	Story Points	Status
2			Tech	1. Highest		Ben		1. To Do
3			Bug	2. High		Casey		2. In Progress
4			UI/UX	3. Medium		Jacob		3. Editing
5			Writing	4. Low		Jeremy		3. Blocked
6			Audio	5. Lowest		Kat		4. In Review
7			Art	Archive				4. To be Implemented
8			Design					5. Done
9			Tech Bug					Archive
10			Writing UI/UX					
11								
12								

This table includes

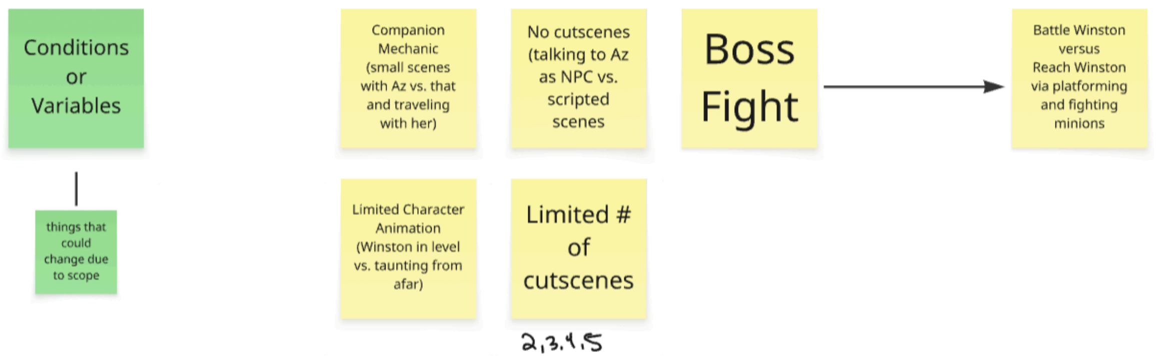
- Name of the task
- Description
- Type or discipline of the task
- Priority
- Value (how valuable the task is to the game)
- Assignee
- Story Points (the approximate 'weight' of the task. How long will this take, relatively)
- Status

The spreadsheet also includes 4 tabs (with this information copied and pasted so you can cut and paste tasks between the tabs). Tabs include

- Current Sprint
- Backlog
- Finished
- Archive

Narrative Brainstorming and Scope

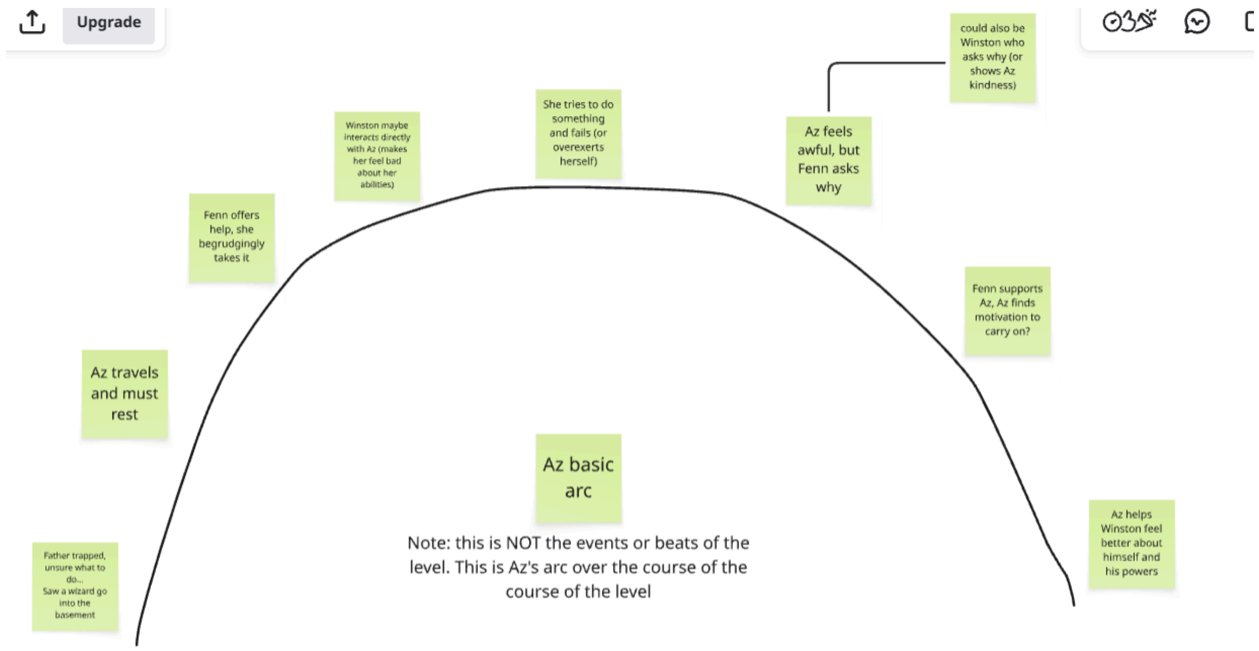
Narrative Limitations:


Conditions or Variables
things that could change due to scope
Companion Mechanic (small scenes with Az vs. that and traveling with her)
No cutscenes (talking to Az as NPC vs. scripted scenes
Boss Fight
Battle Winston versus Reach Winston via platforming and fighting minions
Limited Character Animation (Winston in level vs. taunting from afar)
Limited # of cutscenes
2,3,4,5

First, I brainstormed possible limitations for the narrative. The ones I came up with are as follows

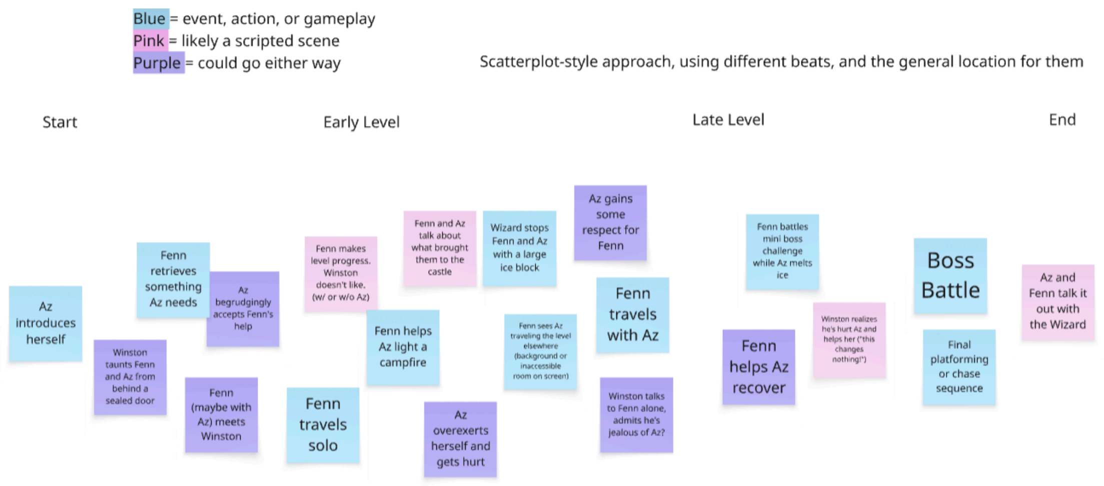
- Companion Mechanic
  - Whether or not we could successfully allow our companion character (Az) to travel with the player
- Limited # of cutscenes
  - Depending on time, we'd need to adjust beats and scenes if we have a set number of cutscenes
- No cutscenes
  - If we are restricted to only dialogue when interacting directly with NPCs, as opposed to scripted scenes
- Boss fight
  - Whether we have time to program and design a boss fight *or* if we have to use existing enemies and platforming to build a climactic ending
- Limited Character Animation
  - Which could affect cutscenes and how frequently we can use our NPC characters

Character Arc Brainstorm for Az


Upgrade
Father trapped, unsure what to do... Saw a wizard go into the basement
Az travels and must rest
Fenn offers help, she begrudgingly takes it
Winston maybe interacts directly with Az (makes her feel bad about her abilities)
She tries to do something and fails (or overexerts herself)
Az feels awful, but Fenn asks why
could also be Winston who asks why (or shows Az kindness)
Fenn supports Az, Az finds motivation to carry on?
Az helps Winston feel better about himself and his powers
Az basic arc
Note: this is NOT the events or beats of the level. This is Az's arc over the course of the course of the level

Next, I brainstormed a basic outline for a character arc for Az. In a theoretical larger game, this level would be one beat or point along the main character's journey, but Az's character arc would be contained within this level.
This narrative outline is intentionally generic, so we could fit many possible beats or situations to the arc. For example, "Az tries to do something, but over-exerts herself/hurts herself" is generic enough where we could adjust it based on technical limitations.

Narrative “Scatterplot”


Blue = event, action, or gameplay
Pink = likely a scripted scene
Purple = could go either way
Scatterplot-style approach, using different beats, and the general location for them
Start
Early Level
Late Level
End
Az introduces herself
Fenn retrieves something Az needs
Winston taunts Fenn and Az from behind a sealed door
Az begrudgingly accepts Fenn's help
Fenn (maybe with Az) meets Winston
Fenn makes level progress. Winston doesn't like. (w/ or w/o Az)
Fenn travels solo
Fenn and Az talk about what brought them to the castle
Fenn helps Az light a campfire
Az overexerts herself and gets hurt
Wizard stops Fenn and Az with a large ice block
Fenn sees Az traveling the level elsewhere (background or inaccessible room on screen)
Az gains some respect for Fenn
Fenn travels with Az
Winston talks to Fenn alone, admits he's jealous of Az?
Fenn battles mini boss challenge while Az melts ice
Fenn helps Az recover
Winston realizes he's hurt Az and helps her ("this changes nothing!")
Boss Battle
Final platforming or chase sequence
Az and Fenn talk it out with the Wizard

Lastly, I brainstormed a bunch of specific moments, events, actions, or cutscenes that could happen over the course of the level. Then, I organized them based on where I thought they would fall (beginning, middle, or end of the level). Thus, we could pick specific beats we like, and ignore the others. The purpose here is to brainstorm a narrative that is flexible/varied, but still satisfying.

Gameplay Design Level & Player:

Level Overall Description:
Levels are split into sections with each one involving a combat or puzzle scenario with narrative cutscenes in between each section. Think of it almost like a boss rush of minigames. A good comparable is how combat in undertale works with combat in between conversations.

Design Goals:
Level require quick movement
Level requires wit
Level must involve impactful input from the companion character
Player must not feel like the strongest person in the room

Player Abilities:
Leaf mode: The player is able to turn into a bunch of free flowing leaves. This form runs off a meter that recharges while not in use.
Health: Player can only take 2 hits before going down. He’s a flimsy little guy.
Attack: Vine whip attack

Companion Role:
Ice Climbers/Atreus from God of War combo

You and the companion act as one with different buttons controlling who attacks
When you go into leaf dash you separate until you connect up again (OR perhaps better is that AZ always gets to you the minute you stop dashing)

Combat:
You're squishy and can't take many hits so enemies shouldn't take too long to eliminate.
Enemies have specific weak points in the form of runes that when you destroy the enemy goes down
Runes do not need to be attached to the enemy itself and can be placed around the room

Game Comparables:
Guardians of the Galaxy Game/God of War: Companion characters actively participate in combat and can be called to perform special abilities

Example Set:

Section 1:
Encounter with Wizard of Ice Makers, in story it could be funny that he's right at the entrance cuz they didn't realize they didn't get everyone.

This section is a chase sequence with you and AZ running after the wizard of ice who is throwing enemies behind him. You and AZ must work together to get through this fast paced section. Enemies should require hits from but both AZ and the player sometimes in longer then 2 strings.

At the end of the end of the sequence the wizard of ice makers runs through a big door which closes behind.

Section 2:
Az and the player approach a big door that AZ can't break through. There is however a small opening that the player can get through. The player's mission in this section is to get the door open, the trouble is that the player can't fight against any of the enemies inside without AZ. Thus this sequence is all about fast paced movement to be able to reach levers (or any other opening device). I think a good idea might be you have to turn on some heaters to melt the ice that's stopping the door from opening. Once the door is open you and AZ(or maybe even just AZ in a fun little cutscene) are able to clear out the room and move on.

Boss Fight:
After this section is the boss room where the Wizard of Ice Makers is and has been prepping for you guys. I am unsure exactly of how this boss fight could go but my initial idea was that he has made molds of himself and it's kind of a game of finding the real one. You would also have to activate several heaters like spotlights to be able to beat him.

Music

Goal:

Find music that feels fantasy but not traditional fantasy.

Found:

Chick Corea- My Spanish Heart (Album of 16 instrumental songs that blend spanish and jazz to create this amazing fantasy like feel with a fun zesty twist)

Ali Di Meola - Elegant Gypsy (Album of 6 songs around 6 minutes each. Similar instruments and fantasy vibes as My Spanish heart however it uses a very different rhythm and voice)

Version Control and Pipelines in Game Development Production

Version Control

- Version control is the practice of recording changes to a set of files over time so you have a record of what changed when.

Version Control Systems

- A Version Control System is just a system that makes recording changes to your set of files easier, often via automation.
- A VCS will track the history and evolution of your project so you don't have to. For every change, you'll have a log of who made it; why they made it; when they made it; and what the change was. This makes it easier to experiment with new code without breaking working code, collaborate on files or projects without overwriting others' changes, easily roll back changes that broke working code, and more.
    - The capabilities of VCSs make them vital to project management on top of being useful for sharing code.
- There are two major types of VCS:
    - Centralized VCSs, which have a single server that contains all the versioned files and the history metadata. Clients clone the latest version of each file to their machine. Clients check out files to make edits and check in files to save changes to the server. Examples include Perforce and Subversion.
        - Pros: well suited to handling a lot of binaries; scale well with larger projects; easy to see what files (and tasks) a user is working on
        - Cons: server is a single point of failure; stable network connection required to get latest changes or save work to the central repository
    - Distributed VCSs, which store all of the history metadata locally. When clients create a clone, they fully mirror the repository and its history instead of just the latest snapshot of each file. Files do not need to be checked out to be edited; changes can be made at any time, and clients must make a commit to save the changes to the history metadata. Examples include Git and Mercurial.

        - Pros: every clone is a full backup of all the data, no single point of failure; network connection only required to push/pull history metadata
        - Cons: not well suited to handling a lot of binaries; steeper learning curve

Sources:
https://en.wikipedia.org/wiki/Version_control
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
https://mattrickard.com/history-of-version-control-part-1
https://book.mercurial-scm.org/read/intro.html

Production Pipelines

- When a project moves from pre-production to production, developers start writing code and creating assets to construct the game and turn pre-production ideas into a product
- Pipelines are a way to visualize the process that developers go through to create their work and add it to the project
- The two pipelines to keep in mind during production are your Technical Pipeline and your Asset Pipeline
- Technical Pipeline
    - Develop systems, tools, scripts, debug utilities, and other code that speeds up your technical and asset pipelines
    - Develop systems, mechanics, classes, and other code to support the contents of your game, especially loading assets and presenting them to the player
    - In short, a tech developer will use a text editor to write code, save it to the project, and then compile and run the project to check whether any errors occur or changes need to be made
    - Benefit from version control, especially when codebases or team sizes get large
- Asset Pipeline
    - Game assets are anything the program uses to present the game to the player. For example, textures, items, music, AI behavior scripts, and so on.
    - The asset pipeline consists of every step from an asset's conception to loading it in the game
    - In short, an asset developer will use a content creation tool to design an asset, export it to a game-loadable file, and load the asset in the game to check whether any changes need to be made
    - Benefit from version control, especially when projects need a lot of assets or team sizes get large
- Considering these pipelines, teams will benefit from collaborating to set up a version control system for their project, establishing conventions like file names or asset axis orientation, and working together to use version control for project management

Sources:
https://gamedevelopmentwiki.readthedocs.io/en/latest/index.html

https://www.proun-game.com/Oogst3D/ARTICLES/TheGameAssetPipeline.pdf
https://magicmedia.studio/news-insights/the-7-stages-of-the-game-development-pipeline/
https://www.gameindustrycareerguide.com/how-to-become-a-video-game-systems-programmer/

Example projects that use VCS across disciplines that I think we can learn from:
https://github.com/TerryCavanagh/VVVVVV
https://github.com/ppy/osu
https://github.com/SlayHorizon/godot-2d-platformer-demo

Comparable Games and Example Design Methodologies

Began compiling ideas about comparable games and references to their design process, design documents, art books, and more. Currently includes Shovel Knight, Hollow Knight, Celeste, Mario Wonder, and general information about 2D platformers and engine tools. Planning to migrate this information to a spreadsheet for easier information transmission for both research references and game inspirations (much like the spreadsheets created by the Digi Diva team). Current document is here: Research: Design Process in Similar Games - Jeremy

UI/UX Research

Source: https://www.isbe.net/CTEDocuments/BMCE-680093.pdf

- Menu options: difficulty (if applicable), music/sound, video
- Game Menu Design
  - "less is more"
  - Make sure theme is consistent and matches the game
  - Game should be the focus the menus should not take over at any point
  - Clarity
    - menus should be easy to read and understand
  - Clear Hierarchy
    - typology can affect what players preserve as important
    - don't create competition between two elements
  - Pattern consistency
    - establish a design pattern and stick to it
    - don't make users learn multiple menu types

Source:
https://code.tutsplus.com/game-ui-by-example-a-crash-course-in-the-good-and-the-bad--gamedev-3943t

- "A good UI tells you what you need to know, and then gets out of the way."
- Six fundamental questions (directly from article)
  - 1. Does this interface tell me what I need to know *right now*?
  - 2. Is it easy to find the information I'm looking for, or do I have to look around for it? (Are the menus nested so deep that they hide information from the player?)

        - 3. Can I use this interface without having to read instructions elsewhere?
        - 4. Are the things I can do on this screen obvious?
        - 5. Do I ever need to wait for the interface to load or play an animation?
        - 6. Are there any tedious or repetitive tasks that I can shorten (with a shortcut key, for example) or remove entirely?
    - Usability is at the center of it all
    - Fundamental aspects
        - How big is it
        - Does it/should it scroll
        - What info is displayed and where
        - How should a player navigate through it
    - It can important to user test your menus because what seems intuitive to designers may be so confusing for players
        - Ex. what button closes things (expected vs reality should align)
    - Make sure that any heads-up displays (HUDs) are not obtrusive
        - Don't want the HUD to take people out of the immersion of the game
    - Simple is better
        - You want as many things one click away as possible
            - Try not to bury information and DEFINITELY do not bury information that is important at all
    - Try not to reuse icons in a way that might trick players
        - You don't want to have the same icon for two distinct things
    - Having UI that blends into the setting is not necessarily a good thing if it makes the UI useless
    - If you have a really complicated game, it can be good to have additional UI to help players get acquainted
        - Players should be able to turn them off once they have gotten the hang of things

Source:
http://ezproxy.wpi.edu/login?url=https://www.proquest.com/blogs-podcasts-websites/top-reasons-using-illustrations-amplify-games-ui/docview/2522341255/se-2?accountid=29120

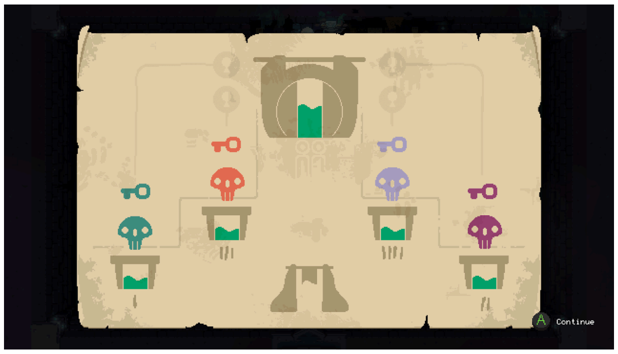
- Using illustrations to make UI/UX better
    - 7 reason why you should use illustrations
        - 1. Provide better understanding of non-english speakers
        - 2. Pictures make interface look interesting
        - 3. Graphics make illusion of actual interaction
        - 4. Better mobile game experience
        - 5. Saves time for designer and user
        - 6. Theme and reputation of brand
        - 7. Storytelling ability

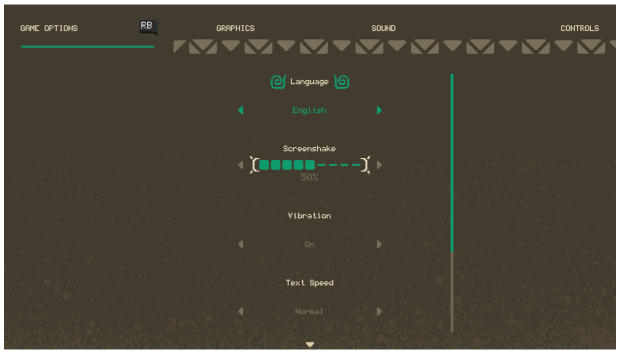
Helpful Websites/Resources

- Many articles about many topics including a lot of UX info
    - https://www.nngroup.com/articles/#popular-alertboxes
- Color Websites
    - https://coolors.co/
    - https://color.adobe.com/create/color-wheel
- Color analysis websites
    - https://cororacle.org/
    - https://www.color-blindness.com/coblis-color-blindness-simulator/
    - https://color.adobe.com/create/color-contrast-analyzer

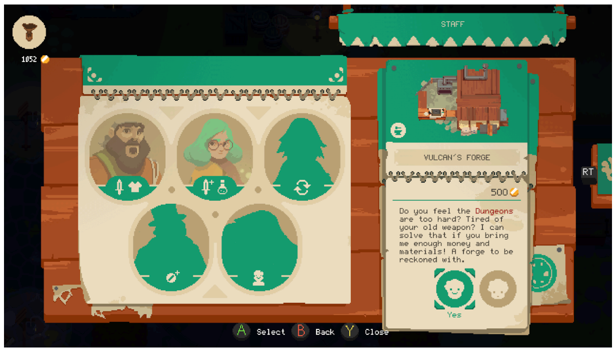
UI Design Inspirations

- All images from https://interfaceingame.com/
- Moonlighter: https://interfaceingame.com/games/moonlighter/
    - Simple graphics, very cute and bright, good color palette, doesn't feel overly complicated or cluttered

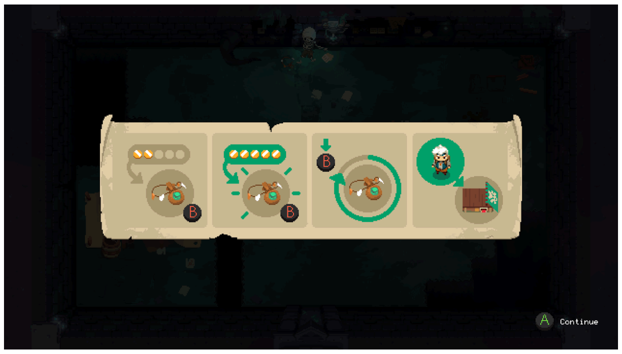




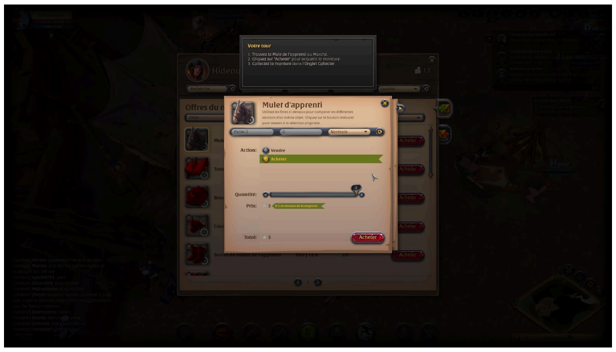
Language
English
Screenshake
50%
Vibration
On
Text Speed
Normal

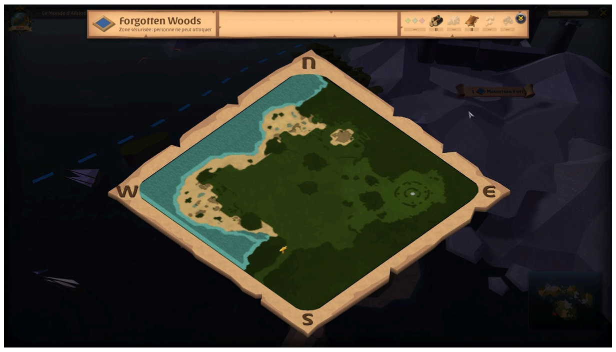


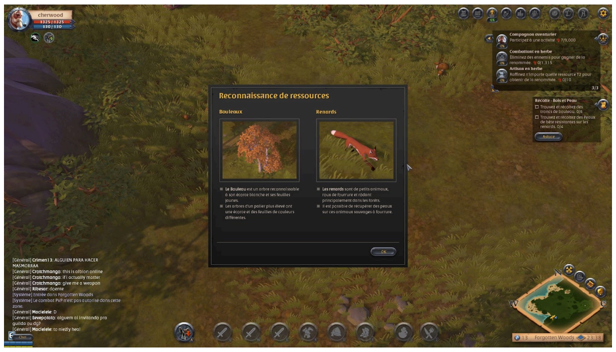
Do you feel the Dungeons are too hard? Tired of your old weapon? I can solve that if you bring me enough money and materials! A forge to be reckoned with.

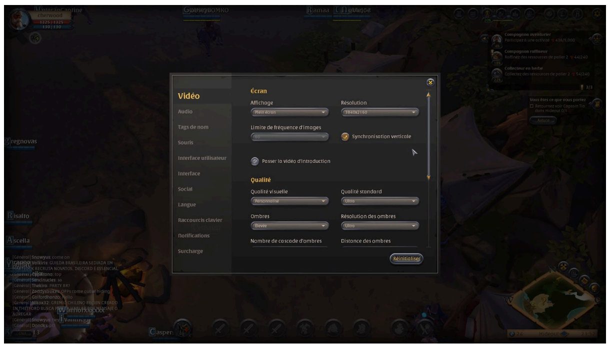


- Albion Online: https://interfaceingame.com/games/albion-online/
    - medieval/fantasy look, good typography

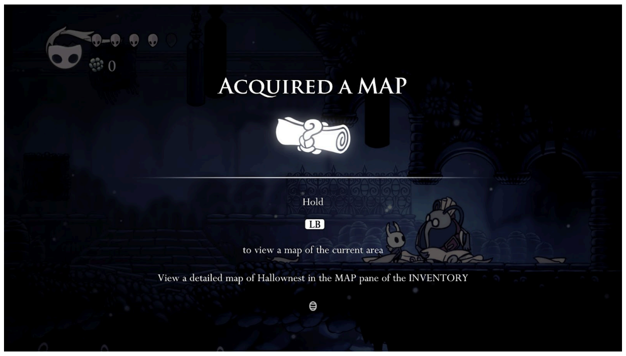






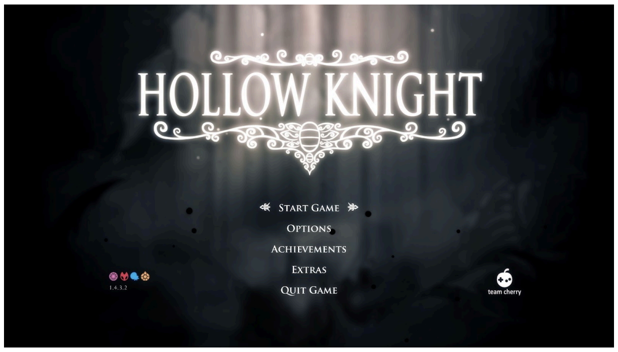


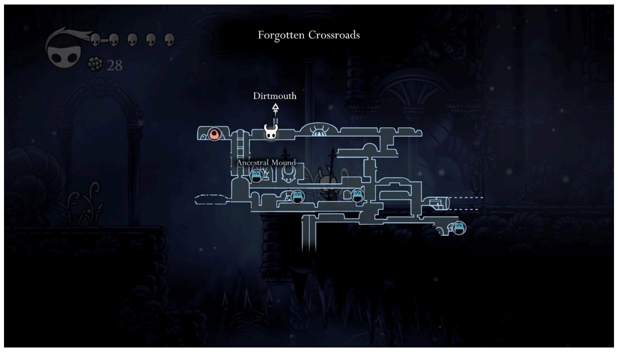
- Hollow Knight: https://interfaceingame.com/games/hollow-knight/
    - Very very simple but effective UI, good use of contrast, using embellishments to enhance UI rather than more complex graphics

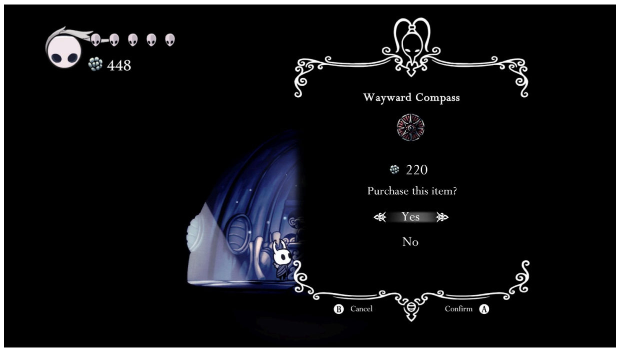


Hold LB to view a map of the current area

View a detailed map of Hallownest in the MAP pane of the INVENTORY







For Potential Further Research

- https://www.gamedeveloper.com/design/setting-the-tone-main-menus-are-the-game
- http://devmag.org.za/2011/02/02/video-game-user-interface-design-diegesis-theory/
- https://researchdiscovery.drexel.edu/esploro/outputs/graduate/The-UX-of-Settings-Menus-in/991020340714604721