Aaron Nemoyten

Game Design Portfolio

Aaron Nemoyten
556 James Avenue, Redwood City, CA 94062
(415) 336-0814
aaron@nemo10.net

Summary

This is a long and comprehensive portfolio, so I've included a summary of its contents at the top along with information about which areas of game design each project covers.

Here is a video of gameplay design and programming work from unreleased projects.

Oberak

Wildseed Games, 2023 (PC, Unreal Engine)

SYSTEMS DESIGN, COMBAT DESIGN, LEVEL DESIGN, TECHNICAL DESIGN, LEADERSHIP

Led game design for Wildseed's unfinished game Oberak. The demo we were working on when the company shut down will be released on Steam soon. **Here's a playthrough video**.

Unannounced Project

Mesmeron, 2022 (PC, Unity Engine, prototyped and pitched only)

SYSTEMS DESIGN, COMBAT DESIGN, TECHNICAL DESIGN, LEADERSHIP

Led design for this ambitious ARPG that was pitched to investors but not funded. Built combat and AI for a Unity-based prototype.

The Garbage Pail Kids Game

Jago Studios, 2020-2022 (iOS/Android, Unity Engine)

SYSTEMS DESIGN, COMBAT DESIGN, CONTENT DESIGN, TECHNICAL DESIGN, MONETIZATION DESIGN, ECONOMY DESIGN, LEADERSHIP

Promoted to Creative Director for this mobile free-to-play RPG in 2020. Worked with a globally-distributed team to ship key features and maintain a steady pace of live content updates.

Moustachevania

Personal Project, 2017-2021 (PC/Mac, Unity Engine)

LEVEL DESIGN, SYSTEMS DESIGN, TECHNICAL DESIGN, *A MOSTLY-SOLO PERSONAL PROJECT*

This was a solo project built in Unity, first as a bare-bones game with platforming-focused progression, then as an ambitious vertical slice with progression focused on collecting 'charms' that basically function as level-editing tools. This was pitched to publishers but did not receive funding. I built a character controller, save system, dialogue system, and more.

Idle Overlord

Personal Project, 2018-2020 (Originally iOS/Android, now available on WebGL, Unity Engine)

SYSTEMS DESIGN, PROGRESSION DESIGN, MONETIZATION DESIGN, ECONOMY DESIGN, TECHNICAL DESIGN, *A MOSTLY-SOLO PERSONAL PROJECT*

Designed and programmed this mobile-targeted hybrid idle game featuring extensive spreadsheet-driven systems design. Also built an online community on Discord and used analytics to measure performance. The game was not completed but is playable on itch.io.

Color Switch

Color Switch Ltd, 2018 (iOS/Android, Unity Engine)

SYSTEMS DESIGN, LEVEL DESIGN

Served as Lead Game Designer for the game's Unity-based relaunch. Was responsible for making sure the game was a pixel-perfect match to the original BuildBox version and designed new game modes and levels.

Heroes of Dragon Age

Electronic Arts, 2013-2015 (iOS/Android, Unity Engine)

SYSTEMS DESIGN, CONTENT DESIGN, MONETIZATION DESIGN, ECONOMY DESIGN

Sole combat system designer for Heroes of Dragon Age. Prototyped on paper and then in Unity, then used Excel to design and balance 100+ characters with four tiers each for the game's launch.

Superhero City

KlickNation Inc., 2009-2011 (Facebook, PHP/JavaScript)

SYSTEMS DESIGN, UI DESIGN, TECHNICAL DESIGN, MONETIZATION DESIGN, ECONOMY DESIGN, PRODUCTION, LEADERSHIP

Joined KlickNation as employee number six and wore many hats, including lead SE, producer, and game designer. Designed many systems including Leagues, League Wars, League War Tournaments, Equipment, Raids, and City Mastery.

Madagascar

Toys for Bob, 2005 (PS2/XBox/Gamecube, RenderWare Engine)

CINEMATICS SCRIPTING, CINEMATICS DIRECTION, WRITING, GAMEPLAY DESIGN

Cutscene design and scripting for this 2005 game.

Personal Projects: Level Design

LEVEL DESIGN, *PERSONAL PROJECT*

Designed and released two Doom 2 levels. Some of the most fun I've had as a game designer!

Oberak

Wildseed Games, 2023 (PC, Unreal Engine)

SYSTEMS DESIGN, COMBAT DESIGN, LEVEL DESIGN, TECHNICAL DESIGN, LEADERSHIP

Overview

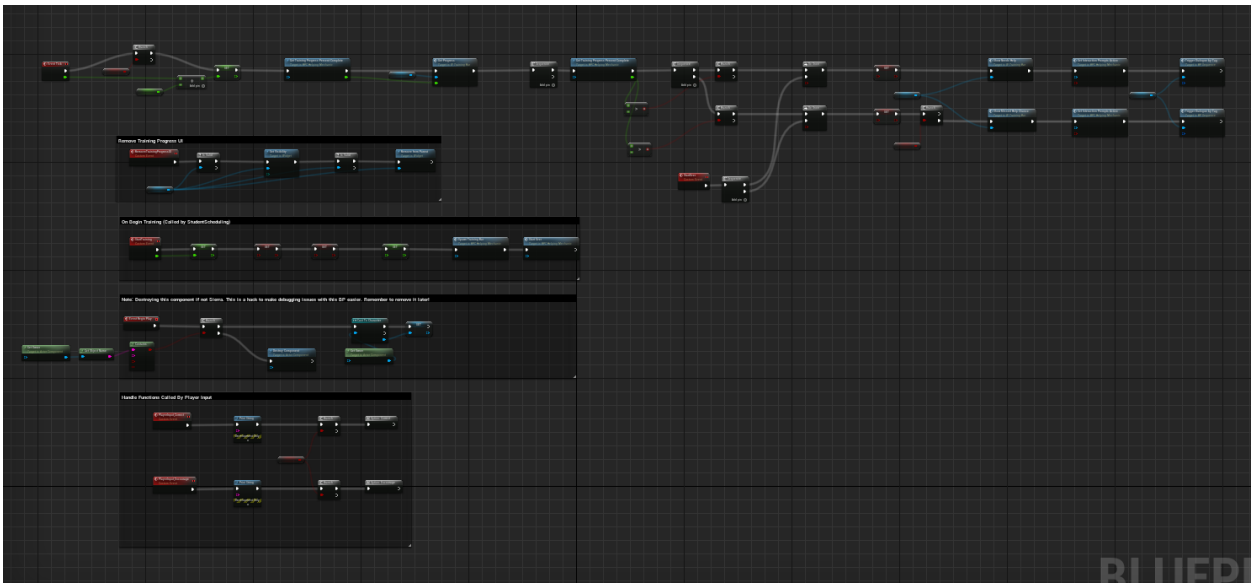
I spent a too-brief five months working on Oberak before Wildseed Games unfortunately had to shut down due to funding issues.

I was the most senior game designer on the team, reporting directly to the Creative Director.

- Design Leadership
    - Led design on the game's showcase 'demo' level for investors, outlining the complete experience using Miro while consulting with artists, animators, and programmers on capabilities and scope. Collaborated with the team on a narrative framework for the demo and gave feedback on writing. Provided feedback on combat design that kept the team focused on immediate goals, focusing on refining each mechanic in order to 'find the fun' before moving on to the next.
- Systems Design
    - Refine and simplify existing systems design for prototyping and playtests. Write specs, review with team, and implement in UE Blueprints.
    - Designed and implemented the Workout and Helping mechanics, in which players pick a workout for a student and then optionally 'encourage' or 'correct' them, which causes them to either finish the workout successfully or fail.
- Level Design
    - Designed traversal level section aimed at introducing the player to wall-running and mantling. Redesigned the game's "Hanato" (home base) area to create clear sight-lines between points of interest. Other contributions on additional level sections to maintain narrative, visual consistency, and ease of navigation.
- Combat Design
    - As part of a multidisciplinary team, drive combat design towards our Creative Director's stated experiential goals. Started from zero for a fresh combat demo experience and worked up to a fully-featured real-time martial arts-inspired combat system with combos, a dodge, special moves, and more. Worked with Unreal Animation Montages to adjust timing of animations, hit timing, VFX, and more.
    - Designed and implemented a Combat Area system that manages combat encounters from start to finish.
- UI design and programming with Blueprints for Workout/Helping mechanics, Combat Tutorial, Intro/Prologue, and more.
- Also contributed to art direction, writing dialogue, combat AI, audio programming, particle effects,

Workout Help Mechanic

The project spec called for a fairly complex system of stats and interactions to drive a feature where players could set a schedule of workouts for students, who would then need either moral support or technical instruction to improve their workout performance. For our investor demo, I pared the system way down into something that would just show what the final result would look like, without any of the stat checks going on in the background.

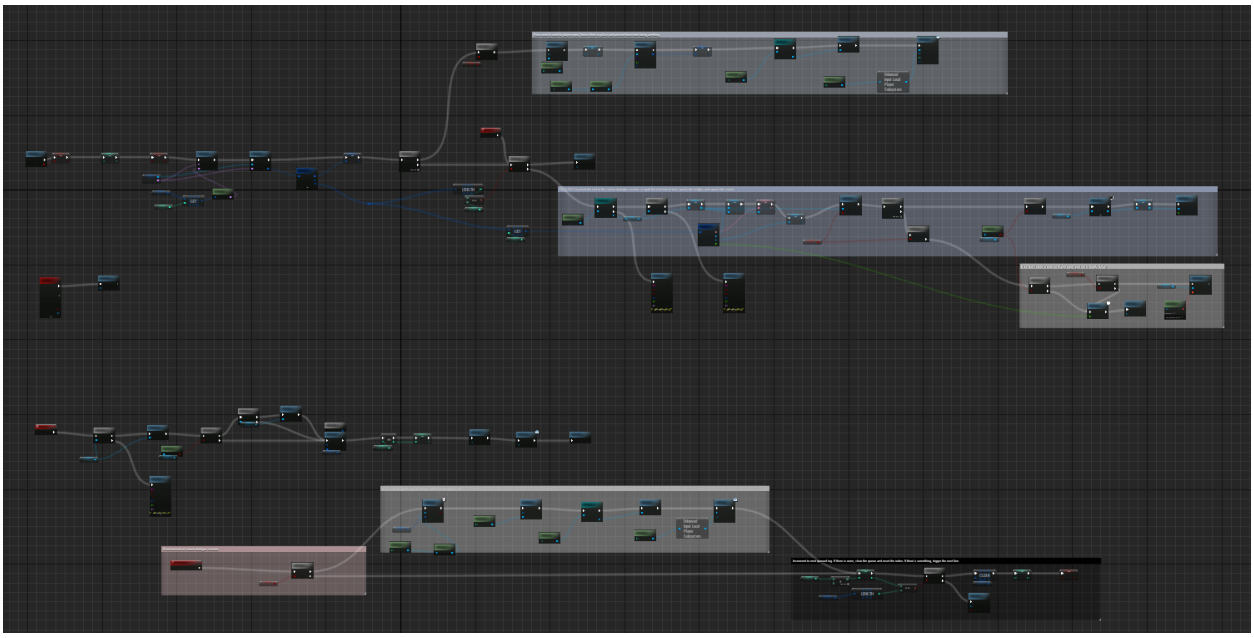


The “Working Helping” mechanic’s blueprint

The sequence at the very top is called every tick once the student has started working out and uses a number of “Do Once” gates to make the student ask for help and enable input, then disable input once the workout is nearly complete. If the player has helped the student, they will successfully complete the workout. If not, the student will “fail” the workout and play an animation in which they fail to complete the last pull-up and fall on the ground.

There was a lot more to this in another blueprint that makes the student walk to the workout location, play animations, and spawn UI elements as well. (not pictured)

Dialogue Manager



The dialogue manager blueprint.

Oberak's original dialogue implementation was a quick hack for a week-long game jam, so it relied on actors having trigger volumes and then adding a UI directly to the viewport. This was really easy to work with, but it meant that there was no central way to manager UI and multiple dialogues could trigger at the same time. Plus, it always drew itself on top of the Lyra framework's Pause and Options menu!

I rebuilt Oberak's dialogue system to use a central manager that could be accessed via our custom game mode, and which added a single instance of the dialogue UI to the gameplay layer of Lyra's UI framework. It was also refactored to account for two different types of dialogue: Lines that could play without pausing gameplay, and lines that paused gameplay and required player input to continue. This required some careful branching to account for lines they had voiceover recorded and lines that didn't, and different behavior for each depending on the context, to keep everything relatively playable and consistent before dialogue was finalized and VO could be recorded. It's also set up so that 'cutscene' dialogue can interrupt gameplay dialogue but cannot itself be interrupted, and separate gameplay dialogues triggered independently would queue up instead of interrupting each other.

Most of the branching pictured in the Blueprint is around the different types of dialogue, with the grey sections at the top and bottom accounting for suspending and resuming gameplay, respectively.

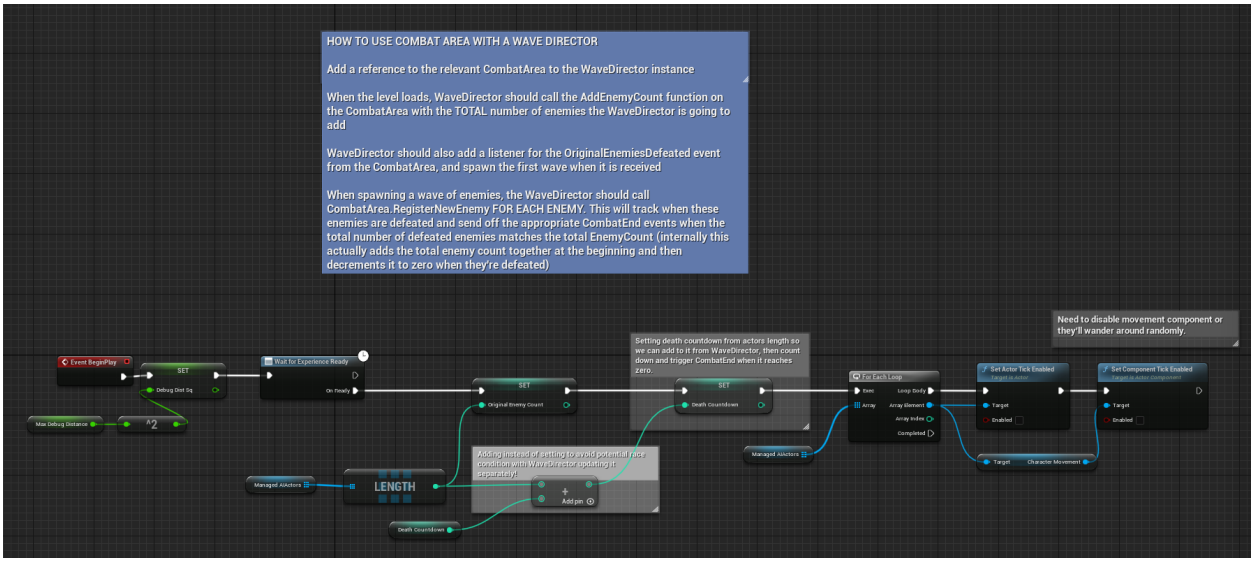
All dialogue was driven via a data table that used tags as row names. Each data table entry could include as many lines as necessary for the exchange or cutscene.

Gameplay dialogue (while the player is running across the bridge).

Cutscene dialogue, which pauses gameplay and requires player input to continue.

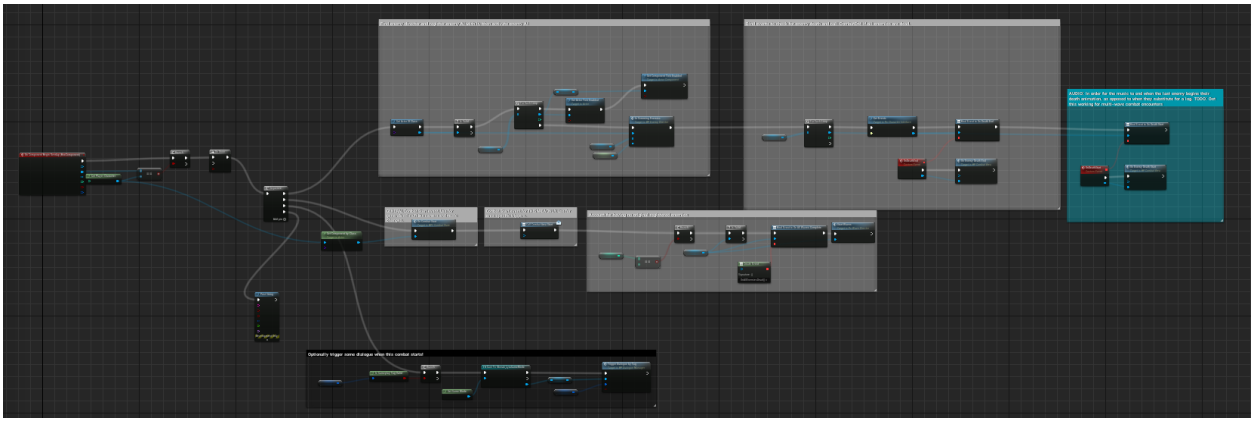
Combat Area

The team eventually decided on arena-style combat encounters in which encounters would have discreet areas and would only be complete (and allow the player to leave) once enemies had all been defeated. A few piecemeal systems had been created to start music and put up barriers, but they didn't work well together so I built a fresh Blueprint script that handled everything at once and dispatched events to other game systems for combat start, combat area start, combat end, and combat area end to manage the player going into a combat pose, enemy AI engaging with the player, music starting and stopping, etc.



Combat Area's BeginPlay event. The comment includes detailed integration instructions!

On BeginPlay, the script counts up the enemies within the combat area instance and then disables them.



The combat area's OnBeginOverlap event, which activates the combat area when the player enters.

Once the player's collider overlaps with the combat area's collider, a few things happen:

- The 'combat barrier' goes up (VFX and collider) to keep the player within the combat area
- All enemy AI is turned on
- The player character switches to their combat stance
- Custom events are bound to enemies' OnDeath events
- Combat music starts
- CombatStart and CombatAreaStart events are sent
- Dialogue is optionally triggered (the player character and a faceless enemy exchange some words when each battle begins)

When each enemy is defeated, the total enemy count is decremented and when the last enemy triggers their death event, combat ends and the following things happen:

- The 'combat area' dissolves and the collider is disabled
- The player character switches back to their relaxed stance
- Combat music ends
- CombatEnd and CombatAreaEnd events are triggered

I'm pleased with how easy it was to set up combat areas: Drop an actor into the level, set a reference to the combat barrier for that encounter, and add the relevant enemies to an array.

Unannounced Project

Mesmeron, 2022 (PC, Unity Engine, prototyped and pitched only)

SYSTEMS DESIGN, COMBAT DESIGN, TECHNICAL DESIGN, LEADERSHIP

Overview

I spent a number of months working with Jago Studios' CEO (and an actual Hollywood writer/producer who will remain nameless) to develop and IP and game pitch for a free-to-play horror-themed third-person ARPG. While we were ultimately unable to find funding for the game, our pitch did get us a number of meetings with high-profile companies and investors.

Besides the pitch and IP development, I also spent some time prototyping accessible third-person combat in Unity in a networked-friendly way using a library that a friend's company had developed for their game.

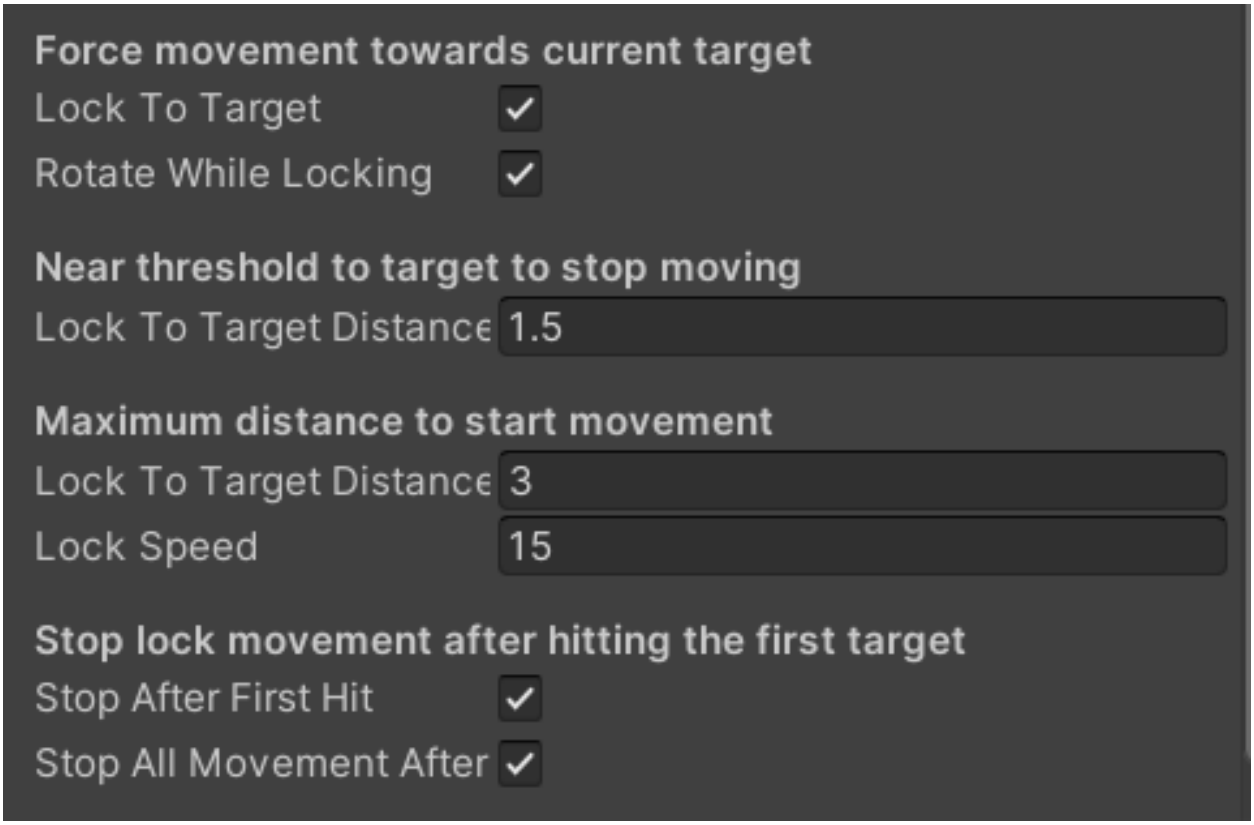
Accessible ARPG Combat

We wanted to build combat that required some skill in terms of enemy prioritization, skill usage and cooldowns, and movement, while still keeping things accessible enough for a broad audience.

To make this work, I quickly built out two important features: Generous auto-aim and lock-on melee attacks.

Generous auto-aim was relatively simple to implement: I cast out a large cylinder in front of the player and picked the closest enemy it touched. The enemy's health display would highlight when it was the currently selected target to make sure it was very clear where the player character would attack next.

Melee attacks are easy to implement at a basic level, but I wanted to make sure we were translating player intent as much as possible without requiring precise targeting, so I added a whole lot of options for every attack. This allowed us to tune the player turning towards and moving towards the target enemy, stopping at the right time and distance, moving at the right speed, and locking movement until the attack animation was done playing.



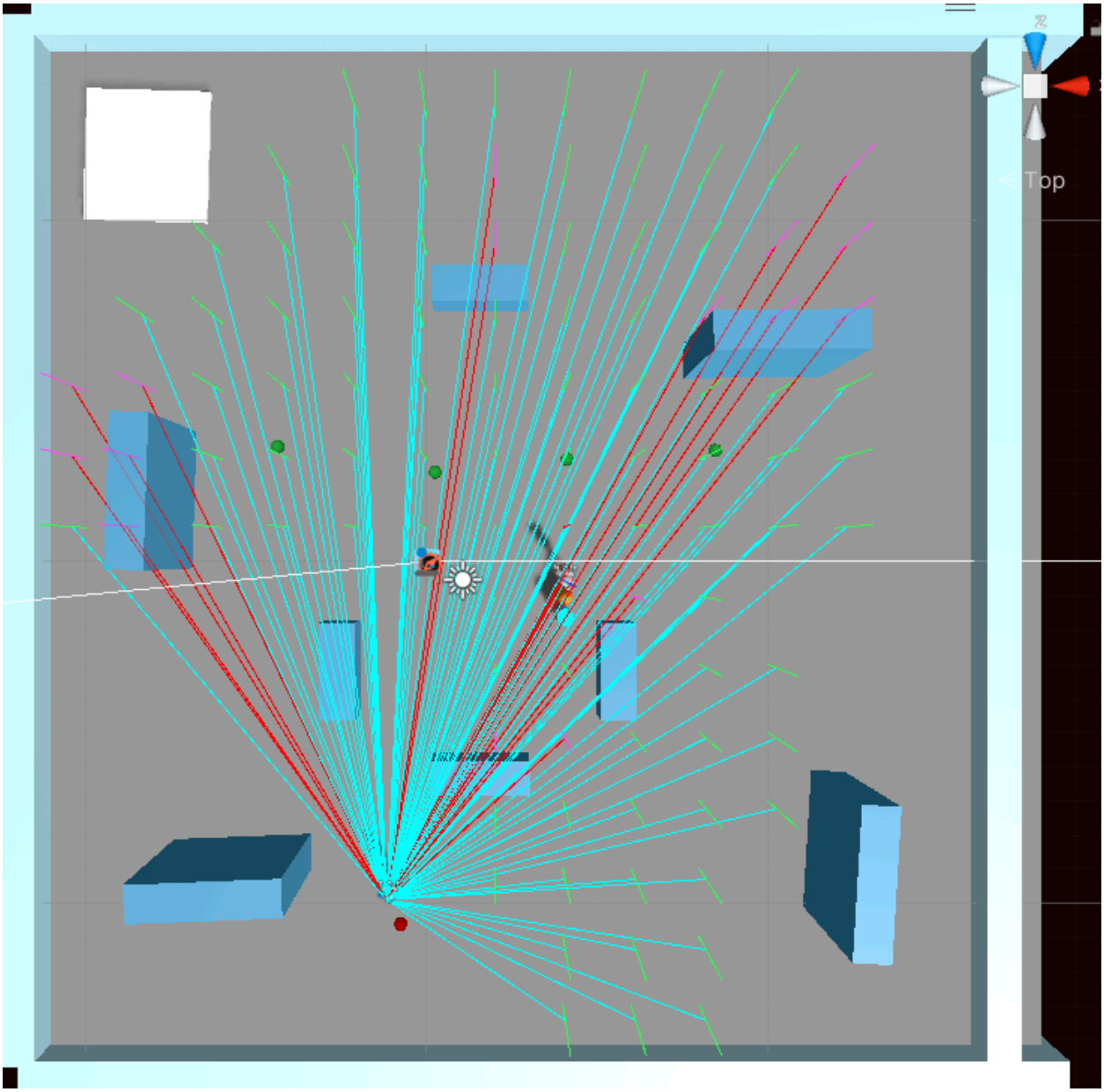
Lots of options for melee attacks and how they do or do not lock onto enemies.

Reverse-Cover AI

Inspired by a GDC talk about how enemies in DOOM find the best locations from which to attack the player, I built an enemy AI manager from scratch that did the following:

- At run-time, automatically create a grid of waypoints
- Check each waypoint to make sure it's on a valid part of the navmesh and remove if not

- Enemies, at intervals, get a selection of all surrounding waypoints, check if the player is visible from them and that they are reachable, and then picks one that is closest to the “ideal attack distance” (configurable) away from the player
- “Claimed” waypoints are removed from the waypoint selection temporarily so other enemies don’t try to occupy the same spot
- If the player gets “too close” or “too far” (configurable), try to move further away from them to another valid waypoint
- Repeat



Debug view for cover checks from potential enemy destinations to the player.

The Garbage Pail Kids Game

Jago Studios, 2023 (iOS/Android, Unity Engine)

SYSTEMS DESIGN, COMBAT DESIGN, CONTENT DESIGN, TECHNICAL DESIGN, MONETIZATION DESIGN, ECONOMY DESIGN, LEADERSHIP

Overview

On the Garbage Pail Kids Game, I started out as a Producer but as the game transitioned to live operations I became the game's Creative Director, initially handling all game design (including live operations data entry), eventually delegating various responsibilities as other team members could be trained up.

- Oversaw all game design, live operations, and writing starting in October of 2020, working with a fully-remote team of 12
- Picked characters from Topp's vast GPK card library, wrote animation and sound effects briefs, and provided final approval for animation and SFX
- Designed and implemented all character abilities beginning in October of 2020
- Designed the game's Worm Wars game mode
- Redesigned the game's PvP scoring and matchmaking system to increase engagement and competition among the game's most active players
- Completely redesigned PvE events and dramatically increased engagement
- Created a Google Sheets template the team used for every monthly content update, including templates for characters, events, loot boxes, and messaging

Major Features

Worm Wars

Worm Wars was a high-level feature that used procedural content to dramatically boost engagement among elder players.

Inspired by Galaxy of Heroes' roguelike-inspired Galactic War feature, Worm Wars used a random team generator instead of saved PvP teams to provide a set of fresh and challenging battles every day.

The team generator had a chance to create a fully 'cohesive' team, where all members shared a faction and the leader slot was filled by a character who actually had a leader ability. The generator also had a chance to match teams based on other criteria, and tried to balance the team between DPS/tank/support archetypes. Everything was based on chance though, with a deliberate chance for a completely random team that wasn't strategically cohesive at all. This was also a great way to demonstrate new characters and their abilities, since as soon as a new character was released it had a chance to show up in Worm Wars

Worm Wars in action. The map alternates between battles and treasure chests.

PvP

As originally launched, PvP had some technical issues and had rewards balanced for a much larger DAU. After rebalancing rewards and rewriting the scoring and ranking systems, I led the design of a Battle History tab to address players' concerns that their ranks dropped between sessions and they wanted to understand why. A Battle History feature seems obvious in hindsight, but prioritizing against a lot of other features was difficult.

The Battle History feature in action. Not pictured: A Revenge button for when players lose while defending. The Revenge button is, to say the least, an important feature.

PvE Events

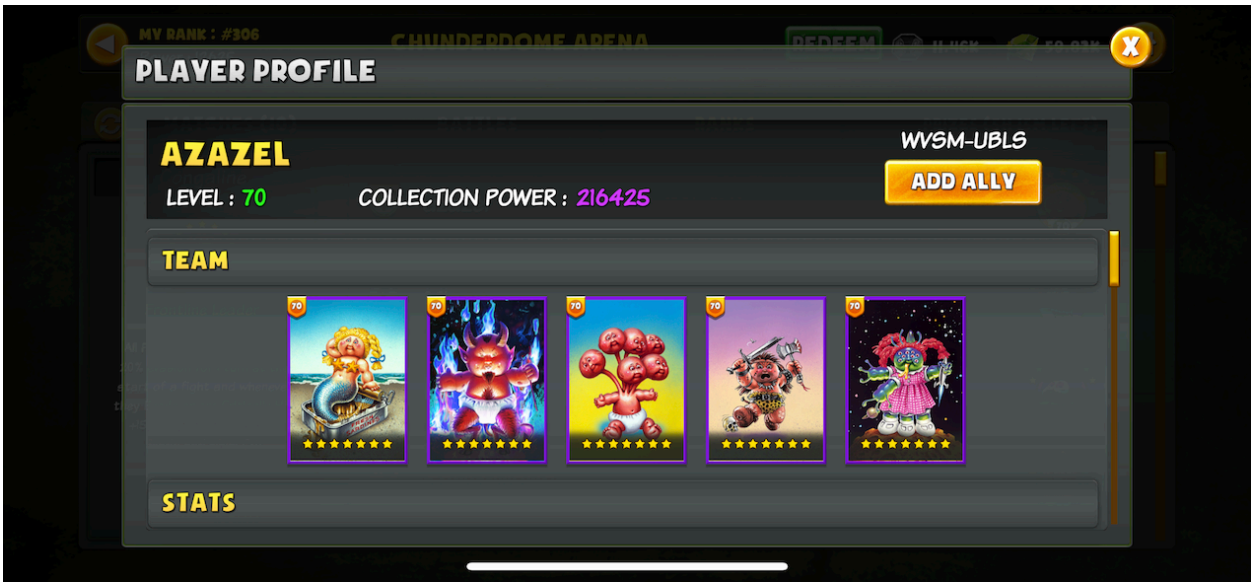
Redesigning PvE events increased engagement with the feature by a factor of 5. Our original PvE event design was based on acquiring a character by collecting all of their tokens across seven increasingly challenging battles, which meant that once the player engaged in an 'event' once and unlocked the character, they could never play it again! With a smaller player base than we had anticipated, we needed to find a way to boost engagement that didn't rely on PvP, so we opted to redesign events. I designed a new version of PvE events that included both limited-quantity rewards and unlimited rewards for each battle, which encouraged advanced players to attempt every battle in an event to collect all of the limited rewards while also allowing lower-level players to grind the highest tier battle they could actually complete. With a limited number of completions available each day, elder players couldn't beat the entire event in a single day and everyone else still had a reason to come back every day to grind, even if they were stuck on the first or second battle.

PvE events were redesigned to support both limited and unlimited rewards.

(continued on the next page)

Social Features

GPK launched with no social features outside of PvP. I led the design of an Allies system that also included a brand new player profile screen and a new currency that could only be acquired by sending and receiving to allies and which could be spent on a new “Ally Pack” in the store.



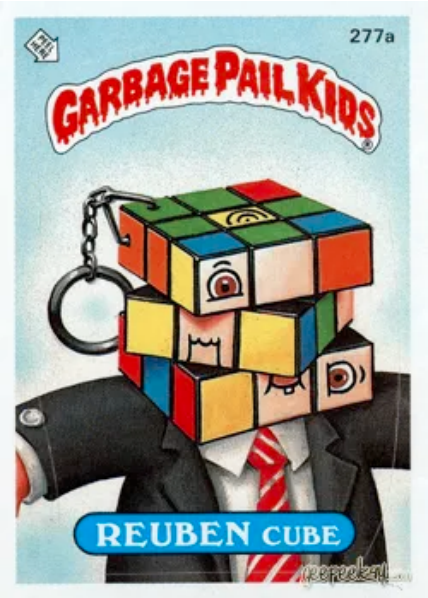
Viewing the profile of another player.

Ally list UI. The Allies feature included an exclusive ‘Ally Coin’ currency that could be used to buy an exclusive pack in the shop.

New Characters and Abilities

I love to tell stories about characters through their abilities. Here are a few of my favorites:

Reuben Cube



Basic attack: **Blockheadbutt**

- Reuben attacks one enemy. If Reuben has any debuffs, apply copies of each to the enemy.

Special attack: **Solved**

- Reuben taunts for 2 turns if not already taunting and converts all debuffs to equivalent buffs.

Special attack: **Scrambled**

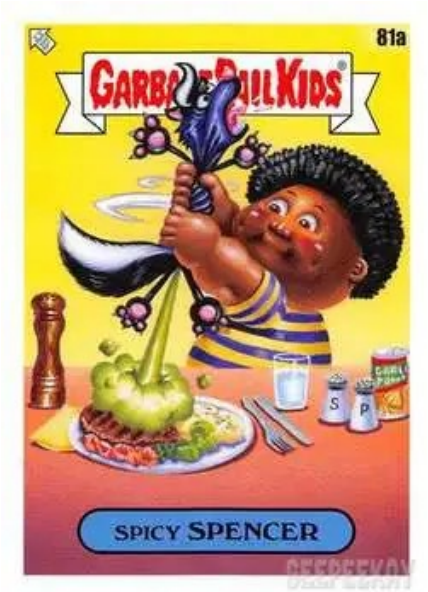
- Reuben taunts for 2 turns if not already taunting and applies 3 debuffs to himself.

Leader ability: **Geeky Squad**

- While Reuben is taunting and has no debuffs, Geeky teammates receive +5% Physical Damage.

The focus of this design was to tell the story of “scrambling” the cube, which applies debuffs, and then “solving” it to convert the debuffs to buffs. When Reuben taunts during these attacks, enemies are forced to attack him, which will allow him to collect *even more* debuffs that the player can choose to ether convert to buffs OR use his basic attack to apply them to enemies.

Spicy Spencer



Basic Attack: **Special Sauce**

- Spencer attacks all and applies Offense Down for 2 turns

Special Ability: **What Smell?**

- Spencer Taunts for 2 turns. He also removes up to 2 debuffs from each teammate, adds them to himself, and heals 2% of his max health for each debuff added.

Passive Ability: **Skunked**

- Whenever Spicy Spencer is attacked, 25% chance to apply either Offense Down or Defense Down to the attacker for 2 turns.

Spicy Spencer clearly enjoys the smell of skunk, so everything about his ability design is about how smelly he is, and just how much you wouldn't want to touch him. He actually enjoys it so much that he's able to transfer debuffs from teammates to himself and it heals him in the process – like having something that most would consider a huge disadvantage is still good for him. And of course his passive ability shows that anyone who touches him is just gonna end up stinky too.

I think if I could rebuild the game from scratch, ideas like being smelly or slimy would be their own specific status effects that would function in some specialized way… but since we didn't have anything like that and there were other priorities, I had to simulate those ideas using the existing mechanics and systems.

Moustachevania

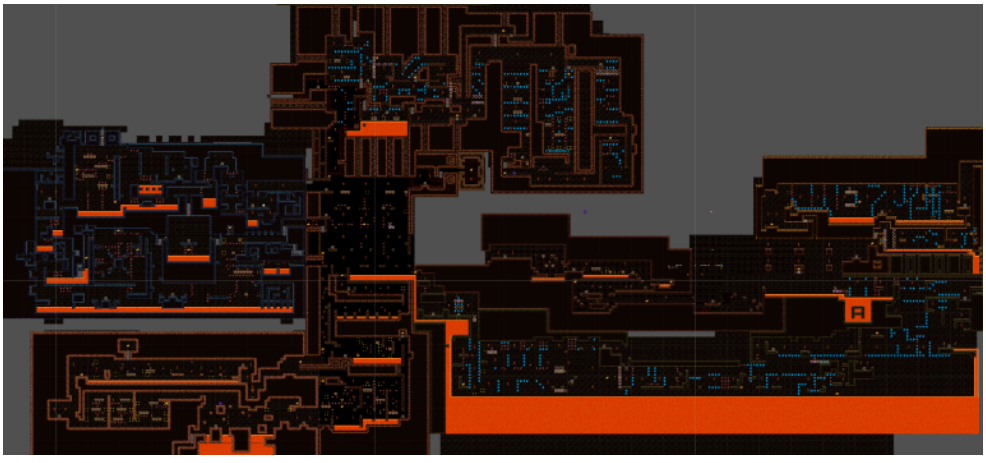
Personal Project, 2017-2021 (PC/Mac, Unity Engine)

LEVEL DESIGN, SYSTEMS DESIGN, TECHNICAL DESIGN, *A MOSTLY-SOLO PERSONAL PROJECT*

Overview

Moustachevania was a platformer that I worked on, on and off, between 2019 and 2022. It started as a very modest project but I scoped it way up and made a vertical slice demo in an attempt to find a publisher. Unfortunately, I wasn't able to find a publisher, so I put the project on the shelf, probably permanently. Still, it was a lot of design work! Here are some highlights:

- Built a deep and complex character controller from scratch that included "coyote time," input buffering, dash, double jump, teleport, advanced speed boost mechanics, and various convenience/usability features (inspired by Celeste) that corrected for slight errors by bumping the player a few pixels in the "intended" direction when hitting floors, walls, and ceilings depending on controller inputs and current direction and facing.
- Built one complete two-hour "metroidvania"-style level that relied solely on a progression of platforming abilities (pictured below)



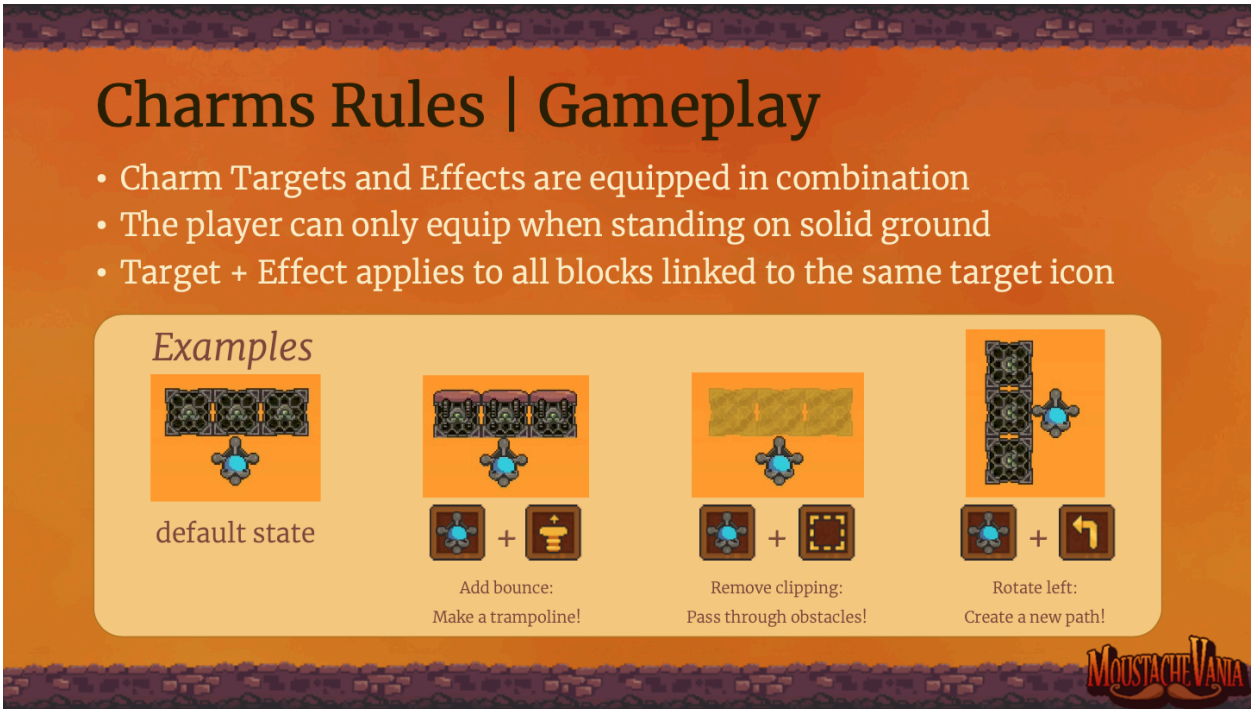
Two hours of gameplay in a single level.

- Built a second complete experience that takes roughly an hour to complete with a progression that also includes a ‘Charm’ system (described in detail later)
- Built a dialogue system that integrated with Yarn and included custom actions for controlling camera, particle systems, sound effects, and more.
- Developed a save system that works by assigning all collectible, destroyable, and activate-able objects unique ids, then simply appending the id to a list when the object has been collected, destroyed, or activated. When loading a saved game, all objects are initially spawned in the world, and then will check the saved list for their unique id. If it’s found, they’re collected, destroyed, or activated automatically.
- Developed a custom-built auto-tiler that accounts for 30 different combinations of surrounding tile states.

Charms and Adjustable Objects

One of the unique selling points of the scoped-up version of Moustachevania was its ‘charm’ system, which would allow the player to collect items that manipulate parts of the level in specific ways. The inspiration was “what if I built a Metroidvania but the progression was actually level design tools?” - an extremely technically ambitious design idea that I distilled into a much simpler concept where certain platforms could be manipulated in different ways.

Here are some slides from the Moustachevania pitch deck that explain the charm system:



Charms | Gameplay

Target Charms

Sets of blocks and other objects grouped with color/shape coded gems can be affected with a combination of a Target and Effect charm.

Effect Charms

Rotate left/right

Make Ghostly/Solid

Stop, start, slow

Breakable

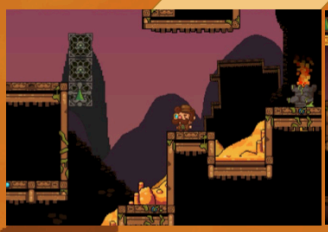
Add/remove bounce

Remove spikes

MOUSTACHEVANIA

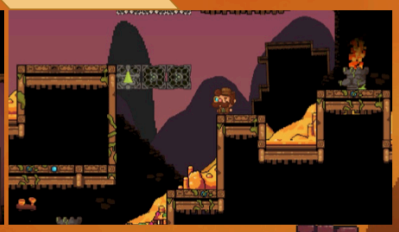
Area Progression | Exploration

Explore new area → Find Ability-Gated Location → Explore to find new equipment or ability → Return to gate with new ability and pass → Unlock new area



Find a blocked path...

Explore and find a new ability



Return and un-block path

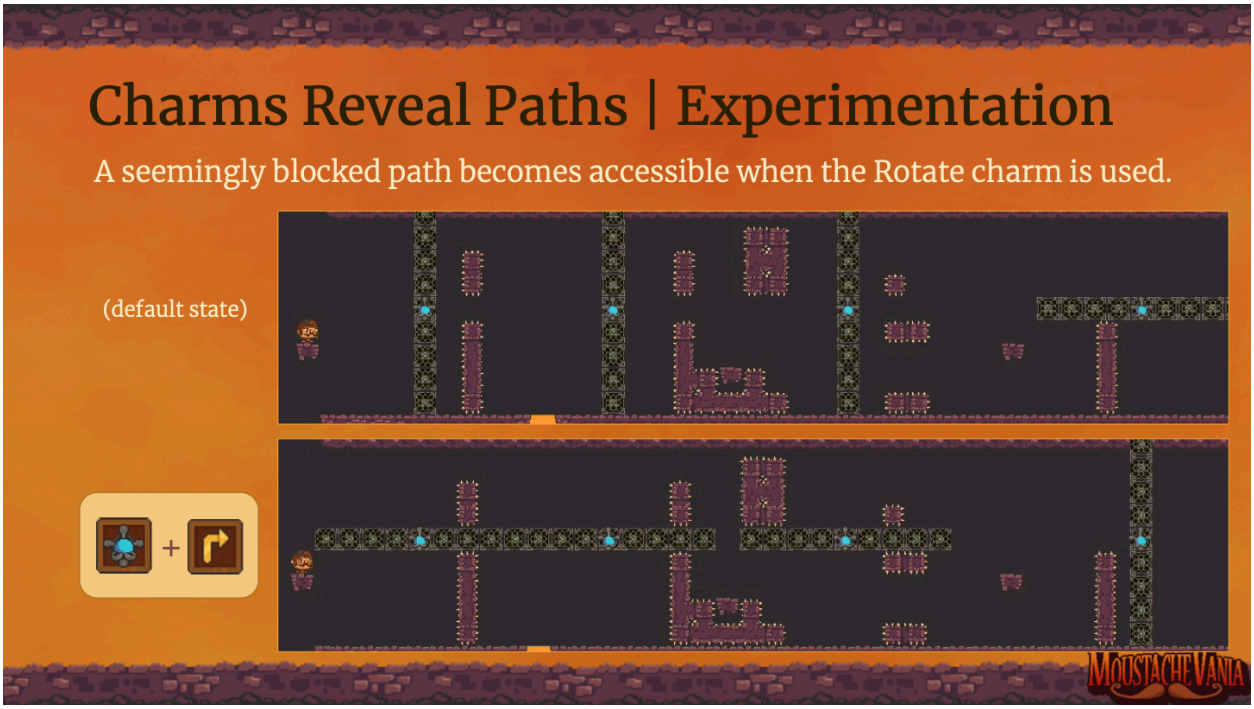
Reveal Alternate Routes | Experimentation

With no active Charms, this area is impossible.

Reveal Alternate Routes | Experimentation

Bottom path becomes accessible

Top path becomes accessible (albeit difficult)



Idle Overlord

Personal Project, 2018-2020 (Originally iOS/Android, now available on WebGL, Unity Engine)

SYSTEMS DESIGN, PROGRESSION DESIGN, MONETIZATION DESIGN, ECONOMY DESIGN, TECHNICAL DESIGN, *A MOSTLY-SOLO PERSONAL PROJECT*

Overview

Idle Overlord was a mostly-solo project I built with the goal of combining the addictive mobile idle game gameplay of Idle Miner Tycoon with the core fantasy and mission structure of old Facebook games like Mafia Wars. I acted as the sole game designer on the team, bringing in a few friends to help with some of the coding and SDK integration for mobile builds. The game was initially developed over six weeks in spare time during a YCombinator 'Startup School' season, and in keeping with YC's "talk to customers as early as possible" mantra, I posted a survey on the idle games subreddit and created a Discord server for interested players to discuss idle games and provide feedback on builds when they became available.

Some highlights of my work on Idle Overlord:

- A data-driven idle building economy balanced to last for weeks to months at a time
- A procedural system to generate 'minions' with unique stats, rarity, and names
- A 'mission' system that requires minions and is designed to generate high engagement during the early game and high retention further into the player lifecycle
- A collectible and consumable 'boost' system that can speed up missions or increase resource generation speeds

- Every system in the game was data driven, using TSVs exported from Google Sheets

Generators

The key to balancing an idle game is that each additional ‘generator’ needs to be slightly less efficient than the one before it *at the same level* but more efficient than the one before it *when the one before it is upgraded.* Here’s a screenshot of the spreadsheet driving the system:

A	B	C	D	F	H	I	J	K
Id	Initial Cost	CostScaleFactor	BaseGenerationAmount	TimeToBuild	<-- Serialized Va	Reference --->	Base Efficiency	Output V Cost>
1	20	1.15	1	0	0		0.05	20
2	410	1.15	19	30	30 seconds		0.0461	380
3	8508	1.15	362	60	1 minute		0.0425042	7240
4	178658	1.15	7001	600	10 minutes		0.0391888724	140020
5	3796472	1.15	137175	1800	30 minutes		0.03613214035	2743500
6	81624145	1.15	2719213	3600	60 minutes		0.03331383341	54384260
7	1775325161	1.15	54529741	7200	2 hours		0.0307153544	1090594820
8	39057153532	1.15	1106081276	14400	4 hours		0.02831955676	22121625520
9	869021666088	1.15	22690704341	28800	8 hours		0.02611063133	453814086820
10	19552987486973	1.15	470718661545	43200	12 hours		0.02407400209	9414373230900
11	444830465328639	1.15	9873559285233	86400	24 hours		0.02219622992	1974711857046
12	10231100702558700	1.15	209378698202646	86400	24 hours		0.02046492399	4.18757E+15
13	237873091334490000	1.15	4488346464021020	86400	24 hours		0.01886865992	8.97669E+16
14	5590017646360510000	1.15	97249002835943400	86400	24 hours		0.01739690444	1.94498E+18
15	13276291910106200000	1.15	2129510039600070000	86400	24 hours		0.0160399459	4.25902E+19
16	318631005842549000000	1.15	4712179815627040000	172800	48 hours		0.01478883012	9.42436E+20

The columns shaded in red are imported to use in the game, while the others are simply used as design reference.

Generators, in this case, are the floors of the player’s underground base/lair. This is where the core of the game economy exists, so it had to be airtight. The key thing about idle games is that the player is always balancing upgrading existing generators with buying new ones, and that balance isn’t interesting unless there’s some conflict between how efficient it is to buy vs upgrade. To keep this conflict interesting, there are a few factors at play:

- The first level of each new generator generates cash less efficiently as a ratio of cost to cash generated than the generator before it (column J)
- Each upgrade increases the generator’s output by its initial output, but costs 1.15 the previous upgrade cost (column C) – so each upgrade is less efficient… this means that the advantage of upgrading vs. buying new diminishes after a few upgrades
- However, generator efficiency doubles and premium currency is awarded at upgrade increments 10, 25, 50, 100, 150, 200, 300, and 400 (this is defined in another tab). This means that there’s still an incentive to upgrade rather than buy anew, at least until a certain point is reached

Notice that each generator *also* has a ‘Time to Build,’ which creates a delay between spending the resources to build the generator and the generator being completed. This provides an extra resource sink for the game’s premium currency, a feature modeled after base-building/tower defense games like Clash of Clans.

Missions

Idle Overlord also included Missions, a system inspired by Mafia Wars and similar games. Missions give the player something to do while they wait for their generators, and also act as s vehicle to tell the story and flesh out the game's world.

Id	City	Name	Mission Req	Time (s)	Description	Image	Priority	Skill	Skill Id	XP Factor	Repeat	Loot Table (nam	Final Loot Table	Item Req	Generator Req	Generator Level	Influence Req	GeneratorCashR	GeneratorCashL	Story
1	1	Get in the spirit		10s	Move a few case	mission_1	0	Bootlegging	1	1	1			0		0	0	0	1	X
2	1	Shake down a schmuck	1	10s	Another dumb m	mission_1	0	Photography	2	1	1			0		0	0	0	5	X
3	1	Crack some knuckles	2	30s	Help a local crev	mission_1	0	Management	3	1	1			0		0	0	2	1	X
4	1	Enjoy big-screen magic	3	120s	Who even steals	mission_1	0	Haggling	4	1	1	Basic Plus 25%	Chance of Gold+	0	2	0	0	1	5	X
5	1	Take a cheap shot	4	10s	Hit a local bar or	mission_1	2	Safe Cracking	5	1	25			0		0	0	2	1	
6	1	Borrow a garbage truck	4	1m	You gotta start s	mission_1	0	Fast-Talking	6	1	1			0	3	0	0	2	5	X
7	1	Clone a credit card	6	5m	It's amazing wha	mission_1	0	Hacking	7	1	1			0	3	0	0	3	1	X
8	1	Forge a passport	7	10m	Why steal wallet	mission_1	0	Counterfeiting	8	1	1	Basic One of Anything		0	3	0	0	3	5	X
9	1	Get local garbage contract	8	30m	Put in a low-ball	mission_1	0	Politics	9	1	1	Green+ Mission Speedups		0	4	0	0	3	10	X
10	1	Give people what they want	9	30m	Cook up a batch	mission_1	2	Pharmacology	10	1	10	Random Hidden One of Anything		0		0	0	4	1	
11	1	Load up a minivan	9	1m	There's a shipm	mission_1	2	Smuggling	11	1	100			0		0	0	4	5	
12	1	Cut a few corners	9	1h	Who needs cash	mission_1	1	Gem Cutting	12	1	5	All Mission Speedups		0		0	0	4	10	X
13	1	Forge a Banksy	12	30m	Spraypaint a girl	mission_1	0	Art	13	1	1	All Instants		0	5	0	0	5	1	X
14	1	Build an acid bath	13	5m	Sometimes you j	mission_1	0	Chemistry	14	1	1	Random Hidden One of Anything		0	5	0	0	5	10	X
15	1	Dump low-level waste	14	10m	Take on a new a	mission_1	0	Nuclear Physics	15	1	1			0	6	0	0	6	1	X
16	1	Bypass the guards	15	2h	Just because yo	mission_1	0	Piloting	16	1	1	Purple Only Generator Speedups		0	6	0	0	6	10	X
17	1	Make liquor, but quicker	16	1h	Turn moonshine	mission_1	2	Bootlegging	1	1	5	All Instants		0		0	0	6	15	

A few key features of Missions:

- There's a 'mission required' column so missions can unfold in a mostly-linear fashion
- Missions have a required generator, which should encourage players to continue to build new generators instead of focusing on upgrading existing generators
- Missions have a tagged Skill, which determines which assigned minions get bonus XP for completing the mission
- Loot tables are referenced by name instead of id so design can be done in the spreadsheet far more easily
- Some missions are repeatable. This gives players something to do if they're blocked by a mission that requires a generator they cannot yet afford
- Very importantly, mission payouts are actually dictated by generator payouts instead of being absolutely defined. This is beneficial both for saving time balancing the game and to manage payouts for a later "City Reset" feature (or "Prestige," for those of you familiar with the term) that would allow the player to start the city over again with increased payouts for everything. This also guarantees that mission payouts seem "worth it" relative to the most recent generator the player unlocked around the time the mission became available.

Minions

The minion list in action. Idle Overlord definitely needs a UI art pass that it will probably never get.

The implementation of minions in Idle Overlord was inspired by the Managers feature in Idle Miner Tycoon, but I wanted to spice it up quite a bit. Minions in Idle Overlord have a rarity, four randomly-rolled stats (Charming, Evil, Devious, and Sneaky), an optional specialization for Legendary minions, and a name generated from a table with prefixes, suffixes, and first names.

The minion name generator is inspired by mafia movies, so it generates names like Lazy Eye Frankie or Hamburger Tony or (since it's gender-inclusive) Lizzie the Lizard.

Minions can be recruited for free every four hours, by spending premium currency, or with in-game currency.

Minions can do two things: Be assigned to generators to increase their output, or go on missions to gain experience. Minions increase generator output based on their relevant stat (as each generator utilizes a specific skill, which has a stat tied to it) and Legendary minions will provide a huge boost if matched with a generator that matches their specialization; For example, a Legendary minion with the Politics skill matched to the Political Research generator (obvious, I know) will provide a massive boost… but the player will want to occasionally send them on missions to level them up, which will increase that boost even more.

Goals and Upgrades

To encourage players to fully engage with all of the game's systems, Idle Overlord also has Goals and Upgrades systems. I'll let the screenshots speak for themselves:

Unfinished But Still Playable

If you're curious, Idle Overlord is available to play in your browser on itch.io.

Color Switch

Color Switch Ltd, 2018 (iOS/Android, Unity Engine)

SYSTEMS DESIGN, LEVEL DESIGN

Overview

Lead Designer for the Unity relaunch of the hyper-casual hit game. Highlights include:

- Led recreation of the original version's physics and obstacles by creating side by side videos of each version and comparing until they were pixel-perfect copies
- Along with a junior designer, led the recreation of the original game's bespoke levels for levels the team liked and created new levels to replace the ones we weren't happy with
- Led design of new game modes including Brick and Phoenix, and created levels for those modes alongside our junior designer
- Iterated on tools and process for rapid prototyping alongside Color Switch's lead engineer and artist

Heroes of Dragon Age

Electronic Arts, 2013-2015 (iOS/Android, Unity Engine)

SYSTEMS DESIGN, CONTENT DESIGN, MONETIZATION DESIGN, ECONOMY DESIGN

Overview

On Heroes of Dragon Age, I served as the game's sole Combat System Designer from pre-production until just after the game launched. Duties included:

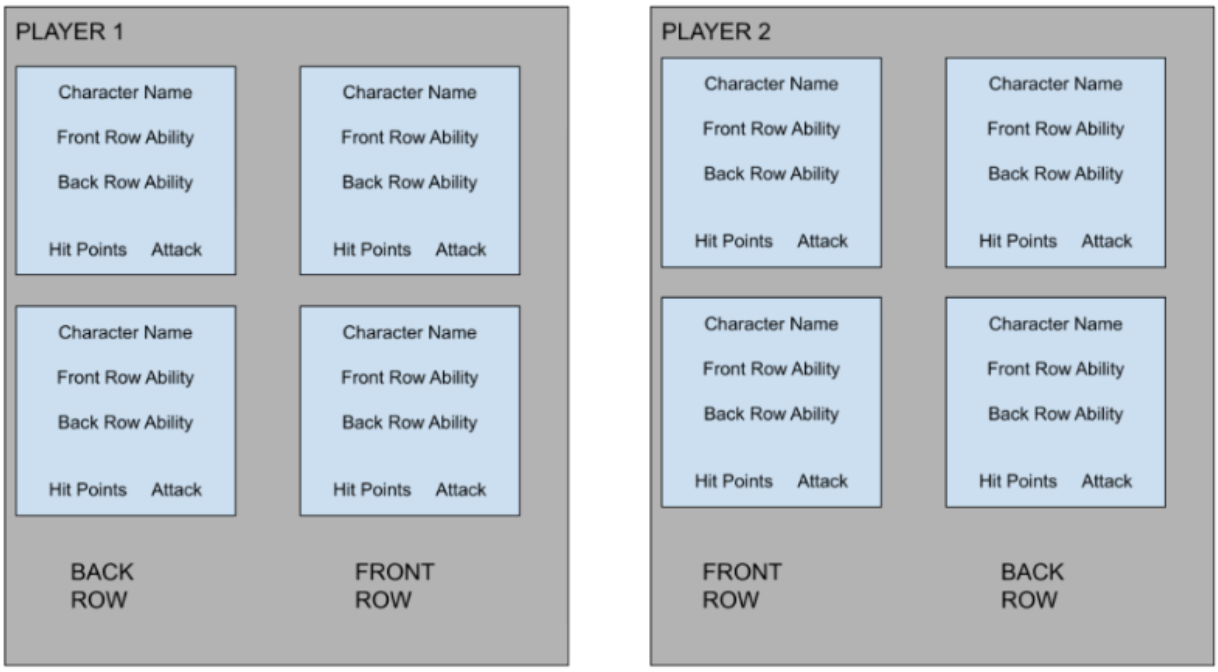
- Sole designer of initial paper prototype for combat system
- Designed a Unity prototype along with one programmer
- Collected, organized, and prioritized characters from across the Dragon Age IP
- Complete design of the combat system and all character abilities
- Built and maintained a master design spreadsheet with all characters at all rarities and projected relative power levels
- Built and maintained XML export scripts for characters, live service PvP and PvE events, store items, and announcements

Paper Prototype

An initial paper prototype was created by brainstorming character 'card' ideas, then creating the cards by making Powerpoint slides and printing them four to a page and cutting them up.

I ran a studio-wide playtest for several hours, allowing players to pick and arrange their cards into a 2x2 grid, then rolling dice for initiative and attack damage.

Since Heroes of Dragon Age's combat system is entirely automated, character choice and placement are the only strategic decisions players can make, and our test players took this very seriously. They did end up forgetting some rules or components of card abilities when playing, so I took that as a cue that they were too complex for players to remember during battle and removed them from the final game design.



Recreation of the paper prototype of Heroes of Dragon Age. The real cards are under NDA, of course.

Final Combat System Design

For the final combat system design, the team compiled an extensive list of every possible character in the Dragon Age franchise, including generic characters like "city guard" to fill out the lower rarities.

Once we decided to add a character to the game, they would be added to a master design spreadsheet used for balancing all characters.

Each character has four tiers of increasing power (and visual ornamentation), tracked in separate rows in the spreadsheet. This meant that for the game's launch with over 100 collectible characters, there were 400+ rows in the spreadsheet.

Each character's attack was broken down into base strength, targeting (single, row, column, or all), speed (which affects turn order), and special effects like stun chance or slow chance. All of these were given a proportional contribution to an internal power rating number that could be used to balance characters against each other.

In an effort to create a brute-force rock/paper/scissors mechanic comparable to other CCG elemental systems, we also designed a "Faction" system that was loosely based on how

characters might be classified in the Dragon Age games. (In hindsight it may have been more confusing than some alternatives!)

A character detail screen. Notice the ‘faction’ color (red) at the top.

The Combat Balance Simulator

It was vitally important that we be confident in the balance of our combat system, not just in evenly distributing power but also in making sure that there was no team that would win 100% of the time against every other team.

To verify this, we built a server endpoint to which we could submit XML that contained a list of teams and a number of times for each of them to fight each other.

This also helped us verify our assumptions about the relative power of various stun chances, AOE attacks, and debuffs.

Heroes of Dragon Age’s combat system required many iterations of updating its master design spreadsheet’s formulas, re-exporting all character XML, running simulations, and parsing the results. Fortunately, that work paid off and the game was balanced enough that despite nearly every character having a unique combination of power, health, and attack attributes, there was no consistently dominant team composition.

Superhero City

KlickNation Inc., 2009-2011 (Facebook, PHP/JavaScript)

SYSTEMS DESIGN, UI DESIGN, TECHNICAL DESIGN, MONETIZATION DESIGN, ECONOMY DESIGN, PRODUCTION, LEADERSHIP

Overview

There's a lot more detail about Superhero City in my resume, since I joined as a software engineer when the team was very small and ended up wearing a lot of hats, including lead game designer (since we had no full-time game designers on the team). Here are some highlights from my design work on the game:

- Solo designer and developer of Leagues, League Wars, and League War Tournaments systems, a trifecta that tripled revenue when first introduced together.
- Designed and coded City Mastery feature, which quadrupled single-player content by adding difficulty tiers with increased rewards to each "city" in the game
- Designed and partially coded Raids, a highly engaging social feature
- Partially designed and fully coded the game's Equipment system, which added item slots tied to the player's avatar and supported equipment with a variety of attack and defense bonuses. The system was modular, so ever N months the team could add another slot to provide a revenue boost immediately

Unfortunately, Superhero City was a Facebook game that was shut down after KlickNation was acquired by EA and became Capital Games, so there's very little relevant visual media available for this portfolio.

Leagues, League Wars, League War Tournaments

One of the great things about building a game on Facebook was that because all game accounts were linked to Facebook accounts, players could easily create Facebook Groups to get together and talk about the game. These groups quickly turned into Leagues (or Clans, but we called them Leagues in SHC because it felt more appropriate).

Eventually, players started to change their in-game names, adding prefixes that indicated which League they were in. Then they started setting up tournaments, organizing times for all players from one League to battle all players in another League, then tallying the wins and losses manually.

Looking at this activity taking place mostly outside the game, we quickly moved to built Leagues and League Wars as formal features within the game.

Because battles in SHC are asynchronous, they're really just a test of which character has better stats. But since there's a degree of randomness to damage dealt, dodges, critical damage, etc., outcomes of battles between somewhat similarly-matched players are not necessarily a foregone conclusion.

So, figuring that the system the players were already using informally was working pretty well, I designed a League War system that worked like this:

- A League could challenge another League to a League War
- Challenges had a time and duration (in days) and would need issued and accepted by League admins
- Challenges had an optional ‘ante’ – an in-game currency requirement that each player would need to put into a central ‘pot’ as part of the prize for winning
- Challenges also had a “team size limit,” which would limit the number of collectible abilities player characters would bring to battle, and would also therefore increase the influence of Equipment, which had more complex stats like Dodge Chance. This would also somewhat negate the effect of big spenders who could collect hundreds of premium-exclusive abilities.
- Once a challenge was accepted, all pending challenges would be hidden and a giant countdown would appear for both Leagues
- When the League War starts, all players on both teams can attack any other player up to two times. But wins and losses both count, so attacking a player with no hope of winning is a bad idea.
- Normally, PvP battles would result in the winner ‘stealing’ a percentage of the loser’s ‘unbanked’ currency – in League Wars, the currency won in each battle would go into a central ‘pot’
- Players are still subject to the normal daily battle cooldowns, which require premium currency to refresh.
- (There might have been some extra limitations that I forgot – it’s been 12+ years!)
- After the time limit expires, the wins and losses are added up, and the winning team gets to split the currency in the ‘pot’ evenly among all members

After designing League Wars, I worked long hours to implement it as quickly as I could, basically by myself. KlickNation didn’t even have a UI designer at the time, so version 1 of the feature was plain-looking to say the least. Building a game that’s basically a web site with relatively low UI/UX standards makes for extremely efficient development!

After League Wars launched, we quickly noticed that the larger unofficial Leagues from the Facebook Groups organized themselves into multiple smaller Leagues in the game, since the membership limit was only 10 (if I remember correctly). Then, rival sets of Leagues would have tournaments, where the lower level Leagues would battle each other, etc.

Much of the tournament was run by a single player who was also a youth soccer coach. Rather than build an entire automated League War Tournament system in the game, I built an admin interface that would allow that player to organize and schedule an entire bracket of League Wars at once, and then made sure he received a check for his efforts.

During the first official week-long League War Tournament, **daily revenue *tripled***.

We made sure there was a new League War Tournament every few months after that.

City Mastery

Superhero City's had entertaining single-player content appropriately broken down into 'cities,' but it was expensive to build and time-consuming to balance, so the pace of new city development was very slow.

Figuring we could squeeze a lot more engagement out of our cities, I set out to design and develop a feature called City Mastery.

City Master is pretty simple in concept: When a player completes all missions in a city, reset all progress and start again with a higher difficulty.

There's a bit more to it than that; There are several numbers that needed to be scaled, like energy cost, currency payout, etc., and I decided to scale them all differently. They catch is that boss battles reward unique abilities and premium currency, so there's a major incentive to get extra boss rewards. But since one of the goals of the feature was to soak up extra player energy, I actually made the other payouts *less* efficient relative to the energy spent. This was obscured by the fact that the energy requirement for each mission went up as well, so in absolute terms the payouts *were* going up.

I set the maximum number of mastery levels at 4 and voila: That's how I quadrupled the game's single-player content in less than a week!

Madagascar

Toys for Bob, 2005 (PS2/XBox/Gamecube, RenderWare Engine)

CINEMATICS SCRIPTING, CINEMATICS DIRECTION, WRITING, GAMEPLAY DESIGN

Overview

Way back in 2004, I spent six months at Toys for Bob working on the Madagascar PS2/XBox/Gamecube game. I was hired as a *very* junior "cinematics guy," asked to implement in-game cinematics based on storyboards from an artist. The team quickly grew to trust me and allowed me to build most of the game's cutscenes without any storyboards, so most of the time I just churned through the work autonomously based on which levels weren't already checked out from version control by somebody else. Here are some highlights:

- Directed and scripted about 80% of the cutscenes in the game
- Wrote several key scenes, including the end of New York Street Chase and the scene where the main characters first meet the lemurs
- Created a bullet hell minigame over a weekend using the game's proprietary scripting language, which was integrated into the game as an easter egg in the second level
- Responsible for calibrating automatic lip-sync settings for the game's fully-rigged main characters using internal tools

Here's a link to my 'cutscene demo reel' from the game on Youtube.

Here's a link to a video of the 'spaceship minigame' I created in a weekend.

Personal Projects: Level Design

LEVEL DESIGN, *PERSONAL PROJECT*

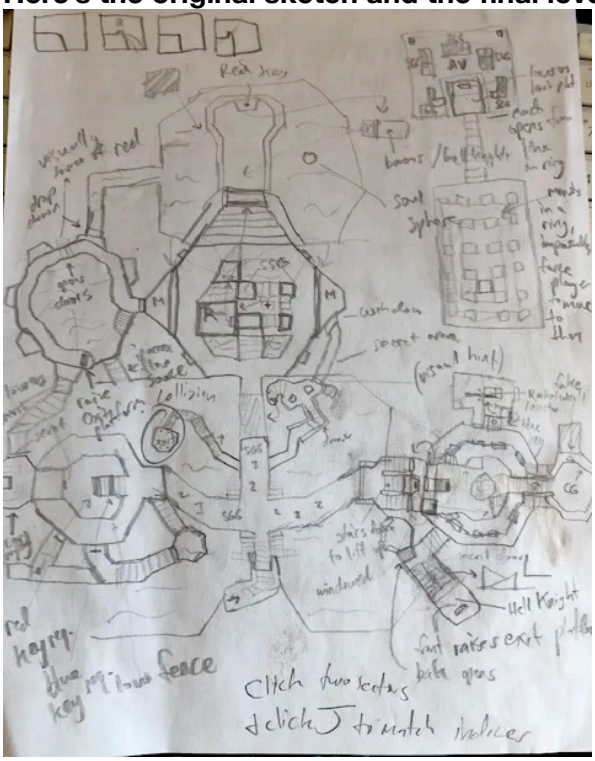
Doom 2: Fork And Knife In The Road

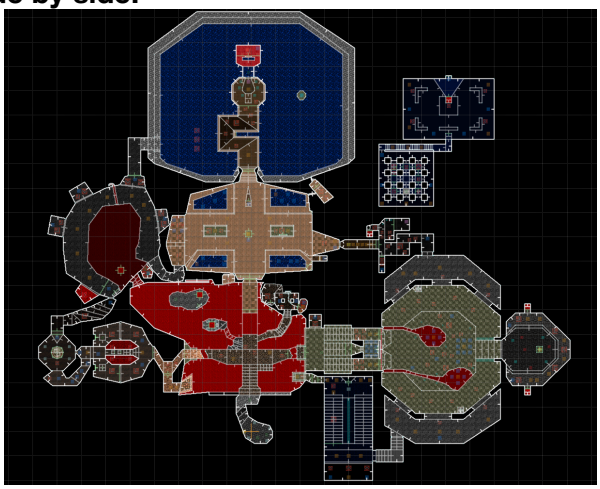
I love open-ended FPS levels with multiple objectives that can be completed in any order. I sketched out this level on a piece of paper more or less as it is in-game, with a few key ideas:

- A central outdoor area that splits into three possible paths, but that also has an isolated area with a clearly marked exit
- One path leads to two keyed doors
- The other paths are both accessible, and both branch off into two paths each, one of which ends in a key and the other ends in a switch that lowers a barrier to the centrally-visible exit
- Both main paths have a major weapon pickup, and both include encounters that are better-suited to that weapon

Based on the above criteria, the four branches can be completed in any order with roughly the same level of difficulty.

Here's the original sketch and the final level side by side:





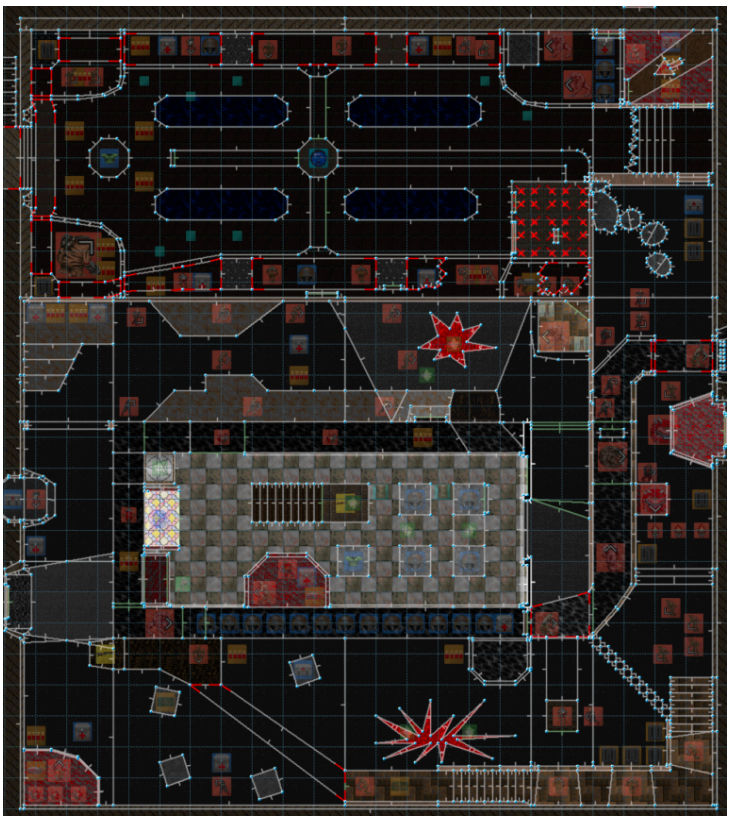
To watch a playthrough of the level and for a download link, check out the level's page on my portfolio web site.

Doom 2: Fueled By Blood

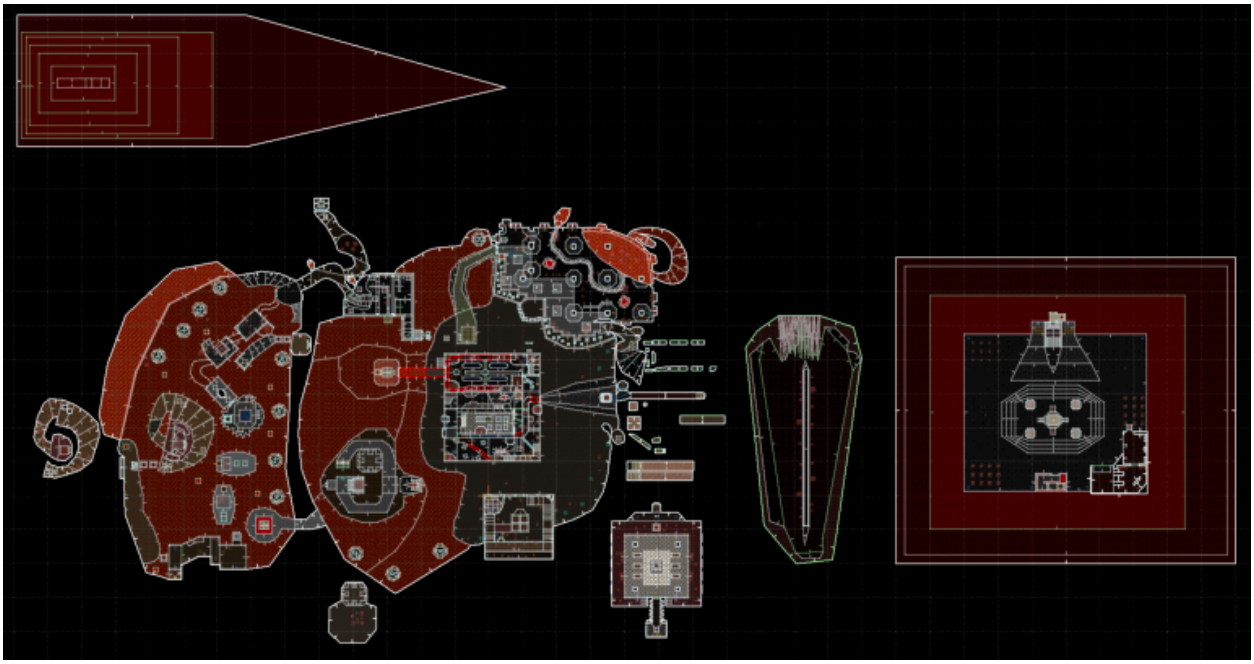
This was a much bigger and more ambitious level than Fork and Knife In the Road. The original idea was to make the entire level a spiral that keeps returning the player back to the center, and to make the center room change every time the player returned to it. After a few loops back to the middle though, it started getting more and more difficult to build variation into the journey back outwards, so I turned the central spiral area into one area out of many in this huge central area. Many of the sub-areas of the level are also spirals in a way; They force the player to traverse the same area more than once to complete an encounter, collect a key, or hit an important switch.

Ultimately, I'm quite happy with the level. Through multiple rounds of playtests, I was able to refine the puzzles and learned how to better highlight switches and other areas of interest by paying closer attention to lighting, line of sight, and the paths that players tended to take versus ignore. Even when they got stuck, testers had fun and found the combat to be challenging without feeling unfair, and (to my pleasant surprise) had kind words to say about the aesthetics of the environment as well.

For a more detailed postmortem and download link, as well as links to playthrough videos with and without commentary, check out the level's page on my portfolio web site.



The original central area of the level.



The final, complete level. Notice the above-pictured area towards the center.

A screenshot from Fueled by Blood