**My Brand**

**Skills**

* Strategist – Find solutions, big picture
* Philomath – Love learning, acquire knowledge
* Time keeper – planning, deadlines
* Optimist – find the good
* Problem solver – Find solutions, find flaws, can’t ignore issues

**Specialties**

Level Design

* Environment design
* Tutorial

UX Design

* Accessibility, readability, ease of use
* Feedback, effects

Technical Design

* Tools programming

**Strengths**

Lorem ipsum

**Weaknesses**

Lorem ipsum

**Homepage**

I’m Luke, a Level designer and UX designer. I specialize in level layouts and stage design. I love creating levels that utilize their environment and build off of the world that their a part of. On the UX side, my focus is accessibility and feedback. I always make sure that the player can see what they’re able so they’re never lost. I particularly enjoy creating tutorials, as the cross between level and UX. I strive to make them feel natural and not pull the player out of the experience.

I’ve worked on many 2D and 3D games, creating levels, user interfaces, particle systems, and more. I have experience in Unity, Unreal, and even a custom game engine. I have experience with several coding languages, various graphic design programs, and can even use 3D modeling software to create prototypes. I’m always looking for opportunities to develop new skills.

ChatGPT Version:

Hey there, I'm Luke, a dedicated Level and UX designer with a strong foundation in crafting immersive gaming experiences. My specialty is crafting intricate level layouts and stage designs that feel at home in their environment. I prioritize creating stages that enhance the player's journey and blend seamlessly into the overall experience of the game.

My UX design philosophy prioritizes accessibility and user feedback. I take pride in guiding players seamlessly, so they never feel lost or confused. I believe UX is at its best when it’s invisible, while still maintaining a strong impact on the player. Crafting tutorials is where my passion for both level design and UX converge, guiding the player through new mechanics through careful level design.

Driven by a thirst for knowledge, I’m constantly seeking opportunities to expand my skill set. Whether it's mastering new tools or delving into different coding languages, I thrive on the challenge of acquiring new skills. I'm well-versed in graphic design programs and 3D modeling for prototyping. With proficiency in Unity, Unreal, and even custom engines, I bring versatility to the table.

In essence, I'm a problem solver at heart, constantly seeking out opportunities to refine my craft and make meaningful contributions to every game I work on.

* Strategist – Find solutions, big picture
* Philomath – Love learning, acquire knowledge
* Time keeper – planning, deadlines
* Optimist – find the good
* Problem solver – Find solutions, find flaws, can’t ignore issues

**Platforming Adventure**

**Project Hook**

Designing an entire platformer by myself

* Building levels as a full experience *Read More >*
* Planning for a tight deadline *Read More >*
* Developing skills in new fields *Read More >*

**Project Homepage**

Platforming Adventure is a solo project I created to practice level design. I wanted to lean into rapid iteration, focusing on the *Kishotenketsu* design philosophy popularized by the *Super Mario* games. This involves introducing a mechanic, developing it, and adding an unexpected twist. This is all done in a short amount of time, usually just a level or two, and then the mechanic is dropped to focus on something new.

**Building Levels as a Full Experience**

I believe Level Design is the most important part of a good platformer. It takes thorough planning to create levels that are each exciting and unique. If done properly, any level should be a fun experience on its own. Together, a group of great levels can make any game stand out.

**Objective**

My goal for this project was to build three levels, each focused on a different mechanic. Each level should be:

* Easy to pick up and play
* Short (1-3 minutes)
* Focused on a unique mechanic

Since I was building the entire game myself, I had some limitations. The unique mechanics for each level would also have to be programmed by me, and would cut into some of the level design time.

**Research**

My main goal for this project was to practice a form of level design called *Kishotenketsu*. In this style, a mechanic is introduced and developed at the start of a level. It then has some form of twist that uses the mechanic in new and unexpected ways. At the end, it returns to its simple form for a final challenge.

A video game with a game

Description automatically generated with medium confidence

I researched the *Kishotenketsu* design style through various videos and blogs. I also played levels from several different games to see how it was implemented. I looked into how mechanics were implemented and broken into smaller challenges. One thing I hadn’t considered was that there is a “cooldown” area between each challenge. This would be a safe place to collect some coins or power-ups, a checkpoint, or even just a straight path with no enemies. This gives the player some variety, so the entire level isn’t justthe core mechanic.

**Picking a Core Mechanic**

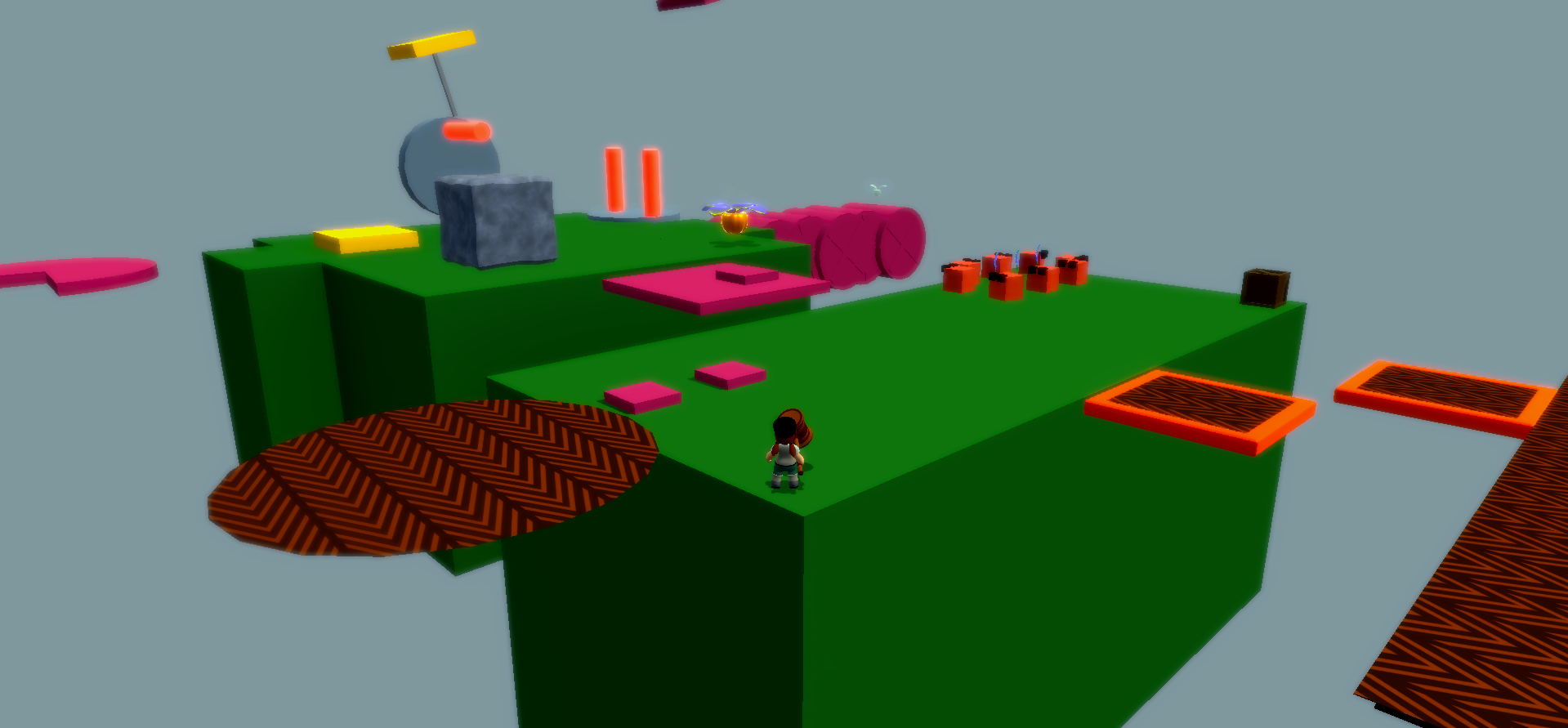
When I started designing the first level, my goal was to pick a mechanic to focus on. My first attempt was to feature breakable blocks that the player could destroy by attacking. When I tried sketching out some challenges, I discovered that the mechanic didn’t seem to stand on its own. I did more research to figure out why it didn’t work. I found out that players abilities are best used to accent the level’s mechanic, rather than being the mechanic themselves. For example, the dash ability in *Celeste* is present in every level, but no level is just about dashing. Rather, they each have their own unique features that work with the dash to let the player show off their skills. With this in mind, I wanted to find a new mechanic that would highlight the player controller.

A drawing of a game map

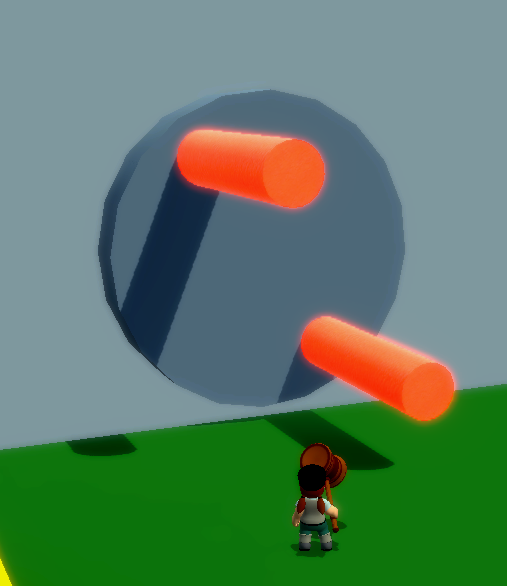
Description automatically generated

**Rapid Prototyping**

To find something fun to build my levels around, I created a test level with everything I could think of. I took simple scripts for moving and rotating platforms and made as many combinations as I could. Then, I ran around the level and played with each one to see what they played like.



I particularly liked the idea of rotating platforms with spikes on them, and started thinking about other ways I could build off that concept.



**Back to the Drawing Board**

Once I had a core mechanic, I went back to sketching out a level.

**Playtest Feedback**

I had many people play my level over the course of the project. There were many minor pieces of feedback that I implemented over the weeks, but one stands out in particular. I designed my level from a sidescroller perspective. Most playtesters, however, but the

**Planning for a tight deadline**

**Overview**

This project was built from scratch in just ten weeks. While the level design was my core focus, everything had to be built by me. It took a lot of planning to make sure the project stayed on schedule. Throughout the duration of the project, I had to change and cut many features to make sure the game would be finished in time. It’s far from what was originally planned, but I’m happy with how it turned out.

**Creating a Schedule**

I started by making a rough outline of what work I needed to have done by the end of each week. I broke the time into three categories: system, level, and polish. The first two weeks were dedicated to developing the player controller, adding enemies, and updating the camera. The bulk of the time was dedicated to designing the levels, with two weeks planned for each. The last two weeks were for cleaning up external features to make the game more appealing.

A screenshot of a computer screen

Description automatically generated

At the start of each week, I would look at my plan to see what I needed to do. I would then make a list of specific tasks that I need to accomplish that week. As I completed them, I marked each one to keep track of progress. I also tracked how long each task took so I would be able to plan better in future weeks.

A screenshot of a computer

Description automatically generated

**Unexpected Roadblocks**

I had planned to spend two weeks on each level initially. One week would be sketching, whiteboxing, and playtesting, and the other would be bug fixing, setdressing, and polish. When I started working on the first level, the sketching and whiteboxing phase **didn’t go as planned**. I ended up losing most of that week and starting the level over from scratch, which put me behind schedule. I was able to create a much stronger level, but knew I would still need more time to polish it.

**Adjusting the Plan**

At this point, I had a difficult decision to make. If I stuck to my original plan, I wouldn’t have time to polish the levels. Otherwise, I could make just one level, which would prevent me from practicing rapid iteration. It was a difficult decision, but I decided to cut down to just one level. While I had hoped to focus on building several levels with unique mechanics, I knew that I wouldn’t be able to create strong levels if I also had to program the mechanics too. I already had a working level select screen, so I could always add more levels in the future. This way, I could guarantee I reached a level of polish I was happy with.

**Finishing the Game**

Following the new plan, everything went smoothly through the level design process. Thanks to the extra time, I was able to **adapt to unexpected feedback** that came up during playtests.

**Developing skills in new fields**

Lorem

**Basil and the Isles of Spice**

**Project Hook**

Designing an entire platformer by myself

* Building levels as a full experience *Read More >*
* Planning for a tight deadline *Read More >*
* Developing skills in new fields *Read More >*

**Project Homepage**

Say something about how *YOU* work on a team

**Olympus Quest**

**Project Hook**

Creating a polished combat system

* Designing and implementing VFX for combat
* Creating a robust third person camera

**Project Homepage**

Olympus Quest is a VFX case study inspired by action RPG combat systems. This gave me an opportunity to develop new skills in creating shaders, animations, and code for a combat system. I also developed a new camera controller that follows the player dynamically, detects collision, and x-rays the player when they are behind an object.

**Designing and Implementing VFX for Combat**

My primary goal with this project was to create the VFX for a sword attack in a melee combat system. I was inspired by games like Final Fantasy VII and Kingdom Hearts 3 which have robust combat systems with a multitude of attacks and abilities. These types of Action RPG games feature flashy effects that bring basic attacks up to the next level. My goal was to make a basic three-hit combo in this style to practice creating visual effects.

**Deciding Where to Start**

This project proved to be quite the challenge, as I had never created this type of effect before and wasn’t sure where to begin. In the past, I’ve used trail renderers for this type of ability. However, this approach often leads to attacks looking choppy, and the trails can often warp or lose definition at certain camera angles.

After some research, I discovered that many games use 3D meshes that trace the path of the attacks, combined with shaders and particles to produce the desired effect. Once I knew how to implement the effects, I needed animations to apply them to.

**Animations Define the Combat**

I

**Creating the Effects**

I went through several iterations and tried multiple different art styles in order to create the sword trail. I tried using hand-painted textures and scrolling them along a curve that followed the blade. This worked, but didn’t match the art style of the game. My next attempt used a generic, donut-like curve rather than following the sword directly. This made the trail less polygonal, and the added thickness makes it visible from any camera angle. Finally, I created a custom shader that uses noise textures to generate the flow of the blade. I was happy with this result, but it still felt like it was lacking.

**Supplemental Effects**

While the slash effect looked exactly how I wanted it, the attacks still felt incomplete. I decided to do more research and looked into some games that had similar combat systems. I discovered that many games use particles in addition to the trail to add even more flare to the attacks. I created small particles that come off the trail, a ring effect that emanates from the center, a burst at the end where the blade stops, and a distortion wave that warps the surroundings slightly.

[IMG]

**Final Results**

With all of these effects added at once, the attack feels much more impactful. I then timed it to match the animations of the attacks and got my final result.

I added similar effects to the enemy when the attack hits. There are particles pulled in to the point of contact, a burst of light, and even a small chance of an addition slash across the screen for more variety. In the end, I’m happy with how this combat system turned out. The VFX bring an extra level of flare and polish that make the attacks feel more impactful and improve the game feel as a whole.

**Creating a Robust Third Person Camera**

Cameras are one of the most difficult parts of a game to get right. There are many different implementations from static to following the player. A bad camera can make it hard for the players to enjoy the game, and can even end up causing them to get lost or disoriented. I already experienced this issue before on a previous project, so I wanted to make sure I had a solid camera this time.

**Selecting a Camera Type**

The most basic way to implement a third person camera is to have it follow the player’s position with an offset. This approach is great for exploration focused games, racing games, dungeon crawlers, and many others. For action games, however, a more dynamic hands-on approach can be beneficial. My goal is to implement a camera that follows the player and rotates as they move, similar to those seen in games like *Super Mario Odyssey* and *Kingdom Hearts.*

**Camera Movement**

To replicate the movement, there are three states I need to consider. When the player is moving the camera stick, the camera should rotate around them no matter what. If they aren’t, I look at how the character is moving. If they are moving straight, I update the camera’s position, and if they’re moving to the side, I update its rotation. This creates an effect where the camera follows the player when they move away from it, but rotates to look at them if they walk around it rather than moving to the side.

**Camera Collision and Culling**

A common issue with 3D cameras is handling collision. It should never be possible for the player to be off-screen during gameplay, so the camera has to be able to adapt to different obstacles that could get in the way. A common approach is to pull the camera in any time the player would be behind a wall. While this works, its often jarring and can confuse the player. It can also easily be triggered by tiny pieces of geometry that don’t fully obscure the player. Instead, my preferred method is to render the player through the wall in an x-ray effect. This keeps the player visible at all times. This is only an option when the camera is fully on the other side of a wall however. If it is inside of a wall, zooming in is still the safest option to make sure no out-of-bounds geometry is visible.

**Preventing Player Clipping**

When the camera zooms in by a wall, it is often possible for it to clip into the character’s geometry. This causes some faces not to be rendered properly and can show through the character’s body.

One potential solution would be to simply limit how close the camera can get, but then the previous issue can occur again, allowing the camera to be inside of walls. Instead, the solution is to fade the player out when the camera gets close. I chose to use dithering to do this, as it’s the easiest option and has a unique look to it. With this, some percentage of the player’s pixels are not rendered when the camera is close, creating an effect that looks like transparency.

**Target Locking**

Everything so far is common camera controls for any game. However, I also wanted to add extra camera features that help accent the combat system. I created a lock on camera that overrides the basic movement, instead focusing on a point between the player and the targeted enemy. It maintains an angle offset to make sure both characters are on screen during combat, and provides a more cinematic view of the combat.

**Final Results**

There are many different features that go into creating a camera for a 3D adventure game. Movement, collision, and culling all have to be taken into account to ensure the player can easily navigate and keep track of their character while playing. When done properly, a good camera should be unnoticed by the player, allowing them to engage directly with the core gameplay.