Before starting, I imagined a fast-paced, polished 2D platformer with intricate pixel art and smooth mechanics, something close to Celeste or Shovel Knight. The reality? A charming but janky little game where a derpy capybara in a spacesuit waddles through procedurally generated dungeons. The scope shrank (goodbye, elaborate boss fights), the art simplifie, and some mechanics (like knockback) took way longer than expected. Yet, the core spirit, whimsical exploration and challenge, remained intact. I learned that a "finished" game is rarely perfect, but it is playable, and that's an achievement.

What I'd Do Differently
Start smaller. I wasted weeks over designing sprites before nailing core mechanics. Next time, I'd prototype with placeholder art first.

Learn Unity's Tilemap earlier. Hand-drawing levels was inefficient; using grid-based tools earlier would've saved hours.

Ask for feedback sooner. I hoarded my work until it felt "presentable," but classmates could've spotted issues (like floaty controls) way earlier.

Time Management & Resources
Tutorials were double-edged swords. Watching a 26-hour pixel art playlist helped, but I often fell into "research paralysis." Setting strict time limits for learning phases would've helped.

People were my best resource. Eden's feedback on pacing and France's warning about Roblox rules saved me from dead ends. I learned to swallow my pride and ask, "How would you fix this?"

Logging hours was eye-opening. Seeing "35 hours on sprites" forced me to prioritize. Next time, I'd use a project tracker (like Trello) from Day 1.

Recommend to Future Seniors?
Yes, but with caveats. If you love problem-solving and can embrace frustration, game dev teaches resilience. But if you want a stress-free project, pick something with fewer moving parts. Unity is powerful but relentless, bugs will make you question your sanity.

Showing an Expert?
I'd be nervous but eager. An expert might roast my spaghetti code or awkward animations, but their critique would reveal blind spots. For example, my "clever" knockback system might actually be inefficient, or my music could clash with gameplay cues. Growth hurts, but it's necessary.

Insights & Personal Gains

Technical: Mastered Unity's 2D pipeline, sprite slicing, and basic C#. Learned that good code isn't about being "smart", it's about being clear and fixable.

Creative: Pixel art is hard, but limitations breed creativity (e.g., two-legged Capy became a fun quirk).

Emotional: I'm prouder of the bugs I fixed than the features that worked on the first try.

The Stretch: Emotional & Intellectual Challenges

Physical: Eye strain from pixel art, wrist cramps from coding marathons.

Emotional: Nearly quit when the camera refused to follow Capy properly. Felt like an impostor when comparing my work to professional games.

Intellectual: Reverse-engineering Toby Fox's music taught me to analyze why things feel "right." Debugging knockback physics was like solving a puzzle with missing pieces.

. What I Learned About Myself

I'm stubborn in a good way. Even when Capy's animations looked like a toddler's doodles, I kept redrawing.

I undervalue planning. Jumping straight into sprites cost me time; next project, I'll storyboard first.

I thrive on small wins. The dopamine hit from fixing a single bug kept me going more than any grand vision.

This project was a crash course in humility, creativity, and grit. I didn't make the game I imagined, I made something better: proof that I can learn, adapt, and finish what I start. And hey, who doesn't want to play as a capybara in a spacesuit?