The game is meant to recreate the first dungeon of The Legend of Zelda (NES, 1986). We also added our custom mechanics and levels. You can press 4 to enter the levels, as it is separated from normal levels.

Development and Iterative Process

Except for the map, we need to code everything. The milestone is when every basic system was built, we built expandable components / scripts so that when the first of something (i.e. enemy, weapon) is done, we were really confident it is just a matter of time to get others complete. Also, essentially mechanics such as a solid door transition was built on that stage, then we did not touch it for the rest of the project. In alpha, we already built as well as polished everything so it was already quite authentic. In gold, we build the new mechanics and custom levels.

We used Jira to arrange work responsibilities. We can see clearly who was currently working on what, it eased some potential overriding files problems in merging branches. However, I admit that we did not log in time as we were expected to do. So we were not so clear about our progress if we left it for like 2 days.

To be honest, I did not seek a lot of playtesting except to let my roommate play (but he is not a gamer, got confused and thought the game was too difficult). A few feedback from course staff was more about recreating the original gameplay, and we fixed those issues.

I can proudly say that our code follows the “DRY” principle well. I wrote code in the best attempt to be reusable, clear, and having less dependency. Sometimes I get lost while I need to restructure the scripts and codes to make them more “DRY”. Overall it’s a good experience to learn “has a” logic, I find this super useful and interesting. I would recommend everyone who only knows “is a” hierarchy to learn it.

I found ChatGPT very helpful. It answers questions about codes really nicely. For example, I did not know Quaternion.identity existed. I asked ChatGPT for the idea, then it gave me the code I was supposed to use and it worked nicely. But ChatGPT is not good in some ways. For example, when I feel my code structure was messed up, I am not confident enough to throw everything into ChatGPT and hope it can help me solve it. And if I was able to express what I thought in language, then probably I could just change the scripts directly. ChatGPT also did badly for questions like “How can I replace multiple items by one prefab,” it lied.

Custom-Level Creative Process

The idea that I immediately came up with is Portal2. I have to admit that it is kind of tricky because adding the Portal2 function can be done in any game (implementing Portal 2 function in a platformer game will fit perfectly as well), and it's not the original idea. Also, it’s not that “specific” to this game. But the implementation process was still fun, the result was still good.

I think this idea came to my mind mainly due to a promising implementation difficulty and level variations. It is straightforward and fun. I did think about other alternatives such as time reverse in The Legend of Zelda: Tears of the Kingdom. But those ideas are either not straightforward to implement or a bit complicated to design levels.

On designing the levels, the main goal was to let the player understand what this weapon can do and really master it. Therefore I created a fairly difficult level that the player needs to shoot exactly when there's a block behind, I hope this will help the player understand it. My teammate created other levels where they show arrows and bombs can be teleported as well.

Future Improvement

We did a lot of things right in this project. Codes are as "DRY" as possible. We functionally remake the NES such that there's no obvious bugs (although there might be some that's not that obvious). The group work is perfect, we did our own parts with a high standard, and created components that are reliable (although we tend to change each other's code for our own needs). We gained the ability to figure something out, if we meet some problems in the code.

There are some things that went wrong. In the later stage, we did not start early. We did not use prefabs for doors and locks, which will be helpful. We did not make some scripts

reusable to the extent of “codebase” (it’s still based on other components), though it will be improved if I learn more on Unity and C#. I personally think that the sounds are messy, that sounds should be in a centralized sound management component. The animation is not very well managed, there is no animation on attack since I did not figure out how to add it.

I would say completeness is one of the keys to measure improvement. Also, the code reusability, clarity, and structure is an important measure of personal improvement.

I think my interaction with AI assistants is good enough, it is not too less or too much, and I can manage everything in control.