In Licht\\, the player is equipped with the Torch that interacts with receiving blocks to make a way. The player needs to combine the three states of the Torch to solve the puzzle.

(I don't answer all questions on the template 1 to 1, rather, some answers include responses to multiple questions.)

Creative Process

The initial idea is quite different from the implementation now. It was inspired by The Legend of Zelda: Tears of The Kingdom, for which Link is able to reverse the time of a certain object. I wanted to build a mechanism that when the object(s) is(are) hitted by Torch light, the object(s) can go reverse in time. However during the implementation I found this idea to be weird when playing. So I changed the game to its current state.

I don't think the story (if any) and mechanism of this game is deep enough to make a great impact on the player, but I hope players can have fun when they get their hands on the mechanism, gain some expertise via trying, and figure the puzzle out.

Development and the Iterative Process

For the first week, the main goal was to come up with an idea and experiment around that idea. My original two main objectives are: torch light system and time travel for some objects. However, when I really started to implement it, I found a lot of things to be

decided. For example, it will be in 2D or 3D, and if it is in 3D then shall I use First Person or Third Person. I spent a lot of time on that. After I chose to implement it in First Person, I started to realize that the original idea might not be good. But using the same system, I can do an activation system that does not travel back in time but really whatever I want on the object. And to keep it simple, I tried rotation and it basically is what the game is now. In addition, I spent a lot of time on making a menu system, as it involves a progress control.

After the gold spike comes the playtest session. It was conducted at campus, instructors and around 10 other students were able to play the game. Except for the technical thing like mouse sensitivity sucks, the reactions are heavily focused on false instruction text. And most players are not able to figure out how to combine states of the Torch in a short time.

For the second week, I focus mainly on improving the game via feedback. I summarized the reaction, thought about the problem, and translated them to issues that I was able to implement (so on Jira it's clearer than the first week). The new level 3 was added to clarify the usage of torch. And UI text is refined for visibility and clarity. Also, the fixed orientation of the torch that seemed to confuse everyone was changed so the orientation is now following the player. So it is less confusing.

For the time estimate, the first week I underestimated time needed, while for the second week, they were always overestimated. As mentioned earlier, in the first week there were always things to do and decisions to make that I ignored when I was planning. However, in the second week, I overestimated time cause I thought changing existing code would take a long time while it actually did not. A solid example would be the time for changing the Torch mechanism. I thought it would be difficult to decide and change, but actually it was only 1 or 2 lines of modification on the code.

Speaking of DRY, the code is really as DRY as possible. I did stray from the usage of Pub/Sub, since functions can be done by either Pub/Sub or the conventional methods. But after I sort of thought it through and got used to it (*To be fair, it was not I got used to it, but it demonstrated its power when the design scaled up*), I found Pub/Sub really helpful in keeping code DRY. The downside is maybe an imbalance in time: one of the consequences of implementing Pub/Sub is that it would take a lot of time to build it up. Obviously it is good for development afterwards (I overestimated the time to change codes), but for such a small project, it kind of is not worth it. In addition, it stopped me from experimenting with features, as the sunk cost is too large.

I found AI assistants useful in a way that it can help me to reinvent the wheel. The FPS control is written by AI. Also, AI did help me figure out some exact coding problems. But its power was limited and it was not enough to guide me “structure wise” (for example, it

could not teach me how to design Pub/Sub structure). Sometimes it gives seemingly correct wrong solutions as well.

Future Improvements

A lot of things were done right, including using the “interesting / meaningful decisions” as the framework to guide my design DRY code, the use of Pub/Sub, the use of Jira in the second week, overall planning and project management, receiving feedback and using that to identify problems and improve.

But things can go wrong. I think I struggled in coming up with an idea. And even though I did have an idea, I found it was not what I thought it would be, so changes were made and it was less fun. Another big issue (to be fair it was not completely my fault) was time management. For the second week I only spent 3 or 4 days on it, since it was the midterm week. And I was supposed to start early.

Now I fully contextualize the lecture that a good game shall have “meaningful / interesting decisions,” “guidance,” and “juice.” In the final project I will access the project in all three directions. And in terms of project management, I think I’m prepared to make clear goals / issues and time management should be fine.

The use of AI assistants is fine at this stage (I hate copy and paste code given, but for FPS control it shall be just fine, except making a game that is strongly related to the

camera). I will keep myself from over-using AI assistants and take control of my / our own project.