Our first and second weeks were unproductive. I think part of our inefficiency came from our lack of good communication. Our roles should have been better outlined from the start, and we should have had more group meetings in that first and second week so that teammates could show what progress they had made. Our first full team meeting did not occur until the Sunday before the project was due.

Our group had difficulty with the best practice of working with multiple branches and merging our code in a timely manner. Often throughout the first and second week our attempts to merge code into the main, left the main in a broken state. Among other difficulties using git, our group could have done better to communicate these issues. The silver lining is the experience we have gained. Our protocol for merging to the main branch is as follows...

1. Once a branch is in a working state, merge Main onto that branch. Do NOT pull the branch onto Main.
2. Once the branch is fully refactored to have all the functionality of Main and the new branch, replace Main with the branch. Do NOT leave the Main in a dysfunctional state.

I am sure our protocol will continue to develop as we move forward.

- Personal Comments -

**Cody:** I’m working to create the enemy foundation for the game. Originally, I ran into serious issues surrounding my content.mgcb folder, and then issues with my .Net core. These issues prevented me from testing my code for a significant portion of Sprint2. However, I was able to get past those issues and continue to work through them. My enemy design is to use a texture manager to load all of my enemies in, while using an enemy manager to cycle through my enemies along with other functionality. I’m using abstract classes for both my enemy and my enemy states. I plan on utilizing my enemy states more specifically once I’m able to integrate with my team’s code after sprint2. With an abstract class for enemy, I then implement each enemy on top of it in their own separate class.

**Elijah:** I feel as though I did a good job getting a good foundation for my Block classes early on. This allowed me to get feedback and iterate on my design. Originally, I thought to use just one class for the Blocks and differentiate functionality using enums. After feedback, I had time to iterate on this design and make multiple classes. There are still remnants of the previous design but considering the classes do not have anything differentiating them yet – no functionality other than showing up on screen – I don’t mind this and can easily be fixed later. Looking ahead, I will have to adjust my approach to constructing the blocks. Currently I create all the blocks at the start and only draw/update them one at a time. I need to improve my skills when merging branches and cloning other peoples branches.

**Trung:** I am responsible for making the item classes. Initially, I thought of implementing Item classes like Sprite classes. However, I can make a few Sprite classes that other courses can use to reduce boilerplate codes. So, I made a few Sprite classes that can be considered templates, and if a class needs additional functionality, it can be implemented in my classes. Then, I made some item classes that used those Sprite classes, which worked well. I also use a factory to manage my Items so changes can be made in only one place. I can refactor my code better, and there are improvements I could make. I will try my best for the next Sprint and engage more with our team.

**Larry:** I worked on the Link Class with Yuchen, including state transitions and player movement. Our game1.cs was a bit clustered when it comes to the keyboard controllers, we need to modify it and use the separate controller class in future sprints. I also fixed issues within the content.mgcb file to properly add the sprite sheets. Initially, we encountered errors when trying to pull from other people's content folder, but I created a new mono game project and manually added in the new content. We were able to test our code more efficiently with working sprite sheets, which are individually extracted to support smoother frame transitions. I added an item.cs file in the Link folder to solve the Item inventory not found issue in Yuchen’s game1.cs. I need to improve my time management abilities and communicate better with my teammates in future sprints.

**David:** I worked on Keyboard Input and Controllers, as well as Commands. Overall, for the original code the classes worked well, they felt well organized and understandable. I made changes to the game class as needed to keep it from getting clustered and cleaned the command classes, so each one was simple to understand as I added my teammates functionality to them. In the end there was a pretty large disconnect and some of the classes we were given either overwrote changes I had made, or implemented their own Keyboard input or commands. We had ALOT of GitHub issues that were pushed to main would conflict heavily with written code or downright delete others work. Especially the game class. In the future we’re setting stricter rules for using the repository, such as actually checking out the main branch and implementing your code off that and then pushing rather than dragging their project folder into main.

**Yuchen:** For this sprint, I worked with Larry on implementing and refining the Link Class, including handling state transitions (walking, attacking, taking damage) and ensuring Link’s movement felt responsive. We encountered various merger conflicts early on, particularly when integrating new sprite sheets and content into our project, but we managed to resolve them by carefully coordinating branch mergers and updating our content.mgcb files. I also helped address boundary issues so that Link would not walk off the screen. One area I want to improve on is communication and time management; we sometimes found ourselves rushing near the deadline. In the next sprint, I plan to work more closely with the team from the start and review each other’s branches more frequently to prevent major merge problems.