Date	Content	Responsability
17/11/2025	Meet to choose project and discuss initial attack plan	Everyone
19/11/2025	Meet to plan the flow of the project, elements needed, and general attack plan	Everyone
20/11/2025	Finalize UML and divide responsabilities	Everyone
23/11/2025	Set up project and group code for 2 hours, making sure polymorphism work as expected	Game, main, and Display: Josh Board and Player: Taim Level and Block: Linh
24/11/2025	Finish up a rough version of our section on our own time while keeping constant communication	Game, main, and Display: Josh Board and Player: Taim Level and Block: Linh
25/11/2025	Meet and group code for 2 hours, discussing what was and wasn't working, and make changes to system if needed Check point: - Block moves, rotates, and drop as expected - Level 0 work as expected - Basic command interpreter is completed - Score, level, and other elements of the board displays correctly	Everyone

26/11/2025	Coding session to implement line clearing logic, other levels and special effects Keep constant communication so the new parts fit well into existing system	Other levels: Josh Line clearing logic: Taim Special effects: Linh
27/11/2025	Work on the graphic display using XWindow class Check point: - Everything from the first check point - Game should function as expected with all levels and special effects with text displaying the correct colors for each block	Everyone
28/11/2025	Finish up everything, making sure that the program compiles and runs exactly as expected Discuss bonus features (such as sound) if there is any time left	Everyone

Questions:

1. **How could you design your system (or modify your existing design) to allow for some generated blocks to disappear from the screen if not cleared before 10 more blocks have fallen? Could the generation of such blocks be easily confined to more advanced levels?**

   We can add a field in Block that records what turn it was generated on. The turn counter can be kept in Board, which increases when a Block is dropped. After a drop is facilitated, Board would loop through the board, checking each block to see if the turn it was generated on was 10 or more turns ago. If it is, it retrieve the location of this block and clears it off the board. If it's not then nothing happens.

   This can be confine to more advanced levels by only running the search mention above if Board is at that level.

2. **How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?**
   We can use the Factory Method Pattern on Level, which means we made Level into a class, with each level is a subclass inheriting Level. With this, we can easily add new levels by creating a new subclass under Level. This new level may trigger different kind of block generations or other effects if desire. If a new level is added, we only need to recompile Level
3. **How could you design your program to allow for multiple effects to applied simultaneously? What if we invented more kinds of effects? Can you prevent your program from having one else-branch for every possible combination?**
   We can allow multiple effects to applied simultaneously by having each effect affect different part of the game's functionality. For instance, if blind is applied, only the board rendering needs to be notified. We keep a field call "howHeavy", which would increase if the player chooses higher levels, and further increase if the heavy effect is applied, therefor this effect simply changes one field within Level. Lastly, with the force effect, we can just interrupt the current block field within Board, changing it to the desire block. This effect only affect this field and nothing else. As seen, none of the effect changes the same function or fields in our program, therefor can be applied simultaneously
   This works since we only have 3 effects and this is the most efficient way. However, a better design would be making Decorator classes. Specifically, effects that affect the board, such as blind, would be under Board Decorator class. Effects that affect the block generation or movement, such as heavy and force, would be under Level Decorator class. Finally, effects that affect the actual block being generated would be under Block Decorator class.
   If we invent more kinds of effects, we can follow Decorator pattern and add more Decorator classes accordingly.
4. **How could you design your system to accommodate the addition of new command names, or changes to existing command names, with minimal changes to source and minimal recompilation? (We acknowledge, of course, that adding a new command probably means adding a new feature, which can mean adding a non-trivial amount of code.) How difficult would it be to adapt your system to support a command whereby a user could rename existing commands (e.g. something like rename counterclockwise cc)? How might you support a "macro" language, which would**

allow you to give a name to a sequence of commands? Keep in mind the effect that all of these features would have on the available shortcuts for existing command names.

We could use a config file that maps commands' strings to their implementation. The CommandCenter would load the config file at run time into a map<string, Command*> , so changing a command just needs to us to edit the config file and not recompile code. To rename commands, we could add an alias map<string, string> to the CommandCenter, which would translate user-defined names to commands. When we process for user inputs, we would check the alias map first then look up the real command. To implement macros, we would create a MacroCommand subclass of Command, which would store a vector of command pointers and execute them sequentially. When users define a macro, we store it as a new entry in the command map vector.

To implement shortcuts, when users type input, the CommandCenter would check if the input matches the beginning of any name. We would do this by looping through command names in the command map and comparing characters using array indexing to find matches.