j37shen Family Plan

Person	Task	Status	Goal	Due Date
Both	Project planning: ● UML Diagram ● Breakdown of Project	Not started	Mar 25, 202...	Mar 28, 20...
Maria	Set up observer & text display classes; set up block classes: ● Observer Class (.h) ● TextDisplay Class (.h) ● Cell (.h, .cc) ● SBlock, ZBlock, TBlock (.h) ● Implementation for block class (TBlock .cc)	Not started	Mar 25, 202...	Apr 5, 2024...
Joy	Set up block classes: ● Block Class (.h) ● OBlock Class (.h) ● IBlock, JBlock, LBlock, and OBlock (.h) ● Start implementation for block class (specifically O block.cc and IBlock.cc)	Not started		
Maria	Start implementation for block classes: ● SBlock, Zblock (.cc) Implement: ● notify() for Cell observers ● attach(Observers o) for Cell observers Test: ● SBlock, Zblock, TBlock	Not started	Mar 26, 202...	Apr 5, 2024...
Joy	Finish implementation for block classes: ● JBlock, LBlock (.cc) Test: ● OBlock, LBlock, IBlock, JBlock ● Makefile	Not started		
Maria	Begin the Board interface and implementation: ● moveBlock() ● Counter for cells in block Work on graphics: ● Custom colour map	Not started	Mar 27, 202...	Apr 5, 2024...

Joy	Begin the Board and implementation: - Board (.h) - init() - clearBoard() - placeBlock() - dropBlock() Review our UML and Plan and submit	Not started		
Maria	Continue implementation for Board: - dropBlock() Test: - Placing/moving/dropping a block	Not started	Mar 31, 202…	Apr 5, 2024…
Joy	Continue implementation for Board: - checkLineClear() - lineClear() - updateScore() and isLose() Test: - Initializing board, clearing board - Updating scoring and scoring in general	Not started		
Maria	Finish implementation of TextDisplay: - TextDisplay (.cc) Begin implementation of the graphics display: - GraphicsDisplay (.h, .cc)	Not started	Apr 1, 2024 …	Apr 5, 2024…
Joy	Begin player class: - validMove() - Understanding random generator and implementing new move for Level 0, Level 1, Level 2	Not started		
Both	Implementation of levels: - Level 3 and Level 4	Not started	Apr 2, 2024 …	Apr 5, 2024…
Maria	Begin Making Test Suite: - Tests Start special actions: - blind	Not started	Apr 3, 2024 …	Apr 5, 2024…
Joy	Begin documentation: - explain changes in UML and changes in the plan - answering questions Add to testing suite: - Tests	Not started		

Both	Implementation special action: ● force Testing: ● Run test suite	Not started		
Maria	Implementation special action: ● blind	Not started	Apr 4, 2024 ...	Apr 5, 2024...
Joy	Implementation special action: ● heavy	Not started		

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?**
   To implement the disappearance of these blocks, we could easily include a counter on each cell that will increment with each block that is played. Once the block counter reaches a certain number, for example 10, we know that the block has not disappeared before 10 moves and can make them disappear using setType() and unfill the specific cells. This will then notify and update all observers such as TextDisplay and GraphicDisplay so that the block disappears visually. All other functionalities such as lineClear should remain the same. To confine such blocks to more advanced levels, this block and specific rules could be implemented and made in our factory design method. The generation of blocks and cells would include a counter field and an "expiry time" such as ten blocks which could be easily updated per level.

2. **How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?**
   To accommodate for the possibility of introducing additional levels into our system, we could add to our implementation of the levels via the Factory design method. Currently, our levels are implemented using a Factory design method, whose level will control the difficulty of the game of each player and generate blocks given the specific rules of the level. The Factory design method provides more flexibility to the program in terms of making new levels and minimizes recompilation. So, to add more levels, we can include more subclasses of level that will create blocks of that level.

3. **How could you design your program to allow for multiple effects to be applied simultaneously? What if we invented more kinds of effects? Can you prevent your program from having one else-branch for every possible combination?**
   The Decorator design pattern allows us to chain together or combine multiple different effects such as blind, heavy, and force without the need for an else-branch for every possible combination. We would design our program to have an abstract class Decorator with several decorators for the Board class. In order to add more effects, we would add more decorators.

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.**

To make changes to the number of commands or commands themselves, we would create a list (vector) of the commands (strings) to be able to easily access commands. This would allow us or the user to change the command names or add a new command (emplace) into the existing list of commands. These commands could also be organized alphabetically to make finding the commands easier.

To support a macro language, we could create another array with names of shortcuts, with each shortcut calling a sequence of commands.

If there are multiple commands and shortcuts, it may be better to store them in a dictionary rather than an array. Adding keys whose values corresponds to their commands, or if they are a name for a sequence of commands, their values would correspond to an array of commands.