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?**

   In our opinion, the best solution would be to have a "lives" integer counter field for each block. This field would be initiated at 0 as soon as the block is dropped into place. Each occurence of a "dropped" block will decrement the lives counter of each block (already in the board) by one. When doing so, there will be a check for if the given "lives" argument is equal to 10. If so, the block has not been cleared before 10 more blocks have fallen and that block will disappear, meaning we will clear that block. This implementation can be easily confined to more advanced levels since the number of blocks being generated and scoring system is what will be the key difference. This does not change the fact that the generated block will have a "lives" counter and each block will have its "lives" counter increment by one each turn. We do have to take notice of how we count the score for each disappearing block as this will differ based on the level. This will be accounted for by passing an argument representing the level to the clearing method.

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, which will allow for the Block Creation to be delegated to the Level subclasses. As a result, we can easily add a new Level subclass to create its own generate block method and functionality instead of adding logic for each level in one Block Creation method. We can also use a Makefile and separate compilation to allow for minimum recompilation of .cc files when a previous executable is updated.

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?**

   One way to allow for multiple effects is to use a decorator pattern. Instead of having an else-branch for each combination, the decorator for each effect can be either applied or not applied, and even applied multiple times if desired. Adding another effect means adding a new decorator object that can be applied to create that new effect. To implement this decorator pattern, an abstract base class will be added that is inherited by the concrete "game" class and an abstract "decorator" class, which has a pointer to the abstract base class. Then, concrete decorators inherit from the "decorator" class and apply the special effect to the "game" class using the pointer.

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 facilitate the ability to add new commands, we confine the logic for taking in commands to a "command interpreter" class. Therefore, to add in a new command, we only need to add a new check for that command in the command interpreter class (and implement the command as needed in the rest of the files). Then, we only need to recompile the command interpreter class using the Makefile. To allow for the renaming of commands, we can use a array or vector of strings to store the command names and map each command name to a sequence of commands. Then, to rename a command, the command name in the array of strings should be renamed. Then, to allow shortcuts to still work, we cannot hard code "lef" for left for example but need to dynamically check the command names. When a shortcut is typed, for example "le", we check all the command names in the array to see which ones match. If a single command name matches, that command is executed, else no command is executed.

SCHEDULE

Date	**Feature Implementation**	**Person**
Nov 14 - 16	UML Diagram	All
Nov 17	Create Git repository	Jeffrey
Nov 17 - 19	Blocks ● Implement seven types of blocks ● Rotate blocks ● Move blocks ● Print blocks ● Command interpreter → `clockwise`, `counterclockwise`	Jayson
Nov 17 - 19	Board ● Reserve rows ● Know when board is full (game over) ● Know when row is complete ● Print board ● Clear board and start new game ● Command interpreter → `left`, `right`, `down`, `drop`	Danny
Nov 20	Text display ● Print level and score ● Print next block ● Command interpreter → `restart`	Jeffrey
Nov 20 - 24	Difficulty levels ● Read in sequence from external file (0) ● Random number generator (1, 2) ● “Heavy” blocks (3) ● “Dead” blocks (4) ● Command interpreter → `levelup`, `leveldown`, `norandom file`,	Jayson

	`random` • CLI → `-seed`, `-startlevel`	
Nov 22 - 25	Special actions • Implement interaction between two boards • Special actions → `blind`, `heavy`, `force` • CLI → `-scriptfile1`, `-scriptfile2`	Danny

Nov 24 - 26	Support for additional commands
	• Command interpreter → `sequence file`, `I/J/L`
	  ○ Autofill commands → `lef = left`
	  ○ Multiplier prefix → `3ri = right * 3`
		Jeffrey

Nov 26	Scoring • Scoring system • High scores	Jayson
Nov 26 - Nov 30	Graphical display • Assign each block a colour • Print blocks and board • CLI → `-text`	Danny
Nov 30 - Dec 2	Extra features	All (requires meeting)
Nov 30 - Dec 2	Final design document	All
Nov 30 - Dec 2	QA testing	All
Dec 2	Finish!	