Answers to questions

Question 1

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?
   - Answer: Each cell block has a unique ID corresponding to the tetromino it came from. It is incremented by one every time a new tetromino is generated. Once the ID of the new tetromino is a certain number above the ID of a cell already in the board, it would destroy that block.
2. Could the generation of such blocks be easily confined to more advanced levels?
   - Answer: A level with this rule can be created and code that removes tetromino blocks based on this can be written in the Game singleton. It would send a command to the Board module every time blocks need to be deleted.

Question 2

1. How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?
   - Answer: You would want to have a "Level" class, with each of the different levels as objects of the class. The class has parameters including the probabilities, whether or not it is random, whether or not blocks are heavy, etc. Somewhere in the Game object there would be a vector of the different level objects, where the key is the level number. To add a new level, make a new Level object, set its parameters, and add it to the vector.

Question 3

1. 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 toadapt your system to support a command whereby a user could rename existing commands (e.g. something like `rename counterclockwise cc`)?
   - Answer: Changes to existing command names can be hanled by the command parser which would have a special command which would make the necessary changes in the map of commands. That is, if you enter `rename counterclockwise cc`, the key that maps `counterclockwise` to a function would change to `cc`.
2. 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.
   - Answer: To make shortcuts to chains of commands, you would want a map. The key would be the shortcut string, and the value would be the string of commands that the shortcut corresponds to. When the user enters a new shortcut and the commands it represents, it checks to see if the desired shortcut does not already exist as a command, and then if it doesn't, adds it to the map. If the command parser detects that you've entered one of the shortcut strings, it executes the commands mapped to it.

Timeline

November 25

1. Draw the board textually
2. Board class
3. Write the Block class
4. Commands: drop, left, right, down
5. Commands: clockwise, counterclockwise
6. Commands: I, J, L, 0, S, Z, T
7. Write level 0 with scoring
8. CLI Option: --scriptfile xxxx

Responsibilities:

Felix: 2, 4, 5

Oscar: 6, 8

Alex: 1, 3, 7

November 26

1. Command: sequence file
2. Implement function to check whether a row is filled upon drop, which prompts the blocks in the rows above to drop by 1
3. Command: levelup and leveldown
4. Write level 1
5. Write level 2

Responsibilities:

Felix: 1, 2

Oscar: 3, 4

Alex: 5

November 27

1. Write level 3
2. Write level 4
3. Command: norandom file
4. Command: random
5. CLI Option: -seed xxx
6. CLI Option: -startlevel n

Responsibilities:

Felix: 2, 3

Oscar: 1, 4

Alex: 5, 6

November 28

1. Command: restart
2. Multiplier prefix for commands
3. Understand shortened commands (i.e. lef is enough to distinguish left from levelup)

Responsibilities:

Felix: 1, 3 (work together)

Oscar: 3

Alex: 2

November 29

1. Double check leaks (before making UI)
2. Create graphical view
3. CLI Option: -text
4. Command: hint

Responsibilities:

Felix: 1, 3

Oscar: 4

Alex: 2, 4

November 30

1. Double check all tests
2. Double check we are using smart pointers everywhere possible
3. Double check leaks again
4. Bug fixes

People responsible: Everyone

December 1

1. Bug fixes
2. UML diagram
3. Report writing

   Responsible: Everyone

December 2

1. Report writing

December 3

1. Report writing

December 4

1. Practise demo