CS246 Spring 2021 Final Project
Biquadris
Plan of Attack

Amy Hwang
John Yoon
Liam Mesrefoglu

<u>Breakdown of the Project</u>

Preliminary steps:

Our group has decided to use git.uwaterloo.ca to organize our code files together in a private repository. Moreover, we have consistently met up on Social Media Platforms (such as Discord) to share our ideas in terms of designing the program.

Biquadris Breakdown in Details:

Set up the game board

1) Create a tetris-like grid.
2) Design the user interface.
3) Create a way to add a single block for testing purposes.

Block movements

4) Create block abstract classes.
5) Create block_x classes.
6) Create a way to add blocks into the grid for testing purposes.
7) Add left and right movement to the blocks.
8) Limit the movement of the blocks so it does not go out of the border (left and right.)
9) Add downwards movement to the blocks and the down command.
10) Add a way to stop the block when it cannot move down anymore.
11) Add rotation to the blocks.
12) Add limitations to rotation.
13) Enhance Text Display for testing purposes.

Row Controls

14) Implement the row disappearing. (which requires the implementation of Observer and Subject classes.)
15) If 2+ rows are destroyed, trigger an event that does not do anything for now (for later)
16) Add color to the blocks. (which requires the use of Xwindow)
17) Implement auto spawning of the blocks.

Level Variations

18) Make the program read from a file to know which blocks to spawn.
19) Implement level 0 fully.
20) Implement random falling blocks.
21) Implement level 1 fully.
22) Implement level 2 fully.
23) Implement automatic falling (heavy) with level 3.
24) Create a hard block after x amount of moves.
25) Implement level 4 fully.

Special Actions

26) Special actions, implement blind.
27) Special actions, implement heavy.
28) Special actions, implement force.

Core Commands

29) Implement scoring, seed, and level scales for the scores.
30) Implement the restart command (Clearing the memory for a new game).
31) Ability to use a command multiple times.
32) Ability to find a command that differs from 2 commands (e.g.: rest instead of restart).
33) Finishing the game.
34) Polish Graphics for the game.
35) Implement command-line interface.
36) Polish the game with debugging. Find if there are any edge cases.

Bonuses

37) Discuss if there are any ideas for extra features.
38) If not, implement smart pointers.

Design Patterns which will be used in the program

- Decorator Pattern for Special Actions
- Observer Pattern for both Displays
- Abstract Factory Pattern for Blocks
- Factory Pattern for Levels

Members' Responsibilities

Members	**Tasks**
Amy H.	Classes Blocks, UserInterface, XWindow, TextDisplay, and Terminal / Documentations
John Y.	Classes Grid, Levels, Tetris, TetrisUtil, and GraphicDisplay / Graphics, Command-line Interface
Liam M.	Classes Cell, Grid, Observer, Subject, UserInterface, and Special Actions / Test Cases (find bugs/mistakes)

Although we have decided to distribute our responsibilities as such, we will naturally help each other during the process. Moreover, this is not a solid distribution of work. Everyone will work together to polish the project in the end.

Estimated Completion Dates

Task	**Date**
First meeting for setting up git repos and UML diagram	07/25/2021
UML diagram, Plans of Attack	07/26/2021
Finalize dd1 documents	07/27/2021
Hand in dd1 documents	**07/28/2021**
	07/29/2021
Set up the game board and Block Movements	07/30/2021
Block Movements	07/31/2021
Row Controls	08/01/2021
Level Variations	08/02/2021
Special Actions	08/03/2021
	08/04/2021
Core Commands - Scores, Restart	08/05/2021
Core Commands - Other	08/06/2021
Graphics and Command-line Interface	08/07/2021
Debugging/Edge Cases and Finish	**08/08/2021**
Discuss for possible bonus features	08/09/2021
Work on bonus features	08/10/2021
Finalize UML diagram	08/11/2021
Finalize dd2 documents	08/12/2021
Final Submission on 11:59 PM	**08/13/2021**

Answers to Given Questions

Blocks

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 allow blocks to disappear, the most intuitive and easiest method we could think of is to implement the integer counter field in block classes. This counter will then increase as we create blocks and it will be resetted to 0 when it reaches 10 to clear the blocks or the blocks have been cleared (whichever comes first). To see whether the blocks have been cleared or not, we would have to implement a method or boolean in our Tetris class to have this counter be "notified" towards the change. Moreover, to have such blocks be confined to different levels, we must implement another field, such as integer level field so that we could keep the blocks consistent with a certain desired level.

Next Blocks
How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?

- For minimum recompilation, it is best to create the abstract class Level and create subclasses such as Level_x (i.e. Level_4) which inherits the class Level. With this inheritance method, we would only have to create additional class Level_x and override the common Level methods and add any additional methods we need for specific levels.

Special Actions
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?

- To apply several effects at the same time, we could implement them using decorator patterns. For Conway's life game, one of the questions from our previous assignment, we used decorator patterns to accommodate multiple rules for certain cells. It wraps up the cell with the rules. Similarly, we can wrap up our Tetris of other players with the actions and we can simply add the subclass under the class Decorator if we invent more kinds of effects. The use of decorator will naturally prevent from having else-branch.

Command Interpreter
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 accommodate new command names, we could create a class Terminal which will control all the commands coming from the user. By having a class Terminal, we only have to compile the least file since Terminal would only add one method that corresponds to the new command. It will not be too difficult to adapt renaming since we could just replace the "old" command name to desired one. We have used the controller class in previous assignments, and our Terminal class functions the same way. With this separate class, we create low coupling.