SWEN30006 Software Modelling and Design
Project 1: Tetris Madness

Yifan Cheng 1081430
Yutong Chen 1132677
Mingyue Jiang 1126918

1 Part 1 - Analysis current design

From the given class diagram in the project specification, the biggest problem is there exists large amount of duplicate code in shape classes I,J,L,O,S,T,Z and Tetris class. This will increase the complexity of later maintenance. For example, if you change rotate() function in shape class like I, you need to change seven classes. Likewise, too many methods and attributes are all gathered in class "Tetris", according to GRASP, the class "Tetris" has low cohesion and it is also too complex which will decrease the comprehension of design, simplify maintainability and enhancement. Since we need to extend the tetris by add medium and madness difficulty level, but there is no difficulty level class like shape class. That will also be a problem during the extension.

Besides above, the moveBlock() function in Tetris class contains too much redundant work which reduce the readability and that can be simplified.

2 Part 2 - Proposed new design

New parent class for the block is supposed to be created to reduce redundant codes such that different shapes can be created as child class to inherit from the parent class and simply overwrite code where own unique features should be applied. Based on this, we create a parent class called Block for the seven shapes classes. Since the original code let seven shapes classes extends Actor class. So, we adopt multiple layer extend which let Block extends Actor and then let seven shapes classes extends Block class.

The new design will also benefit the future extension if new shapes of blocks are required to add to the game. In this case, only the unique features of new shapes need to be typed into the program. That is, according to GRASP, the polymorphism which is useful for speculative "future-proofing" against an unknown variation. Due to the similar reason, We create medium and madness classes that both of them extends the Tetris class. please refer 4.2.

To reduce the redundant code of moveBlock() function, we can abandon the if else and just use the switch cases. To ensure each shape class can be moved, we only need to change the forced conversion type from shape class to Block, e.g. (Block) currentBlock).rotate();.

3 Part 3 - Proposed design of extended version

3.1 Design for new difficulty levels

It has already been mentioned that class Tetris is too complex with low cohesion so it is not appropriate to directly add both the behaviours of medium and madness level directly to the class Tetris. For new design, each difficulty level is created as child class which inherit from the class Tetris. Both of the child classes have the methods to decide the speed of a block when the createRandomTetrisBlock is called, while madness class also has the method to disable the rotation of the created blocks.

The design will bring convenience while both applying changes to the existed difficulty levels and adding new difficulty levels to the game. According to GRASP, reassigning responsibilities helps the class Tetris have higher cohesion which ease of comprehension of the design, simplify maintainability and enhancement, and lower coupling which supports the design of classes that are more independent and reducing the impact of change.

3.2 Design for gameplay statistics

A new class StatsRecorder has been created dedicated for statistic recording. This class consist of four int variables round, score, totalScore, averageScore, one String variable difficulty and a Hashtable blockRecord with key being name of the block and value tracks the number of block occurrences in the current round of game.

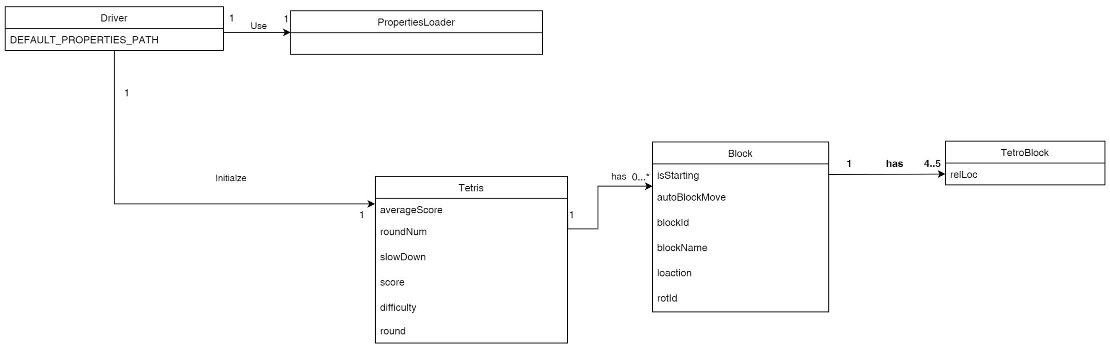
Every time Tetris Madness is opened by the player, an instance of StatsRecorder will be created. Each time a new round of game is started by pressing the Start button, the HashTable will be initialized to its original state, with all values set to zero. Each time a block is created through createRandomTetrisBlock() method, corresponding block count in the HashTable will be updated. The score variable of StatsRecorder updates every time the score variable under Tetris class is updated, which is when a line is cleared from the grid. The totalScore variable also updates every time a line is cleared from the grid, it is then used to determine the value of averageScore, which is calculated through totalScore / round.

When the Start button is pressed by the player, the statistics of gameplay from previous round will output to the "Statistics.txt" file with specified format. In order to update average score per round on the second line of the file, recent round statistics will append to the end first and the whole file will be read and saved to a String variable temporarily, with the line containing average score being altered during the process. We then re-write the altered String back into the file.

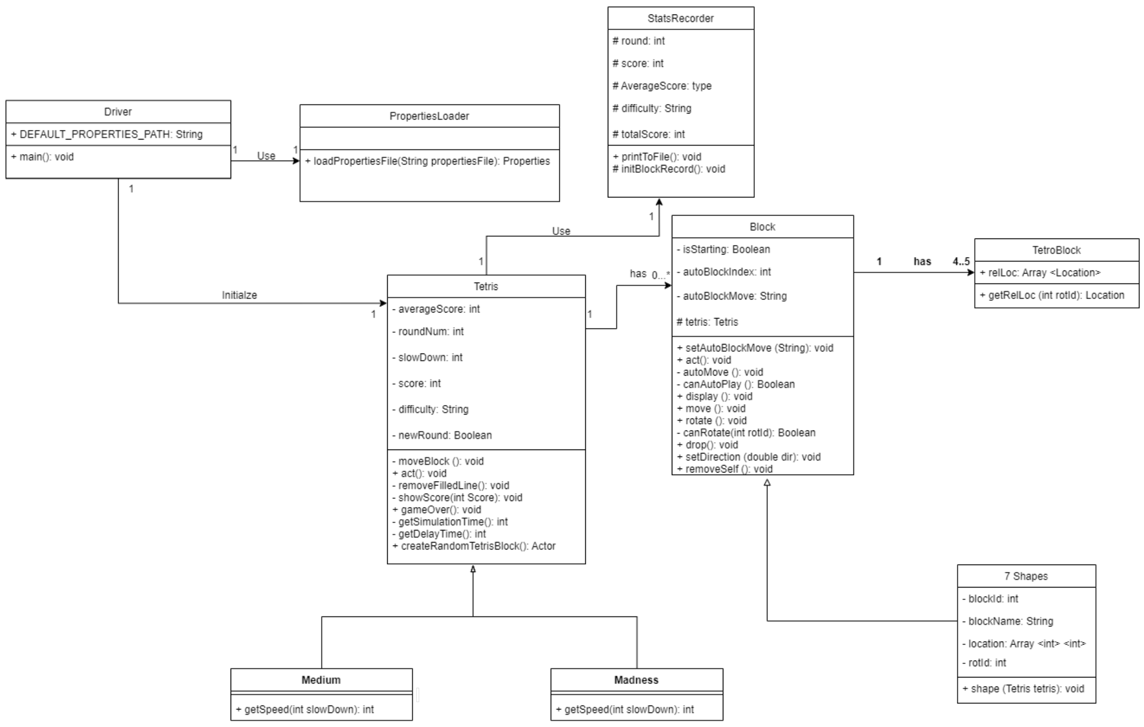
It is worth noting that when the program runs in auto mode, the statistics of the current round of game will be recorded and output to the txt file when gameOver() method is called.

4 Part 4 - Software Models

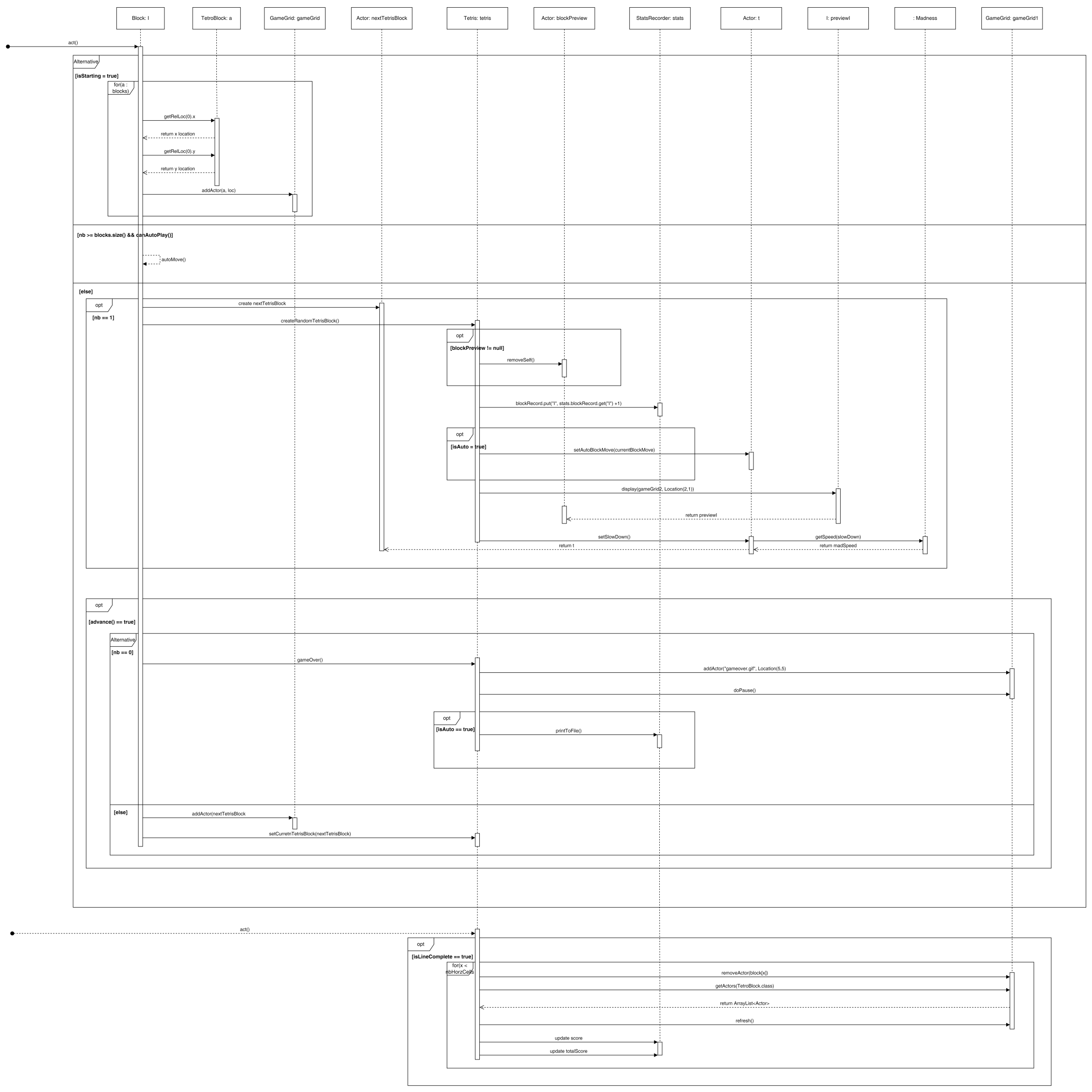
4.1 Domain class diagram



4.2 Design class diagram



4.3 Design Sequence Diagram for one falling shape



Block: I
TetroBlock: a
GameGrid: gameGrid
Actor: nextTetrisBlock
Tetris: tetris
Actor: blockPreview
StatsRecorder: stats
Actor: t
I: previewI
: Madness
GameGrid: gameGrid1
act()
Alternative
[isStarting = true]
for(a : blocks)
getRelLoc(0).x
return x location
getRelLoc(0).y
return y location
addActor(a, loc)
[nb >= blocks.size() && canAutoPlay()]
autoMove()
[else]
opt
[nb == 1]
create nextTetrisBlock
createRandomTetrisBlock()
opt
[blockPreview != null]
removeSelf()
blockRecord.put("I", stats.blockRecord.get("I") +1)
opt
[isAuto = true]
setAutoBlockMove(currentBlockMove)
display(gameGrid2, Location(2,1))
return previewI
setSlowDown()
getSpeed(slowDown)
return t
return madSpeed
opt
[advance() == true]
Alternative
[nb == 0]
gameOver()
addActor("gameover.gif", Location(5,5)
doPause()
opt
[isAuto == true]
printToFile()
[else]
addActor(nextTetrisBlock
setCurretnTetrisBlock(nextTetrisBlock)
act()
opt
[isLineComplete == true]
for(x < nbHorzCells
removeActor(block[x])
getActors(TetroBlock.class)
return ArrayList<Actor>
refresh()
update score
update totalScore