Mukul Chodhary, 1172562
Emily Zhang,
Victor Lok,

Assignment 1 report

1. Code Refactoring

The current design for Tetris is extremely repetitive where each piece class (I, S, J etc.) all contain the same attributes and methods. In the original domain and design model, this becomes convoluted and creates unnecessary work with regard to any updates and modifications to pieces that a player may want to add. Whilst this is functional for a small scale project, it affects the efficiency of extensibility of the game when a large amount of new, unique pieces are introduced as is required in the Tetris Madness extension of the game - requiring a new class for every new piece introduced (in this case, P, Q, and Plus) where the attributes are exactly the same.

In order to change this issue with the current design, we have chosen to encapsulate the pieces in a Shape() class. Each new shape is then an instance of Shape which will reduce the repetition of code that previously existed whilst also making the code more extensible to include more shapes in the future. The structure of shapes will be stored in an Enum ShapeIndex which will contain the piece name along with its sprite pixel locations for the different rotations. By combining these, it greatly increases the efficiency of adding more shapes with variable box sizes by only needing to add the new rotations of the new pieces to the enum in order to add a new piece to the game. The relationships are outlined in our Domain Model (FIgure 1).

In addition to these changes, we added a new Statistics class that will hold the statistics needed for the Tetris Madness extension - which will be explained in more depth in a bit. Our new design model (Figure 2) now contains the changes and new features that we implemented as well. A more detailed explanation of these extensions and changes can be read below.

2. Extensions

Since the first two extensions contain mostly the same changes to the game, we have contained all modifications to the base tetris game in a new DifficultyModifier() class. This class is in charge of determining what difficulty the game is set to and the relevant changes to the game that are required for each level. The difficulty is read through the properties file and will change speed, rotatability and pieces accordingly.

a. Medium difficulty

The first Tetris Madness extension for the game is the **‘Medium’** difficulty where there are three new pieces and the speed is now 20% faster. The three new pieces are added by figuring out the 4 different rotations for the shape and adding those rotations to the enum ShapeIndex. All

rotations of a shape are calculated using a rotation matrix for a 2D space, as we are only rotating by 90° this simplifies to x' = -y, y' = x. Where (x',y') are new coordinates after rotation based on (x,y). To do this, we have created a new function find_all_rotation() which takes in one rotation of a piece and finds rest for the user to add into the shape enum class. Ideally we would have liked to store these values in a text file so that there is less manual labor when adding new pieces to the game, however, implementing this change would be too big to implement in this instance and so we have decided on the former method of finding the rotations.
The speed of the pieces are based on a maximum speed and the other speeds are based on a slowDown variable and so because in this version of the game, the speed is generally 20% faster overall. In order to show this, we have shifted the slow down speeds for each level to be one lower than usual. This means that when starting the game, the slow down speed is now a value of 4 rather than 5 in the easy mode etc.

b. Madness difficulty

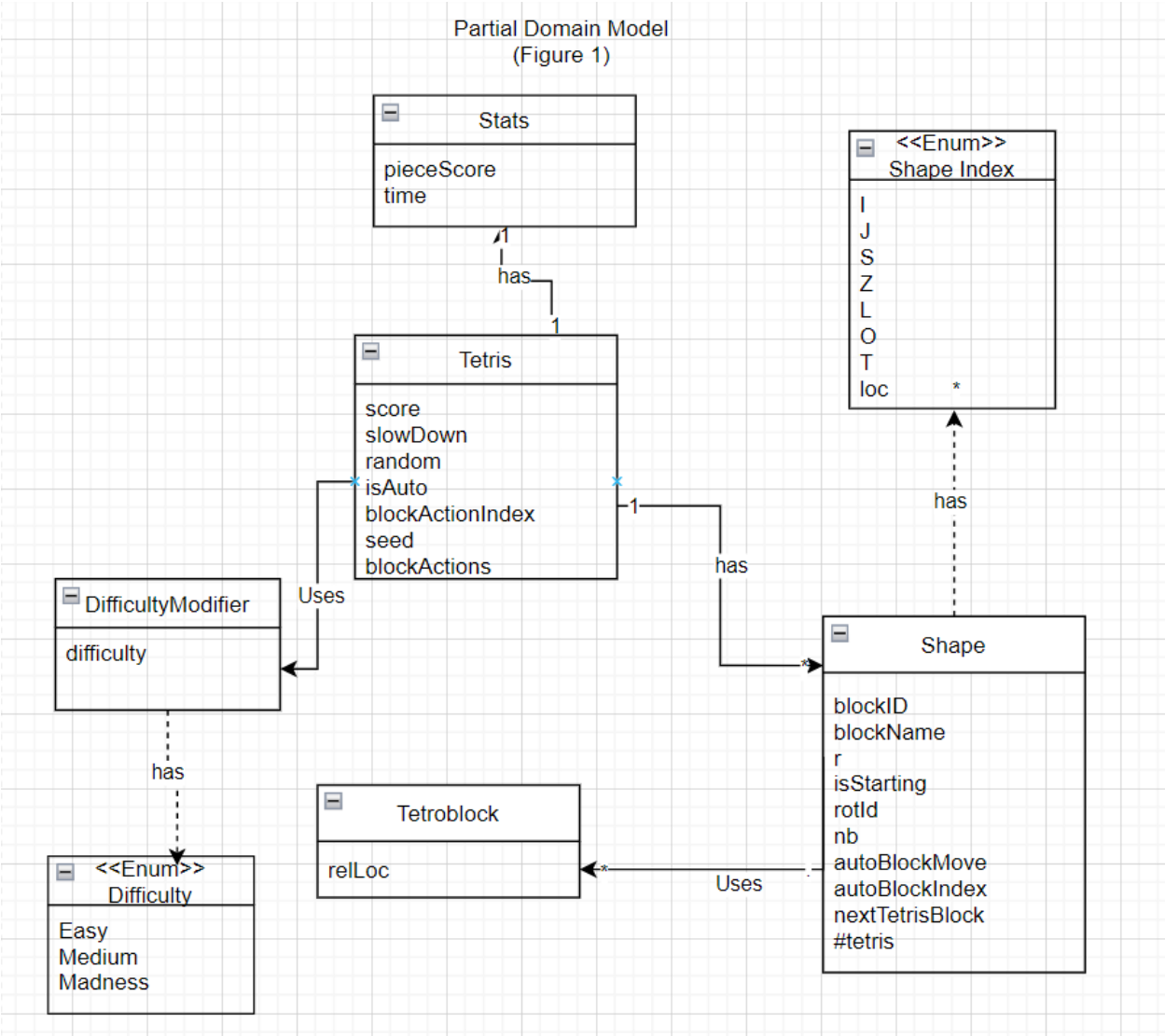
Next, the **'Madness'** difficulty for the game keeps the new pieces added in the medium difficulty. However, the speed is now random and the rotatability of the pieces is turned off. Since this extension is largely the same as the medium difficulty, the majority of the extensions are kept the same. The only additions are to slightly tweak how the speed is determined, and to disable the rotation function. All which can be found within the DifficultyModifier() class. For this extension, the speed of the blocks should be random between S (the speed) and 2S (double the speed). To achieve this, for each block, the base speed is halved and a random value between the base slow down value and half the base slow down value is retrieved and rounded up to find the new speed for the block. Sometimes this will result in a speed that is the same as the base game for that particular level.

c. Statistics

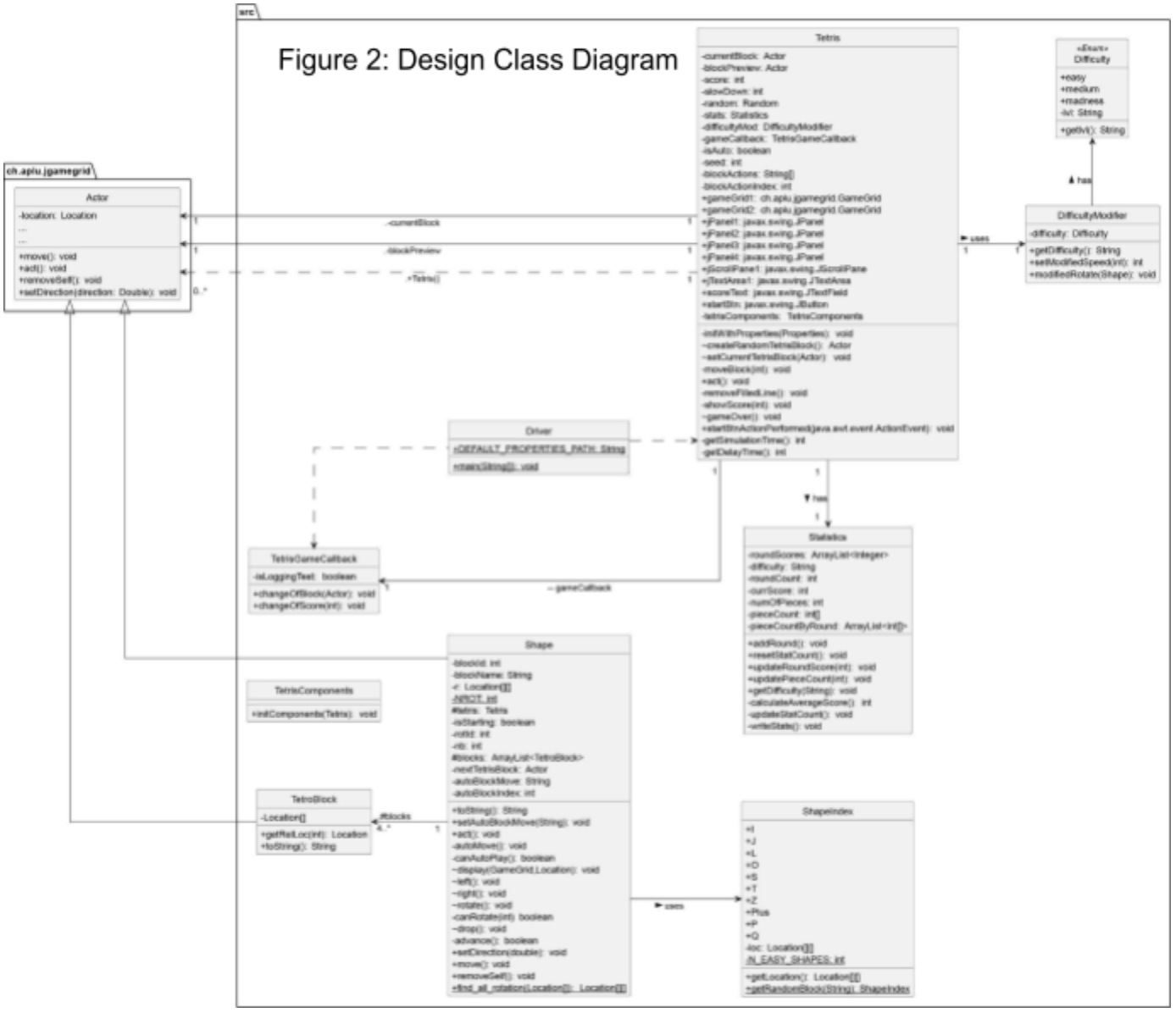
The last extension of the game is to create a document at the end of runtime which includes some statistics from the game. Since this statistics document will be present for all difficulty options, there will be times where the added pieces (P, Q and plus) are not used at all since they 'do not exist' for that game mode. In this case we have chosen to leave these pieces in the list in the document but will keep the value of the number of pieces as 0 to reflect that they were not used in that particular run of the game.

Mukul Chodhary, 1172562
Emily Zhang,
Victor Lok,

Partial Domain Model
(Figure 1)



Mukul Chodhary, 1172562
Emily Zhang,
Victor Lok,



Mukul Chodhary, 1172562
Emily Zhang,
Victor Lok,