Tetris Assignment Ben Higgs 1403099

My version of Tetris runs through a Java applet called TetrisGame. It is all contained inside the Tetris package. In this applet, the general layout is set. A title bar is shown that display “Tetris” and then has three buttons underneath. These Jbuttons are labelled “Start game”, “Change Difficulty”, and “Change Board”. This is how you play the game, and choose some optional settings.

All of these buttons have relevant action listeners attached. When the “Change Difficulty” button is clicked, a pop up alert window is displayed, where you can choose a difficulty to play at. These change the speed of the timer in the main game class. The harder the difficulty, the faster the game. These options also change the score multiplier, and the harder the difficulty the higher the score. The second option button is the “Change Board” button. This button allows the user to change the size of the game board. There are three options, large, normal and small. The buttons both use the JOptionPane that shows a dropdown list to choose from. Upon a choice being made, they manipulate a string that holds the difficulty. The board size button has a function that changes an integer array that contains the height and width of the game board. When an option is chosen, a new different GameBoard is created and is displayed after removing the old one.

My GameBoard class contains all of the build and functionality behind the game. The game is stored as a 2d array of integers. This allows me to keep track of an active shape, colours, inactive shapes and also boundary columns and rows. The GameBoard class is a subclass of JComponent and implements ActionListeners to function.

Upon construction, the constructor method creates a new board using the createBoard method. This initiates a 2d array containing the game board. It is full of zeros apart from a border around the sides and bottom which is all 9’s. A blank 4x4 array is initialized to hold the next shape that will be placed on the board. Game components are then setup that access the difficulty string in the TetrisGame class, and initialize a timer and difficulty multiplier based on this. The game Boolean variable is set to true and KeyListeners are added too.

The Listeners that I have used are a mouse listener and a mouse wheel listener. A class named TetrisKeyListener implements both these listeners, and provides the correct methods that control the movement of the blocks. Left mouse click moves the shape left, right click moves the shape right and the mouse wheel scrolling speeds the travel of the shape to the bottom of the grid.

The paint method that I implemented sets a background colour as Dark grey, and then works out a height and width of each square on the game grid. This allows the grid to be the same physical size no matter what array size the board is. It uses for loops to loop through each item and draw it using the fill3DRect method. Once the main grid is drawn, it checks if the game is active, if so, the next shape box is drawn and the shape will appear under a title placed below the board. If game is not true, the game over text will appear. The score is always shown underneath the game board.

When the game is started, the timer is started by the run method, and it is stopped by a stop method. The ActionPeformed method is called on each repetition of the timer, and the game display is controlled by the game Boolean. If the game is active, and there is no active shape piece on the board (piecactive variable), then the next shape is generated and is added to the game board.

The nextshape method chooses a shape at random from the 4d array of shapes and their rotations. A random rotation is also chosen, and the shape is stored as the 2d array nextshape, and the id is stored. The shape is added to the game board by the addToBoard method. This checks if the game should have finished, and if not, it transfers the next shape into the 2d array shapeactive, and

changes the gameboard in the centre at the top to match the new shape. It updates the lastmove variable and sets pieceactive as true.

On the next timer iteration, ActionPerformed attempts to move the shape down a row on the game board. In a try and catch statement, it calls the moveDown method.

The moveDown method firstly checks by calling the moveDownCheck method that the move is legal. If there is clear space below the shape, it is moved. This is done by a series of for loops, accessing each item that matches the currentshape id. If there is a problem, or the move is illegal, a custom exception is thrown.

This is the class IllegalMoveException. It is a basic, custom exception that is thrown to break a loop of code, and to cause the catch method to be assessed.

If the moveDown method throws an exception, the ActionPerformed method calls the lockBoard method. This simply adds 10 to every item in the gameboard array. This means that colours still match for the paint method in the colours array, so all blocks colours are unchanged. checkRows is then called, which checks the gameboard for full rows. This method assesses the board, if a full row is found, the integer rowtally is increased, and the row is removed by the removeRow method. This takes two 2d arrays, and removes a row, shifting the rest of the rows down and adding a clean row at the top. When all of the rows are assessed, the checkRows method works out the score using the scoring system, difficultymultipler and the rowtally and adds it on. Execution is passed back to ActionPerformed, where a new shape Is added to the board and a new next shape is generated.

If the game finishes, the gameover method runs, where the game Boolean is set to false, the timer is stopped, the TetrisGame gamestate variable is set to false and the startbutton text is changed to say “Clear Grid”.

Whilst the game is played, you can use the mouse to change the position of the shape on the gameboard, and the listeners are watching for input.

They call the moveLeft, moveRight and moveDown methods. The moveLeft and moveRight methods are similar to the moveDown method, and work in the same way, but instead check the space next to the shape on the grid and not below it. They all throw IllegalMoveExceptions if a move cannot be made. The exception message is output to the terminal to show this.

I was unable to complete the rotation method in time. I began working on it and implemented a method shapeArray. This took two pointers, a row and column, and copied the 4 x 4 2d array from the game board to a temp array and returned it. My rotation method was going to check the board for the first occurrence of the pieceactive shape id. When it found it, the shapearray would be generated and checked if it matches the current rotation of the shape. It would keep trying, searching in a radius of 3 columns to the left and 3 rows up. When it found the shape block, it would try and swap in a different rotation of the shape from the gamepieces 4d array. When it found one that matches it would be substituted in, or would throw an IllegalMoveException if no possible rotation is possible.