Scott Forsberg
CSC 143
2019-10-03

# Report 1

## The LEFT and RIGHT Arrow Keys

The EventController class has a method called keyPressed which can process and match button pushes. An EventController object called ec is created as a member of this class, then passed as the data of a message sent to the method addKeyListener of the JFrame object called f, which is the active window we see as the game in play. This facilitates the detection of the physical keys, and allows the ec object to process them.

The keyPressed method in the EventController class has case statements to match which key has been pressed, and execute the appropriate code. Matching the left key for example would be case statement “KeyEvent.VK_left”.

This sends a message to the handleMove method of the EventController class with “Direction.LEFT” as the sent data. The handleMove method sends a message to another method called movePiece of the game object with the direction as the sent data, which in turn sends a message to the move method of the specific piece object actively in play, which in the default scenario is LShape. The move method of the LShape class first checks whether the piece can move by sending a message to the canMove method also defined in the LShape class. If the checks within this method find that the piece is indeed able to move, it returns true back to the move method of the LShape class.

IF the piece is able to be moved, the move method of the LShape class sends a message to the move method of the Square class with the direction being the sent data. This method then sends a message to the canMove method of the square class, which has case statements to match the direction and perform custom tests to determine if the piece can indeed move.

Specifically the “grid.isSet” is utilized to see of the “col - 1” is empty for a left key press, and a “col +1” for a right key press, with “grid.isSet” revealing whether or not a given square is occupied.

The Square class canMove method will the return a Boolean true or false back to the move method of the same Square class, and then match the direction via a case statement, and then perform the adjustment of the row or column respective to the direction. Specifically, down would be “row = row + 1”, right would be “col = col + 1”, and left would be “col = col - 1”.

In the Game class, first a message is sent to canMove in the Square class, which matches the direction in a case statement, and performs check of the surrounding squares, in the example for left it will check if “col == 0”, or if the col to the immediate left is set as determined by “grid.isSet”.

**Piece Drop Down With Spacebar Press**

An instant drop down effect was implemented to drop the piece to the bottom upon the space bar being pressed. This was first achieved by using the regular timed drop down events, but altering the delay to be set to zero for the increments between the space bar press, and the piece reaching bottom.

A boolean data member was declared in the ancestor class Tetris, so that subsequent descendant classes could all have access to it.  This variable is downPushed. The EventController class has the key-matching method keyPressed which updates the downPushed to true when spacebar pressed.

The regular timed events processed in the actionPerformed class were then modified to check if the downPushed variable is true. When a true value is found, the setDelay method of the timer object is accessed, setting the delay to zero and allowing the piece to free fall. When the value then becomes false, when the piece reaches bottom, the delay is then reset to the default 800 milliseconds in the updatePiece method of the game object, before creating a new piece. The new piece then falls at the initial speed.

However, this entire function was rewritten and replaced entirely . A new direction DROP was added to the Direction Class. Then the EventControlller method keyPressed was updated to load DROP instead of DOWN. Then, the Game class movePiece method was updated with a while loops predicated upon canMove, direction DOWN. This rewrite was to better write junit tests.

**Java Unit Testing**

Several methods were implimented in the TestrisUnitTest class, for the purposes of verifying the features of the game. These methods are testCheckRows, testMoveSquare, testMoveLShape, testSquares, and testDropDown.

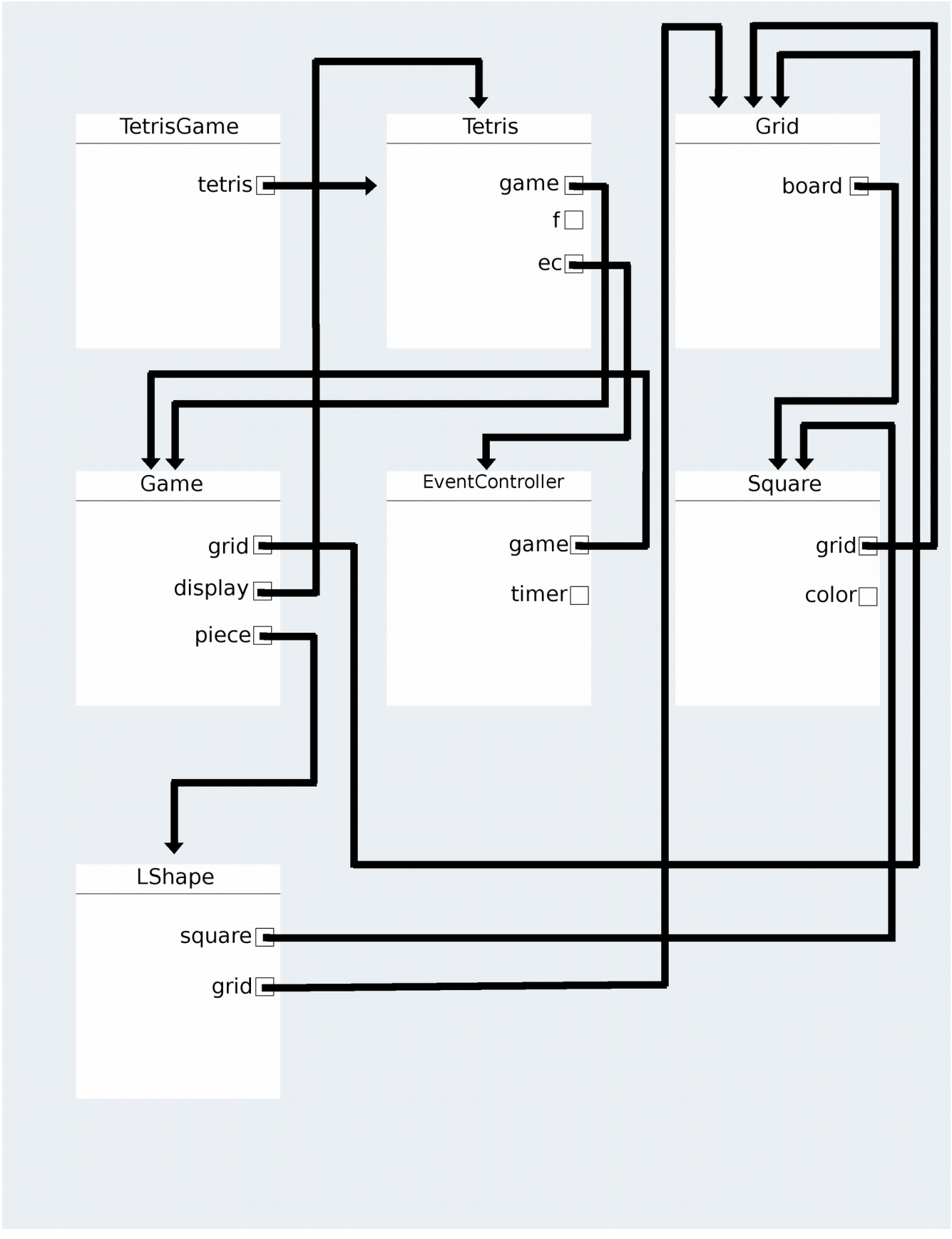
The testCheckRows method first instantiates a Grid object g, and then fills in the two bottom rows with squares, numerically rows 18 and 19, by using the set method of the g object. Three extra squares are then added, two being stacked on top of one another in column 3, and one being on the last column 9.

A message is then send to the student authored method checkRows, which should remove the full rows. To verify this, assertTrue is used with the data of the message being the squares to check, in the form of "g.isSet(19, 3)" for example which checks for the floating test squares after row removal. The lack of a square is also tested, as well as an arbitrary floating square.

The testMoveSquare method also uses a Grid object. A new square object is places on the grid at row 0 col 2 with it's boolean mobile data member set to true. Another is instantiated at 3,5 with it's mobility set to false. The Square objects have messages sent to their move methods with the data being "Direction.LEFT" and "Direction.RIGHT". The assertTrue check is performed afterwards to assure the squares have moved to the correct position.

The testMoveLShape class also instantiates a grid object. It then creates two int arrays lCurRow and lCurCol. Positions 0-3 of these arrays represent positions 0-3 if the LShape piece, col and row respectivly. The array is updated and then a move message is sent to the LSpahed class simulating a button press. The testShapeLEft and TestShapeRight methods then run an assertTrue to check if they have updated their col values as expected. IT also attepts to press the right button more times than there are spaces, and then check to see it it went off of the board with the testSquares method.

The testDropDown method uses a Grid object, and like the previous method uses arrays to track the LShape piece movement. It simulates the DROP direction, using a while loop with canMove. Afterwards a check is done via test arrays to see of the piece has decended into to the correct position.


TetrisGame
tetris
Tetris
game
f
ec
Grid
board
Game
grid
display
piece
EventController
game
timer
Square
grid
color
LShape
square
grid