

Jeffery Zhu, UTEID: JZ995  
Lee McCuller, UTEID: LPM284  
CS 315H  
10-18-06

## Program Assignment 4b – TetrisBoard and Brains

### Problem description:

In the second part of the assignment, we had to implement the TetrisBoard, JBetterBrainTetris, JAdversaryTetris classes, and our own Brain subclass.

Our strategy was to implement class individual methods of the interface that each class was based on one at a time – in doing that we were following the design concepts of decomposition and abstraction without even realizing it. In implementing complex methods, such as the place() method of the TetrisBoard class, we had to decompose the big problem into smaller problems, and abstractly think about the information we were handling instead of just trying to comprehend it all at once. One of the advantages of this assignment was that we were given a very precise list of design goals and restrictions in the handout. This allowed us to focus much more on efficient implementation of the specified interface rather than trying to design a large complex system of interactions between classes.

### Solution Description:

In implementing these classes, we held to the concept of subtype polymorphism – anything that uses a Board should be able to use our TetrisBoard without a problem, and so on for the rest of our classes and their super classes.

#### TetrisBoard:

Our basic design was to add in the data structures necessary to represent a board and its attributes in an efficient manner. Then all we did was implement the specified board interface using the data structures. We had a board buffer and a back buffer for the undo operation. We also have arrays to hold data about the board without needed constant re-computation and we have these double buffered for undo as well.

#### JBetterBrainTetris:

A subclass of the JBrainTetris class. All we did was change the constructor to use our brain and modified main to call itself instead of its super class.

#### JAdversaryTetris:

A subclass of JTetris. It overrides the console construction method to add in a slider and it also overrides the pickNextPiece() to use a Brain object and the slider so that when a new piece is chosen there is a chance based on the slider that it will choose what the brain considers the worst piece to add to the board. To do this it has the brain score the best move for each piece and it decides which piece got the worst score. Here we found a small bug in the original implementation of JTetris that causes the piece not to be committed if pickNextPiece uses board.undo() without first doing board.commit(), which pickNextPiece should not have to call.

#### BetterBrain:

Here we subclass LameBrain and override the rateBoard() method to give scores that are a much more accurate reflection of what boards are the most desirable to a human player. The rubric we decided to use is: It treats all heights the same.. it only cares about the difference

between the average and the max heights only avoiding having a max much higher than the average. The LameBrain wants to avoid creating holes, ours works the same way except if a hole already exists on a row then a new one is much less of a concern because the row can't be filled anyway. It also has a large calculation that determines how bumpy the terrain is. It seeks to create sufficiently bumpy terrain to allow pieces to rest in the bumps and not make holes. At the same time it seeks to reduce cliffs because only 'I' pieces are useful with them and we seek to reduce the need for 'I' pieces. It also seeks to fill rows to delete the total number of filled spots, however it will not do this if it creates a new unique hole in a row, this helps prevent 'L' pieces from standing up and causing holes. Finally it tries not to put pieces over holes that are near the top, so that the holes can be uncovered.

#### Testing:

##### TetrisBoard:

Testing TetrisBoard was done on the provided JBoardTest class and the test sequence of pieces used by the JBrainTetris class. The board successfully completed tests in both environments. Jboard test was necessary to fix the misconceptions and oversights we had in the instructions. Most of these simply regarded the heights array. Once we fixed heights we fixed a couple bugs in the widths calculations (mixed x,y index variables for the board). Once those were fixed we tried JbrainTetris in test mode and it passed first try.

##### JBetterBrainTetris:

Ran it under test mode while our BetterBrain used LameBrain to see if it passed the test sequence. Then ensured that it was using BetterBrain by modifying BetterBrain and testing if the standard test sequence fails.

##### JAdversaryTetris:

Saw the bug immediately and fixed it, then had it inherit from JBetterBrainTetris instead of JTetris to see if the brain could beat itself.

##### BetterBrain:

We just played this slowly and analyzed its mistakes, then played it fast and saw how long it lasted. This one is more of a subjective measure of performance once it avoids common mistakes so we just modified it slowly and tried to identify the subtle changes.

#### Results:

We made a working Tetris board back end that successfully performs all the tests through the provided front ends. The Adversary usually beats its own brain quickly.

#### Log:

Tuesday, October 17

7:50 – 8:15 pm

Jeff drove

8:15 – 8:50 pm

Lee drove

8:50 – 9:20 pm

Jeff drove

9:20 – 10:30 pm

Both debugging and testing

10:30 – 10:40 pm

Discussed brain and adversary

Wednesday, October 18

4:30 – 5:15 pm

Both setup classes together to get project started – JBetterBrainTetris, JAdversaryTetris, and BetterBrain.

5:15 – 5:30 pm

Lee coded a lot of JAdversaryTetris

5:30 – 5:45 pm

Debug adversary (huge bug in the order of commits and nextpiece that break any nextpiece that requires the undo operation like adversary)

5:45 – 6:30 pm

Worked on brain together

Thursday, October 19

7:00 – 7:38 pm

Began the report together

7:40 – 10:00 pm

Worked on and tested BetterBrain together.

10:00 – 10:30 pm

Finished the report.