Danny Chen
Christopher Getz
10/12/13

Assignment 4 Part B Report

Overview:

The purpose of this part of the assignment was to create the structure of the tetris board, and allow for quick computational decisions regarding the state of the board. Then we had to make an AI to play tetris, and based on that AI, make a difficulty setting.

Solution Design:

We created our board with the goal of speed in mind. All of the calculations to update the values are performed when placing down a piece or clearing rows. This ensures that when programs are inquiring about the state of the board, the board takes no time calculating the values, and can directly return the reference or value requested. We did not worry about optimizing the runtime of sanityCheck because it is a debugging tool that will not be run during normal play. Our board is stored with the heights as the first index, so that rows can be copied by reference, which improves the speed of clearing rows.

When clearing rows, our solution passes through the board only once. We originally were going to remove rows one at a time, and shift down everything above it each time. We realized this was slow and inefficient. Instead our design goes through the board starting at the bottom, and copies over the lowest non-filled line that hasn't been copied down. Once the lines being copied reach the top of the filled board, we replaced all the rows above the current one with empty rows. When copying we decided to copy references instead of array values. We did this for speed, and had to make sure that no two pointers in the final board were the same.

Our AI was produced with a genetic algorithm that rates the board based on Average Height, Height Variance, number of holes, and the difference of the highest and lowest column. We evolved the coefficients for these values based on 50 generations with 20 individuals in each generation. We had a 5% chance of mutation, and mutate by a random amount for each value. We selected the 5 with the highest minimum score, and averaged their coefficient to produce the base for the next generation. We thought that these metrics were more indicative of how good a board state was or was not. We also selected the highest minimum score to try and improve consistency.

The adversary simply uses the brain's computation to choose the piece that has the worst optimal placement. If the piece is not chosen at random it checks the score of the best move for every original piece, and then chooses the highest scoring piece.

Our solution assumes that the work done previously is correct, that pieces are circularly linked, and the rotations are correct. Our board can handle any piece, because we traverse the points array, and so a larger piece simply means placing more blocks. Our brain assumes that it can move a piece to any place that it can be dropped straight down to, when in reality it doesn't always have time to.

Danny Chen
Christopher Getz
10/12/13

Implementation:

This is how we implemented the following methods:

Place- First we check if the placement results in the box surrounding the piece to be out of bounds. If it is inside the grid, we backup the grid before changing anything. We then step through each point in the body of the piece and place it into the grid. If there is a block already there, we return the bad placement value. When placing each block, we increment the widths array for that row, and if it is the same as the board length, we know when we are finished to return the row filled value. If the placement works for all points, and neither the filled row or bad placement happens, then we simply return the ok placement value. We update height and width for each block placed, that way we don't have to traverse the entire board updating values.

clearRows- Our clear rows has two pointers, one to the current row, and one to the lowest row that hasn't been copied, which we call the copy row. We increment the copy row until the row it points to isn't filled, and then copy this down. Once the copy row goes above the max height we fill in the remaining rows with empty rows.

sanityCheck- For sanity check, we went through and recalculated the widths and heights array, and checked that the stored arrays match the calculated ones. We also checked to make sure none of our rows had the same pointer value, since we copied pointers in clear rows.

For the brain, we used the same type of board analysis as the LameBrain, but changed the metrics, and used genetic algorithms to determine the coefficients.

Testing:

When testing, we used a combination of manual testing and automated testing. We used our genetic algorithms evolution runs as automatic tests, as sanity check was turned on the whole time. This resulted in 1000+ games of tetris, which actually caught a rare bug in clearRows. For manual testing we used the stress test to be sure that our board was working correctly. We tested with the JBoardTest for a while, but that was mostly useful for fixing clearRow, rather than finding issues. We also watched both brains play games at ~60% speed to check for both behavior and board correctness.

Sanity Test is a rigorous test of all of the data structures, and aside from checking the pointers of all the rows, we felt the described tests covered all of the areas that should be checked. Sanity Test was run an enormous amount of time during the evolution process, and

prints out any inconsistencies or errors in the state of the board and the other data structures. This amount of running makes us confident that our TetrisBoard is completely correct and performs well.

When testing our Brain, we looked at the scores that it was achieving through the evolution. Since all of the evolution program ran successfully, we know that the brain never is returning a move that breaks the program. We also know that it successfully plays tetris because of the relatively high scores achieved after the evolution.

Pair Programming Log:

Chris Getz	105 minutes Driving	10/9/13
Danny Chen	45 minutes Driving	10/9/13
Danny Chen	90 minutes Driving	10/10/13
Chris Getz	30 minutes Driving	10/10/13
Danny Chen	75 minutes Driving	10/11/13
Chris Getz	30 minutes Driving	10/11/13
Danny Chen &Chris Getz	180 minutes commenting and writing	10/12/13