workforce, and satisfying the goals of the NSF grant.

The TYC Education Committee has also been involved in recent discussions and workshops related to the role and nature of mathematics in computing curricula. This has been a topic of debate for some time, of course, but recently those ongoing discussions have taken on more deliberateness. Last fall the MAA and AMATYC jointly engaged in NSF-funded workshops to examine the mathematics associated with a variety of two-year college programs (including IT), a project in which members of the ACM TYC Education Committee participated. This field is one the Committee will continue to monitor and participate in as it formulates additional guidelines and reports.

The TYC Education Committee has recently updated its website, and encourages interested parties to visit <www.acmtyc.org>. The Committee welcomes inquiries and comments related to the two-year college environment. Uniquely positioned (especially in the US) between high schools and upper-division baccalaureate coursework, and serving a student body made up of individuals with very diverse goals, two-year colleges are exciting and important places of education. The Two-Year College Education Committee enjoys its work, is pleased and excited to see a host of updated curriculum guidelines released and under development, and looks forward to fostering collaboration with ACM efforts related to both the high school setting and the university environment, across the breadth of computing curricula.

Please address questions concerning two-year college activities to:

Robert D. Campbell
TYC Education Committee Chair
<B.Campbell@rvc.cc.il.us>

Featured Columns

Nifty Assignments

Tetris on the Brain

Nick Parlante
Computer Science Department
Stanford University
Stanford, California 94305 USA
<nick.parlante@cs.stanford.edu>

This column is a spin-off of the Nifty Assignments panel I chair at SIGCSE. The panel <cse.stanford.edu/nifty/> operates at a very practical level — showcasing favorite assignments to promote the sharing of assignment materials and discuss assignment ideas. In particular, each assignment has its own material such as a web page of handouts and starter files for interested instructors.

This column should operate in a similar vein — discussing assignment ideas or techniques and elaborating ideas that come out of the panel. For this issue, I'll look at the overlooked "brain" feature of the "tetris assignment" presented at Nifty Assignments 2001 and explain why the brain may actually be the best part of the whole assignment.

Tetris OOP Modularity

Most of the work in the Tetris project is in the Piece and Board classes. The Piece and Board make a nice exercise in OOP modularity — dividing the overall tetris problem into classes with features such as modularity, encapsulation, and separate testing. The game playing AI is a relatively simple layer added on top of the core by the brain classes.

At SIGCSE, I emphasized tetris as a large OOP modularity assignment since modularity is one of my favorite themes and the Piece and Board make a great example. The problem is that the Piece and Board classes are pretty challenging to write and debug, so it's not the sort of thing a course can adopt lightly.

Tetris Brain

In contrast, the brain is short and fun. The brain code is simple to write and great fun to watch and play with. It avoids the debugging trauma of getting, say, `Board.clearRows()` to work exactly right in all cases.

The brain strategy is pretty simple. There are only about 30 ways to play a particular piece in a board, so

the brain takes a brute-force strategy, rating each of the possible resulting boards. The brain code is also simplified by playing the role of client to the Piece and Board classes, which do the heavy lifting. The whole Piece-Board-JTetris system is available in a runnable `.jar` file, so students just write their own brain code and use the "Load Brain" button to load their class into the system.

Brain Code

A brain expresses its strategy in a `rateBoard()` method that, given a board with a piece played into it, computes a rating score with lower scores for more desirable boards. The code that enumerates all the possible plays, calling `rateBoard()` for each play, is in the standard brain superclass. The students can concentrate on expressing their strategy ideas in their `rateBoard()` method and the bookkeeping is taken care of for them.

The code below shows a simple brain. The code is from the `LameBrain`, which comes with the tetris package. The `LameBrain` has two strategy ideas...

1. *"Holes" in the board are bad.* A hole is an empty square in the tetris grid with filled squares above it. The more holes there are in a board, the less we like it. In the code below, we count holes with a while-loop that iterates over each column from top to bottom.
2. *Height is bad.* The taller a board is, the less we like it. This is, in effect, why we like to make plays that clear rows. The resulting board has less height. The code measures both the average and maximum column height in the board. Measuring both types of height allows us to express a preference for playing, say, the long skinny piece laying down instead of standing up. The average height is the same for both plays, but the maximum height can distinguish the two in some cases. The `Board` class keeps track of its height statistics, so the code below just queries the numbers from the board and does a little arithmetic.

```
public double
    rateBoard(Board board) {
    int width =
    board.getWidth();
    fint maxHeight =
    board.getMaxHeight();

    int sumHeight = 0;
    int holes = 0;

    // Count the holes and
    sum up the heights
    for (int x=0; x<width;
    x++) {
    int colHeight =
    board.getColumnHeight(x);
    sumHeight += colHeight;

    // y of first possible
    hole
    int y = colHeight - 2;

    while (y>=0) {
    if
    (!board.getGrid(x,y)) {
    holes++;
       }
       y—;
    }
  }

    double avgHeight = ((dou-
    ble)sumHeight)/width;

    // The weights, 8, 40,
    etc. are guesstimates
    return (8*maxHeight +
    40*avgHeight +
    1.25*holes);
    }
```

The `LameBrain` does a competent but not great job of playing tetris. I used to ship the project with a better brain, but I found `LameBrain` motivated students better. Watching the `LameBrain` mis-play can be pretty compelling actually. Students watch the `LameBrain` fail to use some obvious bit of strategy. From there, it's natural to want to improve the code to do the strategy right. (This impulse is probably the same instinct that makes people kibitz or complete other people's sentences.) Students do not have to watch the `LameBrain` play for very long before improved strategy ideas come to mind!

Brain Assignment

At its simplest, we can use the brain as a fun little lab. Students with basic Java programming skills can play around with strategy ideas and see how they work. The brain code is also open ended enough to make a large project. It has the appeal of a creative, open-ended problem where there are no fixed right or wrong answers.

There are many tetris strategy heuristics, but here are few common ones.

- Piling more blocks on top of an existing hole is bad. This is most important for holes near the top than for deeply buried holes.
- A deep trough that can only be filled with the long-skinny piece is bad.
- Holes that are adjacent to other holes vertically or horizontally are a little less bad.

Figure 1 illustrates some of these features.

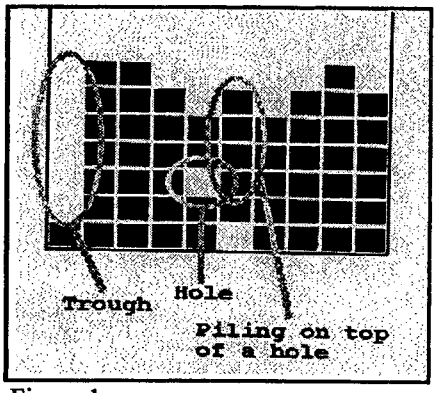

Figure 1

Finally, there's the matter of the weights, such as the 8 and the 40 in the code above. It's easy to make up values for the weights with a little guessing and hand testing. To really optimize the weights, machine support is required. The JTetris GUI code can be modified to run the brain against a few fixed piece sequences and report the average number of pieces played before losing. For maximum effect, we can tune the weights with a separate machine optimizer such as a genetic optimizer.

Conclusions

The tetris core is great if you are looking for a large, difficult OOP modularity project. However, I have come to realize that the brain by itself is a much easier assignment to adopt. It has all the fun aspects of the larger tetris project, but with a fraction of the coding. The brain code appeals to student's creative and playful instincts. The brain can be a large project on its own, but students can have fun writing simple brain code in just a few hours.

Links

You can find the materials for all the Nifty Assignments at <cse.stanford.edu/nifty/> and the tetris materials in particular at <cslibrary.stanford.edu/112/>.

Editor's Note:

We are thankful that Nick loves tetris as this image shows and welcome him to the Featured Columns of `inroads`.