3: One Column of the Can't Stop Board

CSCI 6626 / 4526 April 2018

1 Goals

- To use enumerated types as return values and as subscripts.
- To model one part of the Can't Stop game board.
- To build and execute good test plan.

2 Instructions

Section 45, below, provides information that explains how the game works and provides a basis for future assignments. However, this week, you do not need to implement anything in part 4.

Column state. In your enum module, implement another enum to represent the possible column states (captured, pending, available)

Define a Column class. In the column class, define a static array of 13 const integers, which are the lengths of the 11 columns, with two 0's on the beginning, so that slot 2 of the array stores the length of column 2. Define private data members for the marker array [5 slots], the state of the column, the column number, and the column length. All should be initialized by the constructor.

Implement these class functions:

1. A constructor with one parameter, the column number, and an appropriate destructor.
2. A print function and `operator <<`. Print all the column's data so that you can see what actually happens when you call the functions. Print the column horizontally, with the column number at the left, followed by the column state (captured, pending, available), followed by a display for each square. Print a square using 7 columns to show two spaces followed by the appropriate combinations of the letters TOYGB, indicating the colors of the tiles currently in the square. Your display will be clearest if you use `_` or - to represent tiles that are not present. For example, if a square has a tower and a green tile, the display would be `T--G-`.
3. A public accessor function *state()* to return the State of a column. (Refer to part 4.2, below.)

3 Testing and Submission

In the unit test, create two Column of different lengths and test the Column class functions as usual. Submit all the classes in your program, even if you have not changed them. That lets me compile and run your code if I need to.

4 Context: The Board

The Can't Stop game board has 11 columns, numbered 2 to 12. For simplicity, you will model this as an array of 13 columns, where slots 0 and 1 of the array are left unused. To do this, you need two classes, Board and Column. In this assignment we will build only the Column class; the Board can wait until the Columns are finished.

Markers. The game has two kinds of markers: towers and tiles. Tiles come in four colors (orange, yellow, green, and blue), and each player uses one color. Towers are white; three of them are given to a player at the start of his turn. At any given time, up to five markers, one of each color, might be in a column. Use your enumerated type for the colors.

4.1 The Rules for One Turn.

It is not time yet to implement the logic for the progress of a turn. This information is included here to help you understand the way a Column will be used.

At the beginning of each turn, a player is given four 6-sided dice and three towers. He rolls the dice then arranges the four dice into two pairs. The two pair-totals determine which columns can be used on this roll. Suppose the pair-totals are 6 and 8. The possibilities are:

- The player must decide which pair-total to use first. Suppose he selects the 6.
- If the player already has a tower in column 6, he moves that tower one square forward.
- Otherwise, if he has a remaining unused tower and if column 6 has not been captured, he may call startTower() to place a tower in that column. If he already has a tile in column 6 (marking progress from a previous turn), he places the tower on top of the existing tile, then calls move(). Otherwise the new tower is placed on square 1.
- If all three of the player's towers are in use, and they are not in column 6, this pair-total cannot be used. He then considers the second pair-total, which is 8 in our example. If he cannot use the 8 either, the roll fails and the player's turn is over. (We say he "goes bust".) The towers are removed from the board and given to the next player. His tiles, if any, remain, unmoved, where they were at the end of the previous turn.
- If the roll succeeds, that is, at least one tower moved forward, the player must choose whether to roll again (and risk losing all progress) or stop. Strategy consists of knowing when to continue and when to stop. If he always plays safe, he will lose the game. If he always takes the risk, he will lose the game.
- When a player decides to stop, each tower is replaced by a tile of his own color. Other tiles of the same color in the same column are no longer needed and may be removed. Then the towers and the dice are passed to the next player.

4.2 Representing One Column

Each column has a label (2...12) and a number of squares, as follows:

Column	Length	Column	Length	Column	Length	Column	Length
2	3	5	9	8	11	11	5
3	5	6	11	9	9	12	3
4	7	7	13	10	7		

Column state. A column can have three states: available, pending, or captured. (Define an enumeration and an array of words to represent the states.) In the beginning of the game, all columns are available. When a player, over many turns, moves his marker to the last square in a column, the column state changes to pending. If he stops before going bust, he captures the column. All other player's markers are removed from the column and, after that, no player can put a marker in it.

Column contents. A column may or may not contain markers. The easiest way to model this is by a data member that is an array of 5 ints, subscripted by the Color type. For example, in column 4, the array slot for White would store the position $(0 \ldots 7)$ of the tower in that column. Position 0 would mean there is no tower in the column. Position 7 would mean the tower just captured the column. Similarly, the array slot for Yellow would store the position of the yellow tile, if any, in the column. Use the enum constants as subscripts to access this array.