6: Completing the Board

CSCI 6626 / 4526 February 2018

1 Goals

- To complete the model of the Can't Stop game board.
- To continue defining non-trivial game logic.
- To build a class that aggregates an array of another class.
- To use delegation in the design of some Board functions.

2 The Board

In the previous programs, you implemented a Player class and a Column class. This week, the job is to build a Board class that will form an interface between the Game class and the Columns.

Board data members. Members of this class must include:

- A counter to say how many towers are in use and an array of length 3 that tells which columns contain those towers.
- An array called backBone, of 13 Column-pointers. The Board constructor will set array elements 0 and 1 to NULL , then allocate each of the 11 columns and store the Column pointers in the backBone. Of course, this can be done with 13 consecutive assignment statements. An less tedious technique is to use an allocation loop that takes the column length out of a constant array, `lengths = { 0, 0, 3, 5, 7, 9, 11, 13, 11, 9, 7, 5, 3 }` . This array can either be in the Board class or in the Column class, and that determines how you define the Column constructor.
- A Player* for the current player. At the beginning of each turn, this pointer will be set to point at the current player in the list that was created at the beginning of the game.

Board function members.

- A constructor (no parameters because every Board is the same) and an appropriate destructor.
- A print function and matching definition for `operator<<`. These should display the board so that it is neat and looks something like a Can't Stop board with horizontal racetracks instead of vertical columns[1]. You may need to modify your Column::print() to do this.
- void startTurn(Player*)
  Store the Player* and set the tower-counter and entries in the tower-column array to 0.
- bool move(int column);
  Return false if the player can not move in this column because it is pending, captured, or there is no available tower. Otherwise if there is no tower in the column and you still have an unused tower, place one in the column. Move the column's tower one square forward. Return true if you made a move.
- void stop()
  Replace all towers by tiles of the right color. Use the tower array to decide which columns need to be stopped, then delegate the action to Column::stop().

[1] A more difficult alternative is to let the Board class produce the entire display, using accessor functions to query the Columns about what tiles should be displayed where. This is more consistent with the Model-View-Controller design paradigm. However, PLEASE do not do this now. It is more important to work on the logic of the game.

- void bust()
  Delegate the bust() action to all the columns that contain a tower for this player.
- You may find other functions that you need. If so, they will probably be private functions.

3 The Game class

Modify your Game class: keep a set of Dice and a single Player and add a Board. Remove any other individual components. Write and call a unit test for Board. Write an integration test for Game that includes the new components.