PA3 key: Board game.

The objective of this PA was to implement an efficient Board class as per specifications in the assignment. (The Player class can be implemented as a simple structure without any methods). Here are the things we should look for in the students’ design & implementations:

The following main requirements should dictate the algorithm design and choice of data structures:

* For the INSERT method, it is necessary that the Board class supports a way to search by both Player ID, AND also by <x,y> position to check if the target cell is unoccupied.
* For the MOVETO method, it is necessary to support a way to search by both Player ID (to locate the source cell), AND also by <x,y> position, to check whether the destination cell is occupied.
* For design purposes, m should be assumed to be very large when compared to n. Therefore, storing a full mxm matrix is ruled out (bad design).

To meet the above main requirements I’d design the solution as follows (of course other variants are possible):

* The Board class contains two data structures:
* I) One Balanced BST to search by Player ID. In STL this is best designed as a *map<Key,Value>* where just the Player ID is used as the KEY and the corresponding Player Object is used as its VALUE. For convenience let’s call all this d/s the “ID\_tree”.
* II) Another Balanced BST to search by X coordinate value (and within each node by Y coordinate values). For convenience let’s call all this d/s the “Position\_tree”. In STL this is best designed as a map<X, Y values> where the tree is sorted by X values, and within each node a list of Y values are kept (sorted or not, shouldn’t matter because as I have mentioned in the assignment, the number of distinct Y values occupied per X value is assumed to be constant). For this reason, the list of Y values can either be kept as a linked list, or array, or even another BST – all three options okay, but I’d prefer a simple unsorted linked list).
* Using the above data structure design, I will implement the Insert, Remove and MoveTo methods as follows:
* A) INSERT(ID,x1,y1): Look for ID in the ID\_tree and if it exists, return false. If not, then, search for <x1,y1> in the Position\_tree (first by X value and then by Y value), and if that position is already occupied by another player then return false. If not, then insert the player on the board by inserting a node in both the ID\_tree and Position\_tree.
* B) REMOVE(ID): Look for ID in the ID\_tree and delete the entry from ID\_tree and then also remove the player’s x,y entry from the Position\_tree (remember that the player object in the ID\_tree will store the x,y value.
* C) MOVETO(ID,x2,y2): Look for ID in the ID\_tree and if it exists, let its current coordinates by x1,y1 (this is the source). Then do the following checks:

1. First make sure that you can move the player from x1,y1 to x2,y2. Basically check if (x1==x2) or (y1==y2) or ( (x1-y1)==(x2-y2) ) or ( (x1+y1)==(x2+y2) ). If any of these conditions is true, then that means you can move the player from source to destination.
2. Next, make sure the destination x2,y2 is unoccupied. This can be done by a simple search in the Position\_tree. If it is occupied by another player, then let that player be referred to as ID2.

If both the conditions (a) and (b) above hold, then move the player from x1,y1 to x2,y2 by doing the following, in order:

1. First, remove the other player ID2 (if that exists) from the ID\_tree. When you remove, print that player info.
2. Next, remove this player ID from x1,y1 (in the Position\_tree), and reinsert it into position x2,y2. Update the coordinate information in the ID\_tree.

All of these operations will take O(lg n) time, guaranteed by the balanced BST property.

As for grading, it is important that the students keep track of two data structures, one to search by ID and another by position. If they keep only one and look for the other, then that would mean more time, and hence less points for their assignments. Then follow the grading rubrics given in the PA for distributing points.

Hope this helps. If you have questions let me know.