Quadris Design Plan

Group Members: Samatar Abukar, Lester Lim, Pranav Tripathi

**Quick Overview of Major Classes and Structures**

**Coord Structure:   
Contains:** int x, y

**BlockCoord Structure:  
Contains:** coord1, coord2, coord3, coord4 -> Coordinates of each block

**BlockCell Structure:   
Contains:** char c -> the Character corresponding to the type of block  
 int num -> this represents which block the character is part of

**Level Class:  
Contains:** int n -> Corresponds to level(0-6)

**MakeBlock() -> Creates the blocks**

**LevelUp() and LevelDown() -> change the difficulty of the game**

**Block Class:  
Contains:** BlockCoord bc -> The coordinates of the block’s current position  
 Grid\* g -> Pointer to grid  
 Level \*curLevel -> helps in deciding if the blocks are heavy

**Left() and Right() and Down() -> Shift the coordinates of the block in the correct direction  
Clockwise(int version) and Counterclockwise(int version) -> Rotate the block in the correct direction. The version integer stores the current iteration of rotate so the function knows how to rotate the block**

**Drop() ->Drop the block as far as it can go**

**Grid Class:  
Contains:** vector<vector<BlockCell> > theDisplay -> the actual grid itself

**Update(BlockCoord bc, char c) -> Takes the coordinates in bc and updates those respective coordinates in theDisplay to contain c**

**Check(BlockCoord bc) -> checks to see if the new coordinates of the block are valid in respect to the actual game grid itself. Returns a Boolean**

**RowClear() -> if a row is filled entirely, that row is removed from the display and each row above is moved down one**

Description:

To implement the game of Quadris, our group intends on taking an object-oriented approach.

The design of the game begins through the Level class which takes in an integer value corresponding to the level of difficulty that the game will be played on. This is used in the MakeBlock function which will then use the correct creation scheme to create Block objects according to the level difficulty. As the game progresses, calling LevelUp and LevelDown will change the difficulty accordingly.

The core part of our design involves fluid interaction between the current Block that is to be manipulated and its placement on the grid by using an observer pattern.

Each block begins at the (0,0) position on the grid. The Block class contains the functions necessary to manipulate the Block’s position on the grid. Using a BlockCoord(structure containing four (x,y) coordinate pairs) as one of its private fields, the functions left, right, clockwise, counterclockwise, down and drop will modify these coordinate pairs accordingly. Throughout the design document, these will be referred to as the Manipulating Functions.

The next major part of the design is the Observer pattern which will take the changes being made to the coordinates in Block and apply them to the Grid. After each call of a Manipulating function, the grid will call Check to make sure that the move is valid. If there are characters already located in one of the coordinates, this will return false and the move will not be made. If it is a valid move, then the grid will call update and the correct characters will be placed.

Before each call of Clockwise or Counterclockwise, the grid will first set the existing coordinates of the block to “space” to prevent errors in conversion from one rotation to another. Since the block will have originally passed a check in the previous manipulation, the four characters can be set to space without worrying. This step ensures that each rotation does not leave any unwanted characters on the grid. The program proceeds as usual.

Upon each call of Drop, the grid function will also check if a row has been filled through the vector. If it has, then the grid will call RowClear() which will also update the score of the game and check if any blocks have been destroyed using the blockCell structure. The Drop function is the user’s way of communicating to the program that he is done with the current block and that the next one be brought out. Even if the user has reached a position where no further manipulation is possible, **he will have to** call drop to have access to the next block.

The rules of the game are that of a standard Tetris game, as soon as the player reaches the top of the grid, the game will end.

The emphasis of the design is on ensuring that we run into as few bugs as possible. Employing the check after each manipulation is key because it helps us avoid having to undo any moves on the Grid and every change to the Grid that we do finally make is final and need not be changed again.

|  |  |  |  |
| --- | --- | --- | --- |
| Group Members | Samatar Abukar | Lester Lim | Pranav Tripathi |
| Responsibilities: | -Block (abstract class)  -Sblock  -Iblock  -Tblock | -Zblock  -Jblock  -Lblock  -Oblock | -Level  -GridDisplay\*(Group works on this together)  -BlockCoord  -BlockCell |
| Team Functions: | Random, NoRandom, Sequence, I,J,L,Hint | | |
| Projected Completion Date: | December 2nd | December 2nd | December 4th |