# Features Implemented

We have implemented all the functionalities specified in the quadris.pdf assignment page. In addition, we incorporated some bonus features.

The following additional commands are supported:

- **rename oldname newname** changes the name of the command from oldname to newname. It has no effect if oldname is not an existing command or newname already exists as a command

- **hold** holds the current block so that it can be used later. If a block is already held, the held block becomes the current block. After this command is issued, the drop command must be issued before this command has any effect again.

- **skip** skips the current block if it is skippable.

The following additional options are supported on the command line:

- **-rename** allows users to modify the names of commands using the rename command

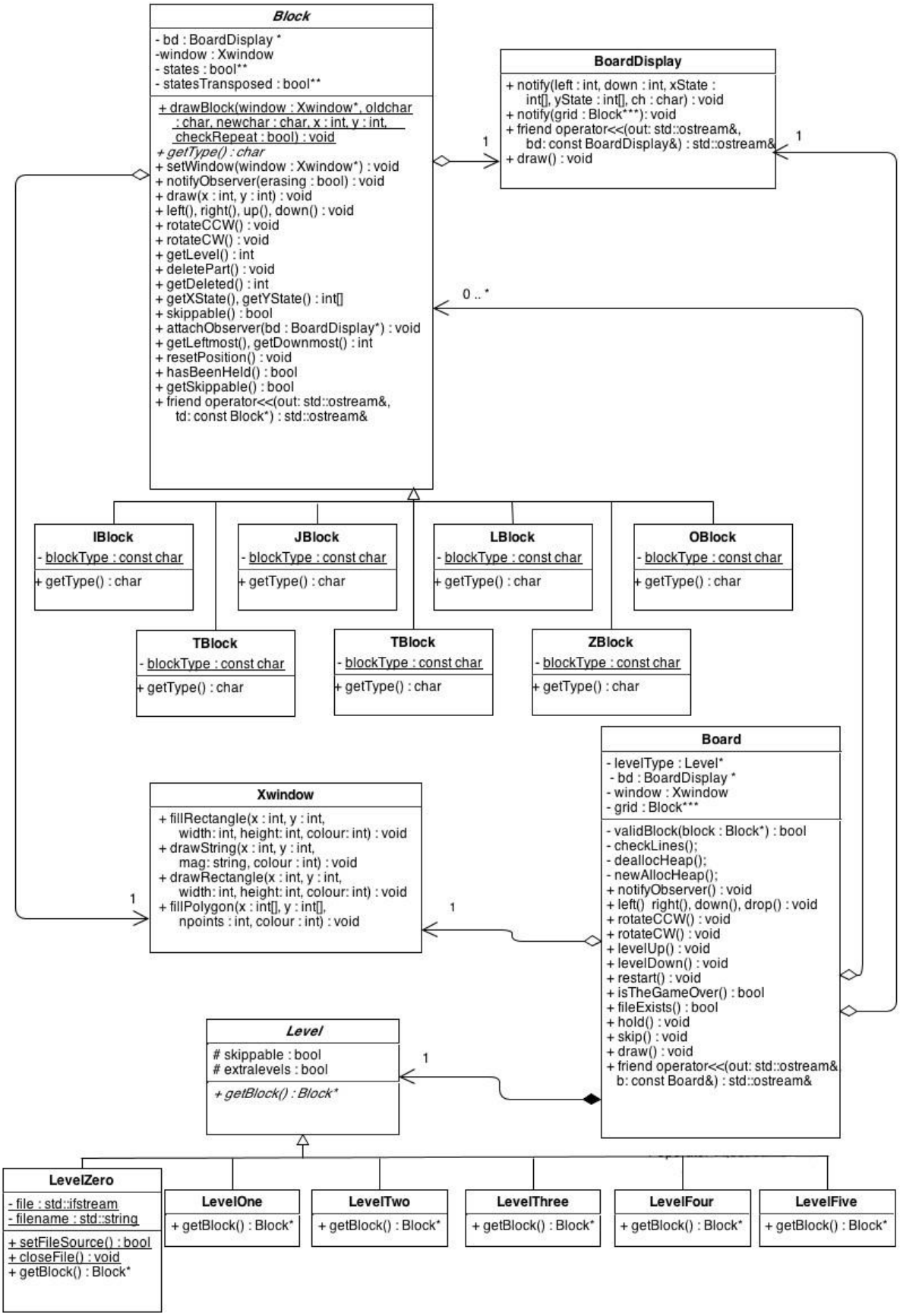
- **-extralevels** allows the game to be played with 6 levels instead of 4. Level 0 remains the same as the non-bonus program. Level 1, level 2, and level 3 of the non-bonus program becomes level 2, level 3, and level 4 respectively. Level 1 now cannot generate S and Z blocks and generates all other block types with equal probability. Level 5 generates S, Z, and T blocks with equal probability and cannot generate any other block.

- **-hold** allows blocks to be held using the hold command

- **-skip** allows blocks to be skipped using the skip command. Skippable blocks are generated at random. Without the -extralevels argument specified, the probability that a block will be skippable in level 0, 1, 2, and 3 are –, –, –, – respectively. With the –extralevels argument specified, the probability that a block will be skippable in level 0, 1, 2, 3, 4, and 5 are –, –, –, –, –, and – respectively.

**Note:** By default, none of the behaviour specified by the above command line options is applied.

In addition, we have modified the Xwindow class to make the blocks appear 3-D on the window. We accomplished this by drawing the border of the block using trapezoids that are of a different shade of colour than the rest of the block.


Block
- bd : BoardDisplay *
-window : Xwindow
- states : bool**
- statesTransposed : bool**
+ drawBlock(window : Xwindow*, oldchar : char, newchar : char, x : int, y : int, checkRepeat : bool) : void
+ getType() : char
+ setWindow(window : Xwindow*) : void
+ notifyObserver(erasing : bool) : void
+ draw(x : int, y : int) : void
+ left(), right(), up(), down() : void
+ rotateCCW() : void
+ rotateCW() : void
+ getLevel() : int
+ deletePart() : void
+ getDeleted() : int
+ getXState(), getYState() : int[]
+ skippable() : bool
+ attachObserver(bd : BoardDisplay*) : void
+ getLeftmost(), getDownmost() : int
+ resetPosition() : void
+ hasBeenHeld() : bool
+ getSkippable() : bool
+ friend operator<<(out: std::ostream&, td: const Block*) : std::ostream&
BoardDisplay
+ notify(left : int, down : int, xState : int[], yState : int[], ch : char) : void
+ notify(grid : Block***): void
+ friend operator<<(out: std::ostream&, bd: const BoardDisplay&) : std::ostream&
+ draw() : void
1
1
0 .. *
IBlock
- blockType : const char
+ getType() : char
JBlock
- blockType : const char
+ getType() : char
LBlock
- blockType : const char
+ getType() : char
OBlock
- blockType : const char
+ getType() : char
TBlock
- blockType : const char
+ getType() : char
TBlock
- blockType : const char
+ getType() : char
ZBlock
- blockType : const char
+ getType() : char
Board
- levelType : Level*
- bd : BoardDisplay *
- window : Xwindow
- grid : Block***
- validBlock(block : Block*) : bool
- checkLines();
- deallocHeap();
- newAllocHeap();
+ notifyObserver() : void
+ left() right(), down(), drop() : void
+ rotateCCW() : void
+ rotateCW() : void
+ levelUp() : void
+ levelDown() : void
+ restart() : void
+ isTheGameOver() : bool
+ fileExists() : bool
+ hold() : void
+ skip() : void
+ draw() : void
+ friend operator<<(out: std::ostream&, b: const Board&) : std::ostream&
Xwindow
+ fillRectangle(x : int, y : int, width: int, height: int, colour: int) : void
+ drawString(x : int, y : int, mag: string, colour : int) : void
+ drawRectangle(x : int, y : int, width: int, height: int, colour: int) : void
+ fillPolygon(x : int[], y : int[], npoints : int, colour : int) : void
1
1
Level
# skippable : bool
# extralevels : bool
+ getBlock() : Block*
1
LevelZero
- file : std::ifstream
- filename : std::string
+ setFileSource() : bool
+ closeFile() : void
+ getBlock() : Block*
LevelOne
+ getBlock() : Block*
LevelTwo
+ getBlock() : Block*
LevelThree
+ getBlock() : Block*
LevelFour
+ getBlock() : Block*
LevelFive
+ getBlock() : Block*

# Program Design

The Block class is responsible for the movement, rotation, and properties (such as the level it's generated in and whether it's skippable or has been held) of a block. The Block class is abstract, and seven concrete classes inherit from it, representing the seven types of blocks in the game. The BoardDisplay class stores a 2-D array of characters representing the current board, and is responsible for printing the contents of the current board to standard output and to the graphical display. The Level class is responsible for generating blocks. The Level class is abstract, and concrete classes representing the various levels of the game inherit from it. The Xwindow class is responsible for graphical output. The Board class generates Block objects using the Level class, and calls the appropriate functions in the Block class to manipulate the current block. The Board class also stores a 2-D array of Block pointers that represent the blocks that have been dropped. Thus, the Board class is responsible for line clearing and scoring. Board's operator << prints the scoreboard, and calls BoardDisplay's operator << to print the current board and Block's operator << to print the next and hold blocks. Similarly, Board::draw draws the scoreboard to the graphical display, and calls BoardDisplay::draw to draw the current board and Block::draw to draw the next and hold blocks. The main program, quadris.cc, is responsible for getting input from the user and calling the appropriate methods in the Board class based on the input. After each command, the main program calls Board's operator << and draw method. The main program is also responsible for processing options specified by command line arguments.
We applied the **factory method** design pattern to generate new blocks. The abstract Level class contains an abstract method getBlock. Each concrete subclass of Level implements the getBlock method based on the characteristics of that level. A polymorphic pointer to a Level object is stored in Board. We also applied the **observer** pattern. BoardDisplay is an observer of Block. When the positions occupied by the current block are changed, Block notifies BoardDisplay of its new positions. In addition, BoardDisplay is also an observer of Board. When a block is dropped, Board notifies BoardDisplay the new contents of the board since lines may be cleared.

For example, a typical command (take "left" for example) is executed as follows:

- The command is parsed and Board::left() is called
- The board's observer BoardDisplay is notified of the position change (erase).
- Board::left() calls Block::left() and moves the block temporarily.
- Board::validBlock() is called to check the validity of the new position of the block
- If it is invalid, Block::right() is called to move back to original position.
- The board's observer BoardDisplay is notified of the final position, (redraw)

And a "drop" command is processed like this:

- The command is parsed and Board::drop() is called
- The board's observer BoardDisplay is notified of the position change (erase).
- The Block::down() and Board:: validBlock () is called in succession until a false is returned.
- Block::up() is called to move it back into the last valid position possible

- All of the current block's positions is permanently stored into the 2d array grid (of Block pointers)
- Board::checkLines() is called to check for filled lines, to deallocate completely-deleted Blocks and to compute scores
- Level::getBlock() is called to receive a new block from the factory.
- The board's observer BoardDisplay is notified of the final position, (redraw)

It is also worthwhile to note that BoardDisplay is highly coupled with board and block, with the reason being that it was designed to be an extension of the two classes, in which they can be drawn. The other classes in our code have fairly low coupling, and good cohesion, as they each serve an individual purpose and operate largely independently of each other's implementations.

# UML Changes

There are several differences between our original UML and design and our final UML and design. We changed the name of the TextDisplay class to BoardDisplay because the BoardDisplay class prints to the graphical display in addition to standard output. We originally planned to call methods of Xwindow in different methods of Board similar to notifying TextDisplay, but realized that doing so would complicate the code and result in unnecessary redrawing. Variables that represent previous statuses are stored in Board and BoardDisplay to prevent unnecessary redrawing. Block's operator << and BoardDisplay::draw were added for easier output. We overloaded BoardDisplay::notify as we realized that BoardDisplay needs to observe Board when blocks are dropped and lines may be cleared. We made Block::notifyObserver public as we realized a Block should not notify BoardDisplay if it is moved to occupy invalid positions. The Board class checks for the validity of the new location (in bound and not previously occupied), and calls the reverse method (e.g, call left then right) if necessary. A parameter was added to Block::notifyObserver to distinguish drawing versus erasing. We also realized more getter methods in Block were needed for Board to check the validity of the location and notify BoardDisplay. We originally planned on having fields in Block representing the positions relative to the bottom left coordinate for each rotation state, and write assignment statements for them. In our final design, 2-D arrays of bool representing the rectangle enclosing the block were added. True represents that the location is part of the block and false represents it is not. When rotation occurs, they are updated using loops. We also realized the notifyObserver and draw method can be put in the base Block class to reduce duplication. In addition, methods were added to signify game over to quadris.cc (Board::istheGameOver), check for missing file (Board::fileExists), and open and close file in level 0 (LevelZero::setFileSource and LevelZero::closeFile). Some methods in our original UML (Level::getLevelNumber, Block::undraw, Board::up, and Board::undraw) were deemed unnecessary and removed. Some differences between the original UML were due to bonus features. Block::drawBlock is static, and was added to draw a single box on the window. Since the colouring of our blocks is complex, it prevents the duplication of large amounts of code. Xwindow::drawRectangle and Xwindow::fillPolygon were also added for fancier graphical output. Block::hasBeenHeld and Block::getSkippable were added for the skip and hold features. Additional subclasses of Level were also added.