Quadris

Requirements & Planning

This project hopes to accomplish being a Quadris game.

Break down:

Finished: July 20
Create the Field class that declares the fields and implement the methods:

- constructor
- left
- right
- down

Finished: July 21
Create a Tetromino abstract class that declares the fields and a set position virtual method. Create subclasses I,J,L,O,S,Z,T blocks. Implement methods:

- constructor
- left
- right
- down
- clockwise
- counter-clockwise
- getC: get letter of the Block (i.e. 'S' is for SBlock)
- getx: x-axis position
- gety: y-axis position

Finished: July 23
Create the main function that starts a new game using level 0. Able to display game board (15x10). Implement the user commands. Implement a drop method in the Tetromino class. Create a function that determines if a Tetromino has reached the bottom of the game board. Implement rotation methods in Field class.

Finished: July 24
If next Tetromino is to be used, make the current Tetromino block fixed in the game board. Create randomization of probability for next block depending on the level of game. Able to take in commands: restart, levelup and leveldown. Check for wall boundaries so current block

cannot go past the left and right of grid. Also able to not conflict with other blocks on the game board. Able to determine number of lines cleared. Determine if game over when beginning block at default position has a conflict with another block on game board.

Finished: July 25
Be able to repeat the sequence of blocks in sequence.txt if reached end-of-file. Create the 3 other level difficulties that depends on the policy for selecting the next block. Make the command interpreter to be able to distinguish commands from other commands need to be entered (i.e. lef is enough to distinguish the left command from the levelup command). Able to calculate scores and hi-scores. Remove a row if a line is cleared from the game board. Able to determine next block and display them too.

Finished: July 26
Be able to move all fixed blocks on the grid lower if a line is cleared. Reserved 3 rows at top of game board and edited all blocks to rotate properly given the reserved spaces. Create functions that are able to calculate scores depending on how many lines is cleared and level difficulty. Ask user if they would like to exit game when they get a game over, if continuing then start new game. Created a command where it takes the number of steps to do one action(i.e. 3right or 3 right).

Finished: July 27
Create a command-line interface that includes:

- -text
- -seed xxx
- -scriptfile xxx
- -startlevel n

And implement these actions.

Finished: July 29
Add a graphical game board using XWindow. That includes different coloured tetrominoes to be able to distinguish from other blocks.

Completed – If Things Go Wrong section

- can do block rotations
- program produces text output and graphics
- four level of difficulty implemented
- recognize the full command syntax
- score calculated correctly

Design

- Declaring and initializing variables, functions, classes
- Display game visually
- Infinite loop in main.cc where the game takes place
  - Receives input from user and displays the playing area and information
  - Input = left/right/down
    - Moves the current block one cell to the left/right/down
    - If this is not possible (left/right/down edge of the board, or block in the way), the command has no effect
  - Input = clockwise/counterclockwise
    - Rotates the block 90 degrees clockwise
    - If the rotation cannot be accomplished without coming into contact with existing blocks, the command has no effect
  - Input = drop
    - drops the current block
    - It is (in one step) moved downward as far as possible until it comes into contact with either the bottom of the board or a block
    - This command also triggers the next block to appear
    - Even if a block is already as far down as it can go (as a result of executing the down command), it still needs to be dropped in order to get the next block
  - Input = levelup/leveldown
    - Increase/decrease the difficult level of the game by one
    - The block showing as next still comes next, but subsequent blocks are generated using the new level
    - If there is no higher/lower level, this command has no effect
  - Input = restart
    - Clears the board and starts a new game

Overview of Aspects

The way this project was implemented is as follows. The main.cc file takes care of general management, and delegates tasks to other components of the project.

First, we have our Field class, which should be thought of as the “playing field” of the game of Quadris. This playing field manages things related to the playing of the game. This includes keeping track of score, generating the next block, graphical display, etc. For example, it keeps track of the members level, score, highscore, activeBlock (the current tetromino), next (the next

tetromino), grid, etc. It also contains the methods essential for the program to run, such as checkLoss() which checks if the user has lost, and completeLines() which checks if any lines have been cleared, clears them from the grid, and returns the number of lines cleared. In addition, since the Field class contains the grid, and thus has direct access to it, it also contains tetromino maneuvering methods such as left(), right(), clockwise(), etc because the valid execution of these actions are restricted by how the grid is filled at that time.

The Tetromino class is an abstract class because it contains the pure virtual function setPos(int o), which sets the positions of the four blocks that make up a tetromino. It makes sense for it to be abstract because a Tetromino doesn't have a tangible shape, and thus cannot exist as an object, but only as a classification of objects. We then have each of the seven possible tetrominos inheriting from the Tetromino class, each with their own specific implementations of the setPos(int o) method.

Answers to Questions

Question 1: How could you design your system to make sure that only these seven kinds of blocks can be generated, and in particular, that non-standard configurations (where, for example, the four pieces are not adjacent) cannot be produced?

To ensure only these 7 kinds of blocks will be generated, we create a finite list of them, and generate configurations from this list. This way, only valid blocks can be generated.

Question 2: How could you design your system (or modify your existing design) to allow some game levels to occasionally generate transparent blocks (which can overlap other blocks), or skippable blocks (if you don’t like it, you can skip to the next block)? Aside from these properties, transparent and skippable blocks behave like regular blocks. Could a block be both transparent and skippable?

This can be done by adding a field in the Tetromino class which represents whether or not a tetromino is transparent or skippable. The driver program would also have to be modified to accommodate the implementation of these properties. Yes, a block could be both transparent and skippable as they would be represented by separate fields.

Question 3: How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?

The program could be designed such that the game driver which generates which block is next is implemented using private functions. Thus, clients do not have to know what is going on behind the scenes, and recompilation can be minimized by hiding private members and functions in another file.

Question 4: How could you design your system to accommodate the addition of new command names, or changes to existing command names, with minimal changes to source and minimal recompilation? (We acknowledge, of course, that adding a new command probably means adding a new feature, which can mean adding a non-trivial amount of code.) How difficult would it be to adapt your system to support a command whereby a user could rename existing commands (e.g. something like *rename counterclockwise cc*)?

At the moment, all of the possible commands are stored in an array, and an "interpreter" function eliminates possibilities based on input, and executes a command if it is the last remaining possibility. To add new commands, the array must simply be expanded to allow them. To rename a function like "counterclockwise" to "cc", we would modify the array of commands, such that the "counterclockwise" entry is just "cc".

Other Questions

1. What lessons did this project teach you about developing software in teams? If you worked alone, what lessons did you learn about writing large programs?

This project taught us that when working on large projects, tasks must be effectively identified and delegated. Having both group members work on the same file was problematic because changes are difficult to keep track of. We dealt with this program by using a log file, to keep track of changes and of which features had been completed, and which still needed to be done.

Essentially, we learned that organization is key.

2. What would you have done differently if you had the chance to start over?

We would've implemented the Field class using the singleton pattern, as it would make sense to only have one game running at a time.

Tetromino maneuvering functions may have made more design sense to not be in the Field class, and instead be in the Tetromino class. However, the restrictions caused by boundaries and other blocks in the way require the grid to be checked, and the grid is private to the field

class. Perhaps we would give the Tetromino class access to the grid by using friend methods, or maybe the Observer Pattern.

In addition, our Quadris project has a segmentation fault when exiting, after losing a game. We are unsure of what exactly causes this but given the chance, this would be fixed, of course.