Demo Plan

Kefan Cao, Jacob Mausberg

August 13, 2021

1 Basic Game play

The biquadris game supports every one of the commands as follows,

- left - Moves the current block of the current player to the left by one cell. If the block is in a position such that it can no longer move to the left by one cell, then the command is ignored and the block stays where it is.
- right - Moves the current block of the current player to the right by one cell. If the block is in a position such that it can no longer move to the right by one cell, then the command is ignored and the block stays where it is.
- down - Moves the current block of the current player down one cell. If the block cannot move down as a whole, i.e, there are cells preventing it from moving down, the command will be ignored.
- drop - The drop command drops the current block down by as many cells as possible until it is in a position where it can no longer move down as whole, then the control is transferred to the opponent's board and now it becomes the opponent's turn.
- clockwise - The clockwise command rotates the current block clockwise while maintaining the bottom left corner.
- counterclockwise - The counterclockwise command rotates the current block counterclockwise while maintaining the bottom left corner.
- levelup - The levelup command brings the current level to one level above it. If the current level is at the maximum level, the command will have no effect.
- leveldown - The leveldown command brings the current level to one level below it. If the current level is at the minimum level, i.e, level 0, the command will have no effect.
- norandom file - The norandom command will require the user to enter a file name consisting of block types. This command is only valid during levels 3 and 4, and will continuously read in from the file as an input for blocks. If it reaches the end of the file, it continues from the start of the file and reads it in again from the beginning.
- random - The random command will restore randomness in the current level, valid during levels 3 and 4. It is to undo what the norandom command did.
- sequence file - The sequence command will require the user to enter a file name consisting of commands, and that file becomes the input source and the program will execute all the commands in that file and return control back to cin.
- I, J, L, etc. - These commands switches the current block to the respective block types. Useful during testing.

- restart - Restarts the game for both players and not for a single player, because once either player 1 or 2 loses, the game becomes single player for the remaining player. The only time restart can be called is by the remaining player.
- hint - the hint command will display a list of possible commands for the player.

Please note that commands like left, right, clockwise, counterclockwise, levelup, leveldown, sequence supports command execution multiplier; while other do not. The command execution multiplier is able to multiply a particular command

2 Game Flags

The biquadris game supports every one of these flags.

- -text - The text flag runs the program in text-only mode. With this flag, your program will only print the text display and nothing else.
- -seed xxx - The -seed xxx specifies a seed to run the block generator with. Useful for testing as it creates the blocks in the same order every time.
- -scriptfile1 xxx -Uses xxx instead of sequence1.txt as a source of blocks for level 0, for player 1.
- -scriptfile2 xxx -Uses xxx instead of sequence2.txt as a source of blocks for level 0, for player 2.
- -startlevel n -Starts the game in level n. The game starts in level 0 if this option is not supplied.
- -help - Prints out a list of possible commands for future playing.
- -textGUI -Enables the textGUI bonus.

3 The Demo

The demo plan will consist of the following. If you're running this program over ssh connection and do not want the lag, you can run this program with the flag "-text" so that the testing process is faster for you. ☺

- Demonstrate that block movements work and stops on the border.
  - Run the program in level 0, since the block will be generated on the top left corner, if you input the command 'left', it will stay in the top left corner and not go off of the screen. Input "13right" (the macro command for moving a block 13 times to the right.), you will see that the block is now at the far right corner of the board, which means that left block and right block movements won't go off the board and are all valid. Short circuit evaluation is used here. If you input something like "10000left", the evaluation will at most execute "10left" because that's the most a single block could move to the left consecutively. Similarly with the other macro commands. Now, input the command "20down", it will drop the block down without moving off the board. Enter "2drop" to drop the current block and the opponent's block. To see that the blocks will stay on top of each other. Enter the command "10right", "drop", you will see that the newly generated block now lays on top of the block you previously dropped. A new block is now generated, let's verify that the counterclockwise and clockwise command do not rotate the blocks when they can't be rotated, i.e, there's no room to maintain the bottom left corner. Force the current block to become the I block (since that one is the easiest to verify, please feel free to try other blocks.), enter 'I' to force the current block into the I block. Now, input the "clockwise" command, you will see that the I block is now rotated clockwise while maintaining the bottom left corner. Input "10right" to move the block to the far right of the board. Since rotating the block will not have enough room to maintain the new position and the bottom left corner, the block is not rotated. Enter "clockwise" or "counterclockwise" to verify this. But as soon as there is enough room, say if you enter "4left" then followed by "clockwise", the block will now rotate.

Awesome, we've now verified basic block movement!! You can verify that it works for other levels if you'd like, but it is fairly trivial that it will.

- Demonstrate that each of the commands work
  - Point 1 verified that the command, left, right, clockwise, drop works as per required. We will now verify that other commands will also work.
  - To verify the counterclockwise rotation works, simply run the program and type in the type of the block you'd like to see rotated, say the I block, then input "counterclockwise" for the rotation. The block rotations are level independent, so it is not necessary to test it for all levels.
- Demonstrate that rows clear.
  - Run the program, then input the command "sequence rowClear.in", which will set you up for row clearing, then enter the command "drop" to see the row get cleared.
  - Additionally, to verify that clearing a middle row will not mess up anything, re-run the program, the input the command "sequence rowMiddleClear.in" which sets you up to clear a middle row, then just input "dr" (short for drop). Which clears a middle row correctly.
- Demonstrate that levels 3 and 4 generate heavy blocks.
  - Run the program with the "-startlevel 3" flag (or 4), then just enter the command "left", "right", "clockwise", "counterclockwise" to see the heavy effect take place. Please note that if a multiplier was combined with a command, the block will only drop one row.
  - If you attempt to move a heavy block, either left, right, clockwise, or counterclockwise, and there isn't enough room for it to move down a row, then the block is considered dropped. To verify this, Run the program with "-startlevel [3|4]" Input "I", then input "20down", then input "left". Since the block is at the very bottom (i.e, there is no room for the block to move down one row) the block is considered dropped. You can see this as a block has generated on the top left corner of the board as your current block.
- Demonstrate that levels generate the correct blocks with the right probability for each level.
  - This can be verified by checking the level[1-4].cc, namely the make function which has a discrete distribution type set according to each level's required probability.
- Demonstrate that level 0 takes in a file as its source of blocks
  - The default level 0 "sequence1.txt" and "sequence2.txt" files are set. Feel free to view the contents of those files to make sure it is generated correctly on the board. Run the program and input "10drop" which will drop 10 blocks, 5 for each board. It should be enough to verify that it works. If the end of file is reached, the program automatically starts at the beginning of the file and reads in from there. It is easy to verify because the default "sequence1.txt" file has only one letter, 'I' in it. (feel free to verify this)
  - To change the default file, run the program with the "-scriptfile[1 or 2] xxx" where "xxx" is the name of the script file. One has been provided called "blockFile.in", which contains each block type exactly once.
- Demonstrate that level 4 generates the brown block if a row hasn't been cleared after 5 moves.
  - Run the program with the flag "-startlevel 4" then input the command "sequence brownGenerate.in", which will drop the blocks 8 times. Then enter "drop" to see the brown block generated on the fifth drop for board 1, then enter "drop" again to see the brown block generated on the fifth drop for board 2.

  - To see that clearing a row resets the counter for block generation (that is, the player gets 5 moves after a row is cleared before a brown block is generated). Run the program with "-startlevel 4" flag then input the command "sequence brownGenLate.in", which will set you up for a row clear. Upon dropping, it will be the third block dropped, input "drop", it will clear the row, then input "3drop", which brings you to your fifth block. Dropping this block will not generate a brown block for player 1, because player 1 has just cleared a row 2 moves earlier. Input "drop" to see this. Input "drop" again to see that a brown block has been generated for player 2. Now, if further rows do not get cleared, after another 5 drops, the brown block should be generated for player 1. Input "4drop", to be on the fifth drop since row clearing for player 1, then input "drop", you will see the brown block is now generated since player 1 has not cleared any rows for five moves. To verify that the brown block gets generated on every multiple of five moves, drop the blocks in a manner such that neither players lose. After 10 drops, the brown blocks should be generated.

- Demonstrate that norandom command is able to switch level 3 or 4's block generation into a none random generation, and instead, generated by reading block types from a file continuously. And demonstrate that the random command is able to restore randomness.
  - To demonstrate, run the program with the flag "-startlevel 4" and input the command "norandom sequence1.txt" (we will use sequence1.txt because you've seen it's contents and since it only has one block, it is both easy to see and makes for a good edge case.) Do the command "4drop", which will drop the two blocks generated for each board by the random generator for level4. You should see that board 1 (the board that norandom was executed from) now have the I block as the the current and next block. Feel free to test that it will continue generating by inputting "10drop" to drop 5 blocks for each board. At the end, it should remain that the current and next block are both still I blocks.
  - To restore randomness, input "random" for the board that has norandom set, then execute the "drop" command, and you should see that the block now generated is no longer an I block (or maybe it still is because of the distribution, but a few more drops should fix this).

- Demonstrate that the GUI works and mirrors the text display.
  - If you have been walking through this demo plan without running with the -text flag, you should see that the GUI mirrors the text display.
  - If you have been running with the -text flag, please execute the next demo point without the -text flag.

- Demonstrate that clockwise rotates clockwise and counterclockwise rotates counterclockwise while maintaining the bottom left corner.
  - Run the program and input the block you'd like to see rotated clockwise or counterclockwise, then input "clockwise" or "counterclockwise"

- Demonstrate that multiplicity can be applied to commands such as left, right, down, drop, sequence, levelup, leveldown, clockwise, counterclockwise.
  - Block movements for the sequence files executed earlier have utilized multiplicity of commands. To further ensure each of these multiplicities work, please execute each of these commands with a desired multiplicity. For the "sequence" command, please use the input file "seqDemo.in" as it contains a single command "right" to avoid verification on your part; however, if you'd like, feel free to make a sequence of your own. Input "4sequence seqDemo.in" to move the current block 4 times to the right.
  - Feel free to try the macro command on 'levelup' and 'leveldown' if you'd like, but they are fairly trivial.

- Demonstrate that the scoring for each level is calculated correctly.

- For simplicity and for ease of verification on your part, we will be constructing a sequence that will clear 1 row and 2 blocks for each level. The calculation is as follows. Run the program with the level you'd like to verify the score for, using the flag "-startlevel n", then enter the command "sequence scoreVerify.in", which will set you up to clear exactly 1 row and 2 blocks. Input "drop" to see the score reflected for player 1. See below for how scores are calculated and that the scores are indeed correct. **Where Level is the level of the block generated.
  - Formula : $(\text{Current Level} + \text{Rows cleared})^2 + \#\text{ of blocks cleared} \times (\text{Level} + 1)^2$
  - Level 0: $\rightarrow (0+1)^2 + 2 \times (0+1)^2 = 1 + 2 = 2$
  - Level 1: $\rightarrow (1+1)^2 + 2 \times (1+1)^2 = 4 + 8 = 12$
  - Level 2: $\rightarrow (2+1)^2 + 2 \times (2+1)^2 = 9 + 18 = 27$
  - Level 3: $\rightarrow (3+1)^2 + 2 \times (3+1)^2 = 16 + 32 = 48$
  - Level 4: $\rightarrow (4+1)^2 + 2 \times (4+1)^2 = 25 + 50 = 75$
- Further checks can be done to make sure that multiple line clears also affect the score, but for simplicity and verification, they will not be demonstrated.

- Demonstrate that special actions work on both the text display, the GUI, and the textGUI (bonus).
  - Run the program in level 0, 1, or 2. (not 3, 4 so that heaviness can be verified. Feel free to verify other special actions in level 3 and 4 though!) Input the command "sequence specialAction.in", which will set you up for a special action for player 1 to be done on player 2. Input the special action you would like to test, either "blind", "heavy", or "force", when doing force, you will need to enter the type of block you wish to force your opponent's block to.
  - For the blind special action, the opponent's board is temporarily blinded for the duration of their move, after the drop the block, their board is unblinded.
  - For the heavy special action, the opponent's current block, becomes a heavy one and will move down by one row every time it moves left, right, clockwise, or counterclockwise. The block reverts back to a normal one when the opponent has finished his turn.
  - For the force special action, the opponent's current block becomes the block of your choosing.

- Demonstrate that the next block is indeed the next block generated (trivial).
  - By now, this feature should be evident. But to test it, simply start at level 1 with the flag "-startlevel 1", then input "drop" until you're convinced the next block is indeed the next block generated.

- Demonstrate that there's no leak in -text only mode.
  - Since we are using automatic memory management, there should not be any memory leaks in our code. We've provided a sequence file which executes every possible command in text mode to ensure there are no memory leaks.

- Demonstrate that the command line arguments(flags) all work as they should.
  - By now, you should have executed the program using every one of the flags but the -scriptfile1, -scriptfile2, and -seed flag. To test these flags, follow these instructions below.
  - To test the "-scriptfile1 xxx" and "-scriptfile2 xxx" for level0, one simple test is to swap sequence1.txt with sequence2.txt, but we've also provided a file called "blockTypes.in" which contains one occurrence of every block type there is. Feel free to run the program with either swapping or using the providede "blcokTypes.in" file to test that blocks are indeed generated in order specified by the files.
  - To test the seed flag, select some integer of your liking, input it along with the flag. This will seed the program such that the levels will start at that seed. To test this, run the program with the "-seed n", where n is some integer and the flag "-startlevel [1 |2 |3 |4]", not 0 because level0 is non-random. Drop a few blocks, then exit the program by inputting EOF. Run the program again with the same flags, you shall see that the blocks generated for both turns are identical. Of course, feel free to change the seed integer to see blocks getting generated in a different order.

- Demonstrate that the restart command works.
  - The restart command can be typed in at any time during the game to restart the game. Run the program and input "10drop" to drop a few blocks, then input "restart" to see the board get cleared.
  - The restart command also updates the high score if a high score was ever reached. To see this, run the program in any level you prefer, then input "sequence rowClear.in", which sets you up for a row clear. Clear the row by inputting "drop". See the score is updated to reflect this drop. Then, input "restart" to see the board reset and the high score updated.
- Demonstrate that the game switches to single player mode when one player loses.
  - Run the program in level0 with the provided sequence1.txt and sequence2.txt files. Then execute the command, "sequence player1Win.in" to set up the scenario where player 1 wins. Enter "drop" one last time to make player 2 lose. You will be prompted to either continue playing or start a new game. Please continue the game by pressing "C". The game now continues and switches to single player mode. All controls are transferred to player 1. The game will continue to be in play until player 1 loses. Enter "10drop", to setup the scenario where player 1 will lose, then enter "drop" to see the end screen prompting you to start a new game.
- Demonstrate that high scores are updated when game over.
  - Run the program and execute the command 'sequence rowClear.in' to have player 1 clear a row, then input 'drop 20' to drop 20 blocks which should cause game over. Input 'G' for a new game to instantiate a new game. The high score shall be updated.
- Demonstrate the end game screen.
  - Since biquadris is score based, so the winner is ultimately dependant on the finishing score for each of the players. The player with the higher score is the ultimate winner. To test that the game end works properly, follow the following steps.
  - Run the program, input "sequence gameEnd.in" which brings you to prepare to clear one row. Enter "drop" to clear this row. Clearing this row brings up player 1's score up from 0. Force player 1 to lose first by inputting "sequence player1L.in". It forces player 1 into a losing position. Input "drop" to see player 1 lose. The message displayed will show what score player 1 finished with. Input "15drop" to force player 2 to lose. Since player 2 did not clear any rows, the game end message should say "Player 1 won."
- Demonstrate that the user mannual is available in the program. (Bonus)
  - To see the -help flag in action, run the program with the -help flag. To see a list of possible commands, enter "hint" to see.
- Demonstrate that the textGUI works and creates a pleasant user experience.
  - To view the textGUI, run the program is '-textGUI' option. Using this option, you will enable the textGUI mode. See the next section on game play instructions. This mode is extremely fun to play compared to the regular text mode as you can use the arrow keys to navigate over to where you'd like.
- Demonstrate that inputting invalid file names does not cause the program to crash.
  - Since Level 0 has default "sequence1.txt" and "sequence2.txt", those two files must be in the current directory for the program to run. If they are not present, then a message is outputted to the screen which says "Could not open input file for Level 0." The same goes for when the "-scriptfile[1|2] xx" flag is used, i.e. if the file does not exist, you will not be able to run the program. Feel free to verify.

  - For any commands that require a file such as "sequence" and "norandom", they will behave in a similar fashion as previously stated for Level 0. Feel free to verify this.

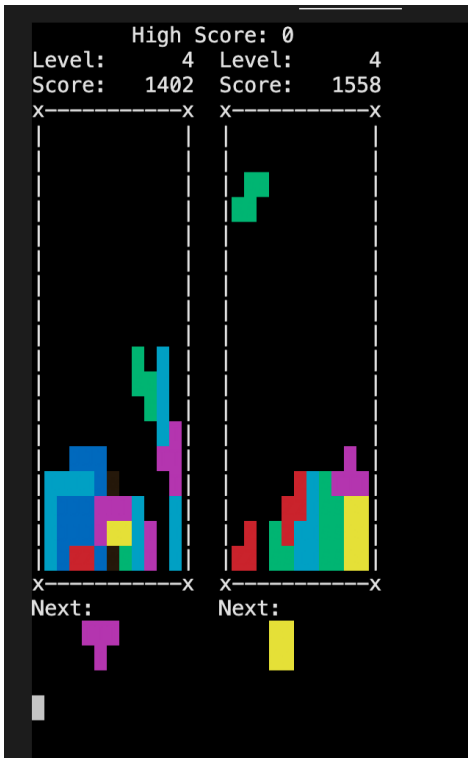
- Demonstrate that short hand notation for commands work.
  - If you have been inspecting the contents of the provided sequence files, it should be very evident that the short hand works, but feel free to try out any of the features in this demo in the short hand notation. Below are the minimal characters required to make each command recognized.
    * left = lef
    * right = ri
    * levelup = levelu
    * leveldown = leveld
    * clockwise = cl
    * counterclockwise = co
    * restart = re
    * hint = h
    * drop = dr
    * down = do
    * random = ra
    * norandom = n

4 Bonus: TextGUI made with ncurses

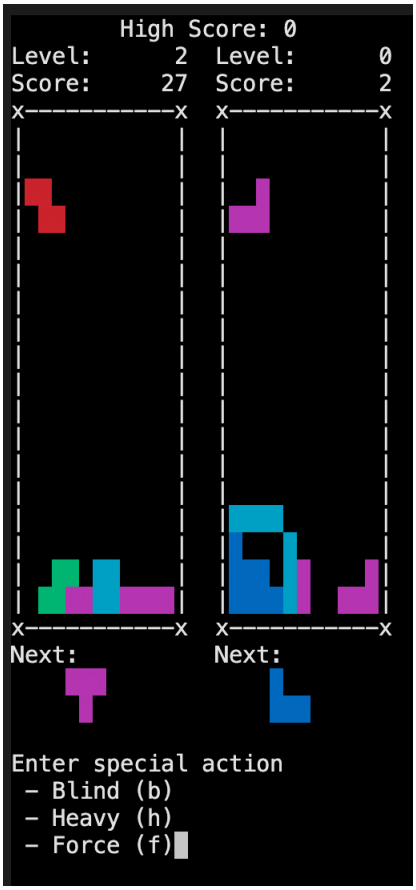
For the bonus feature, we created a textGUI that supports the following commands.

- ← = 'left'
- → = 'right'
- ↑ = 'clockwise'
- ↓ = 'down'
- Spacebar = 'drop'
- 'z' = 'counterclockwise'
- 'q' = Quit Game
- 'r' = 'restart'
- 0, 1, 2, 3, 4 = To switch to the desired levels.
- I, J, O, L, etc. = To switch the current block to the desired block.

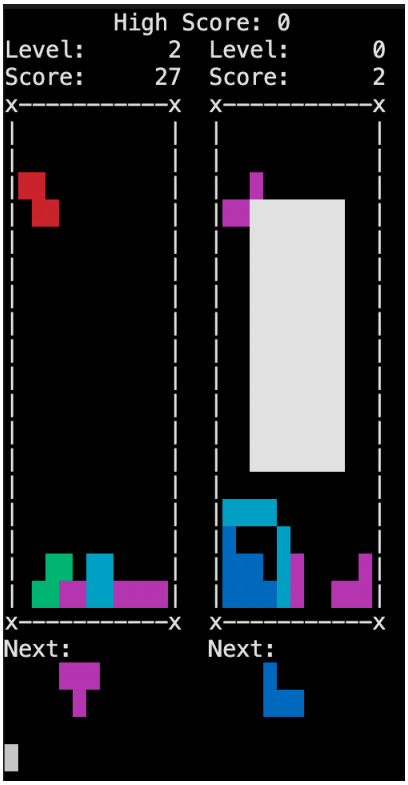
To play the game in the textGUI mode, run the program with the command-line argument '-textGUI'. The textGUI has also been formatted such that a color is related to each block. Below is an image of what that looks like.



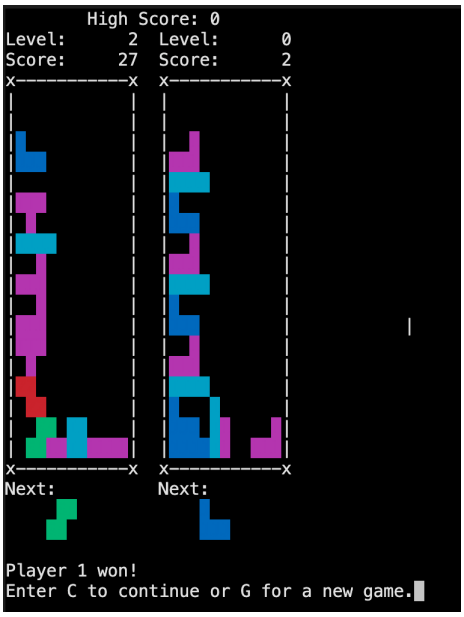
Notice that the board only updates the parts that are necessary and does not reprint to the screen. Triggering a special action leads to the following screen.



You may enter a key for a special action. Choose either 'b'(blind), 'h'(heavy), 'f' (force) options. Selecting the blind option creates a white block in the required position. See graphic below,



Upon game over, a prompt asks the remaining player if they would like to continue playing in single player mode or start a new game



This bonus feature was made using the ncurses library. Please note that the ncurses window does incur memory leaks for performance reasons as documented here. `https://invisible-island.net/ncurses/ncurses.faq.html#config_leaks`. However, the memory leak caused by ncurses is very tiny, so it can be disregarded.